**BILAN TP3**

%somme(A,B,S) : s = A+B

somme(A,B,S):-S is A + B.

%maximum(A,B,M), M = la plus grande valeur

maximum(A,B,M):-A>=B, M is A.

maximum(A,B,M):-A=<B, M is B.

%version2

maximum2(A,B,A):-A>=B.

maximum2(A,B,B):-A<B.

%sigma(N,Res): Res = N+....+3+2+1

sigma(0,0).

sigma(N,Res):-N1 is N-1,sigma(N1,R),Res is R+N.

%fact(N,F) : factorielle de N

fact(0,1).

fact(N,F):-N1 is N-1, fact(N1,F1),F is F1\*N.

%pgcd(A,B,Res) : le plus grand diviseur commun

%A>B, le pgcd sera le pgcd de A-B et B

%sinon, le pgcd sera le pgcd de A et B-A

pgcd(A,A,R):-R is A.

pgcd(A,B,Res):-A>B,M is A-B,pgcd(M,B,Res).

pgcd(A,B,Res):-A<B,M is B-A,pgcd(A,M,Res).

%somme2(BMin,BMax,S) : somme des valeurs entre BMin et BMax, bornes

%comprises.

somme2(BMin,BMin,S):-S is BMin.

somme2(BMin,BMax,S):-BMin<BMax, N is BMax-1,somme2(BMin,N,R),S is R+BMax.

% somme3(B1,B2,S) : somme entre B1 et B2, bornes comprise, B1 n'est pas

% forcément la plus grande valeur.

somme3(B1,B1,S):-S is B1.

somme3(B1,B2,S):-B1>B2,N is B1-1,somme3(N,B2,R),S is R+B1.

somme3(B1,B2,S):-B1<B2,N is B2-1,somme3(B1,N,R),S is R+B2.

%comportement des prédicat

= : unification et pas l’affectation

ex : M = 1 + 1, retourne M = 1 + 1

is : affecter une valeur numérique à une variable, en évaluant mathématiquement le résultat

ex : N is 1+1, retourne N = 2

== : tester l’égalité des deux arbres, égalité de chaque noeud et de chaque feuille

ex : X ==1, retourner false.

1==1, retourner true

x+x == x+x true

write() : applicable à une constante, une variable ou une chaîne de caractères entre guillements simples, nl pour passage à la ligne

ex : - write('bonjour'),nl,write('mengzi').

bonjour

mengzi

true.

% chaineConstruire(Lettre,N,C) : construire une chaîne C constituée de N

% fois la lettre Lettre

chaineConstruire(L,1,L).

chaineConstruire(L,N,C):-N>1,N1 is N-1,chaineConstruire(L,N1,C2),atom\_concat(L,C2,C).

%dernier : dernier élément de la liste

dernier(X,[X]).

dernier(X,[\_T|R]):-dernier(X,R).

**BILAN TP4**

%list1 : met en relation une liste non vide avec une liste cmoposée du

% premier élément del a première liste

list1([T|\_],[T]).

% list2 : met en relation une liste contenant au moins 2 éléments, avec

% une liste composé des 2 premiers éléments de al première liste.

list2([T1,T2|\_],[T1,T2]).

% perm12(L1,L2) est vrai si L1 et L2 sont identique à l'exception de

% leurs 2 premiers éléments qui sont permutés.

perm12([T1,T2|R],[T2,T1|R]).

%membre(X,L) indique si X est un élément de la liste L

membre([T|\_],T).

membre([\_T|R],N):-membre(R,N).

% dernier(X,L) : vrai si X est le dernier élément de la liste L supposée

% non vide

dernier(X,[X]).

dernier(X,[\_|R]):-dernier(X,R).

% rangpair(L,P) : vrai si P est la liste des éléments de rang paire de

% la liste L

rangpair([],[]).

rangpair([\_T],[]).

rangpair([\_,T1|R1],[T1|R2]):-rangpair(R1,R2).

% consn(N,L) : construire pour un entier n, la liste des n+1 premiers

% entiers dans l'ordre décroissant

consn(0,[0]).

consn(N,[N|R]):-N>0,N1 is N-1,consn(N1,R).

% occur(X,L,N) : calculer le nb N d'occurrences de l'élément X dans la

% liste L

occur(\_,[],0).

occur(X,[T|R],N):-X =:= T, occur(X,R,N1), N is N1+1.

occur(X,[T|R],N):-X \== T,occur(X,R,N).

**BILAN TP5**

%delMultiple : supprime tous les multiples d'un entier X dans une liste.

delMultiple(\_,[],[]).

delMultiple(N,[T|R],[T|R1]):-T mod N =:= 0, delMultiple(N,R,R1).

delMultiple(N,[T|R],L):-T mod N \==0, delMultiple(N,R,L).

%delegaux : supprimer tous les éléments égal à N dans une liste

delEgaux(\_,[],[]).

delEgaux(N,[T|R],[T|R1]):-T =:=N,delEgaux(N,R,R1).

delEgaux(N,[T|R],L):-T\==N,delEgaux(N,R,L).

% entre(Min,Max,L) : vrai si L est la lisste des entiers compris entre

% Min et Max.

entre(Min,Min,[Min]).

entre(Min,Max,[Min|R]):-Max>Min,N is Min+1, entre(N,Max,R).

%mot : représenté par la liste de ces lettres

mot([r,o,t,i,s]).

mot([l,o,i]).

mot([l,o,t,o,s]).

mot([l,o,t,s]).

mot([r,o,t,i,n]).

mot([l,o,i,r,s]).

% correct(L,X) : étant donné un mot, répond ok si le mot appartient au

% dictionnaire et/ou donne les mots du dictionnaire qui diffèrent du

% premier par une seule lettre en meme position

diff([T1|R],[T2|R]):-T1\==T2.

diff([T|R],[T|R1]):-diff(R,R1).

correct(L,ok):-mot(L).

correct(L,M):-mot(M),diff(L,M).

**BILAN TP6**

habite(paul , paris) .

habite(luc , paris) .

habite(marc , paris) .

habite(jean , toulouse) .

habite(fred , toulouse) .

habite(louis , toulouse) .

pratique(marc , foot) .

pratique(marc , rugby) .

pratique(marc , judo) .

pratique(paul , tennis) .

pratique(paul , foot) .

pratique(luc , rugby) .

pratique(luc , basket) .

pratique(jean , judo) .

pratique(jean , tennis) .

pratique(fred , basket) .

pratique(fred , rugby) .

pratique(fred , judo) .

collectif(rugby) .

collectif(foot) .

collectif(basket) .

collectif(hockey) .

individuel(tennis) .

individuel(golf) .

individuel(judo) .

rel1(X,Y) :- pratique(X,Y), collectif(Y) , ! .

rel2(X,Y) :- collectif(Y), pratique(X,Y) , ! .

%tous les parisiens

habite(X,paris).

%toutes les personnes de la base

habite(X,\_).

%tous les sports pratiqués par Marc

pratique(marc,X).

%tous les sports de la base

sport(X):-collectif(X).

sport(X):-individuel(X).

%tous les joueurs de rugby

pratique(X,rugby).

%un joueur de foot

pratique(X,foot),!.

%un parisien qui pratique par le rugby

habite(X,paris),pratique(X,rugby),!.

%un sport pratiqué par Fred

sport(X),pratique(fred,X),!.

%interpréter rel1(X,Y).

une personne qui pratique le sport collectif

%rel1 et rel2 ont ils la même interpretation

Non, rel1 : une personne qui pratique le sport collectif

rel2 : un sport pratiqué par une personne

%habite(X,paris),rel1(X,Y).

les parisien qui pratique le sport collectif

%tous les sports collectifs pratiqués à paris

%un parsien pratique le sport

parisienS(X,Y):-habite(X,paris),pratique(X,Y),!.

%un sport pratiqué à Y

sportP(X,Y):-sport(Y),parisienS(X,Y).

%vérifier les sports sont collectifs

collectP(X,Y):-collectif(Y),sportP(X,Y).

%tous les parisiens qui pratiquent un sport collectif

%une personne pratique le sportCollectif

pratiqueCollectif(X,Y):-pratique(X,Y),collectif(Y),!.

%tous les parsiens

parisienCollectif(X,Y):-habite(X,paris),pratiqueCollectif(X,Y).

%pour chaque sport un représentant d ce sport

%une personne qui pratique le sport

unJoueur(X,Y):-pratique(X,Y),!.

%pour tous les sports

representant(X,Y):-sport(Y),unJoueur(X,Y).

%pour chaque ville un représentant de cette ville et tous les sports qu’il aime

%un représentant de chaque ville

ville(paris).

ville(toulouse).

unHabit(X,Y):-habite(X,Y),!.

%pour chaque ville

villeRepresentant(X,Y):-ville(Y),unHabit(X,Y).

%tous les sports qu'il aime

representantVilleSport(X,Y,Z):-villeRepresentant(X,Y),pratique(X,Z).

**BILAN TP7**

%syracuse : vrai si S est le résultat de la suite de syracuse de M à l’étape N

pair(X):-X mod 2 =:= 0.

impair(X):-X mod 2 =:=1.

syracuse2(M,M,0).

syracuse2(M,S,N):-N>0,N1 is N-1, pair(M),syracuse(M,S1,N1),S is S1/2.

syracuse2(M,S,N):-N>0,N1 is N-1, impair(M),syracuse(M,S1,N1),S is ((3\*S1)+1).

%membre(N,L) : vérifier que N fait partie dans L

membre(N,[N|\_R]).

membre(N,[\_T|R]):-membre(N,R).

nonMembre(N,L):-membre(N,L),!,fail.

nonMembre(\_).

pair(X):-0 is X mod 2.

impair(X):-pair(X),!,fail.

impair(\_).

**BILAN Examen 2015**

%nb pair

pair(X):-0 is X mod 2.

%négation nb impair

impair(X):-pair(X),!,fail.

impair(\_).

%calculer le nb de nb pair et le nb de nb impair

nb([],0,0).

nb([X|L],P,I):-pair(X),nb(L,M,I), P is M + 1.

nb([X|L],P,I):-impair(X),nb(L,P,N),I is N + 1.

%listPair retourner les nb pair de la liste

listPair(0,[0]).

listPair(N,[N|LP]):-N>0,pair(N), M is N-2,listPair(M,LP).

listPair(N,LP):-N>0,impair(N), M is N-1,listPair(M,LP).

%retourner les nb pairs

nbPair(0,0).

nbPair(N,N):-N>0,pair(N),M is N-2,nbPair(M,M). %à vérifier

nbPair(N,Elt):-N>0,impair(N),M is N-1,nbPair(M,Elt).

**BILAN TD5 - TD7**

%calcule la longeur de la liste

long([],0).

long([\_T|R],N):-long(R,N1),N is N1+1.

%nuplet

nuplet(0,[]).

nuplet(N,[0|R]):-N>0,N1 is N-1,nuplet(N1,R).

nuplet(N,[1|R]):-N>0,N1 is N-1,nuplet(N1,R).

%tri par pivot

% décomposer : décompose une liste d'entieres suivant un entier pivot en

% deux listes formées respectivement, des entiers de la listes

% supérieurs ou égaux au pivot et ceux inférieurs au pivot

decomposer([],\_P,[],[]).

decomposer([T|R],P,[T|L1],L2):-T>=P, decomposer(R,P,L1,L2).

decomposer([T|R],P,L1,[T|L2]):-T<P,decomposer(R,P,L1,L2).

% ordonner : classe les éléments d'une liste d'entiers dans l'ordre

% croissant.

ordonner([],[]).

ordonner([T|R],LT):-decomposer(R,T,L1,L2),ordonner(L1,LT1),ordonner(L2,LT2),append(LT1,[T|LT2],LT).

%représentation d'arbre de recherche

abe(abe(vide,1,abe(vide,2,vide)),abe(vide,4,vide)).

%mirror d'arbre (erreur)

%mirroir(abe(G1,N1,D1),abe(G2,N2,D2)):-mirror(G1,D1),mirroir(G1,D1).

%estDansAbe : vrai si le nb N se trouve dans l'Abe

estDansAbe(N,abe(\_,N,\_)).

estDansAbe(G,\_,D):-estDansAbe(N,G).

estDansAbe(G,\_,D):-estDansAbe(N,D).

% somme(A,S) : vrai si S est la somme des nb contenus dans l'ABE, le

% prédicat somme doit permettre de calculer la somme des nb d'un Abe

% donné en entrée.

somme(S,abe(G,N,D)):-somme(S1,G),somme(S2,D),S is N+S1+S2.

**BILAN CH7**

%vérifier si X existe dans la liste

estDans(X,[X|\_Suite]).

estDans(X,[\_Y|Suite]):-estDans(X,Suite).

estDans2(X,[X|\_Suite]).

estDans2(X,[Y|Suite]):-X\==Y, estDans2(X,Suite).

estDans3(X,[X|\_Suite]):-!.

estDans3(X,[\_Y,Suite]):-estDans(X,Suite).

%l'ordre de clauses devient important pour le !

%version 1 retourner x = a

p(X):-q(X),!,r(X).

p(a).

q(a).

q(b).

r(a).

%version 2 retourner false

p2(X):-q2(X),!,r2(X).

p2(a).

q2(b).

q2(a).

r2(a).

%programer la négation

%np(X) vrai si p(X) faux

np(X):-p(X),!,fail.

np(\_).

%plus grand entre les deux valeur

plusGrand(X,Y,X):-X>=Y.

plusGrand(X,Y,Y):-X<Y.

%version cut piège, utiliser cut que c’est nécessaire

plusGrand2(X,Y,X):-X>=Y,!.

plusGrand2(X,Y,Y).