HAI815I TP PROLOG PRISE EN MAIN

PROLOG est le langage phare de l'IA, inventé avec l'IA dans les années 70. PROLOG revient à la mode alors que dans les dernières années, il était beaucoup moins utilisé sauf pour des domaines particuliers : traitement automatique des langues, systèmes experts. En revanche, le fragment DATALOG (bases de données déductives) a toujours été couramment utilisé et il l'est encore.

Nous utiliserons SWI PROLOG http://www.swi-prolog.org qui s'installe aisément sur tout système d'exploitation et qui peut aussi s'utiliser en ligne et en partageant ses programmes ici : https://swish.swi-prolog.org Une petite introduction à Prolog http://www.learnprolognow.org et vous avez aussi les transparents du cours.

ATTENTION : PROLOG fonctionne comme une base de données : on charge le programme et ensuite on l'interroge dans la fenêtre de terminal où on a lancé prolog (commande swipl sur les serveurs de la FDS), celle où se trouve le prompteur « ?- »

Pour vérifier que « ça marche » vous pouvez charger et interroger un petit programme:

Programme(7 lignes) 4 « faits » 3 « clauses »		8 requêtes pour la fenêtre « ?- » (une par une) « ; » pour avoir les autres réponses	
s(a,b). s(b,c). s(c,d). s(d,e).	mi(X,Y,Z) :-s(X,Y),s(Y,Z). inf(X,Y) :-s(X,Y). inf(X,Z) :-s(X,Y),inf(Y,Z).	s(c,d). s(d,c). s(X,c). mi(a,b,c).	mi(a,b,d). mi(b,X,d). inf(a,d). inf(Y,c).

Éditez votre programme dans un fichier monfichier.pl sous emacs. ESC X prolog-mode (ne pas utiliser le prolog inclus dans emacs, gnu prolog qui est moins bien et a une syntaxe assez différente).
Attention : les identifiants commençant par des Majuscules sont des Variables.
On écrit aime(pierre,X) : pierre est une constante (au sens logique), pas une chaine de caractère.
« _ » désigne une variable dont le nom n'a pas d'importance.

Une clause « inf(X,Z) :-s(X,Y),inf(Y,Z). » signifie « ∀X ∀Y ∀Z (s(X,Y)&inf(Y,Z)=>inf(X,Z) »
Les deux clauses définissant « inf » signifient (avec un renommage Y :=Z dans la première) :
« [∀X ∀Z s(X,Z)=> inf(X,Z)] & [∀X ∀Y ∀Z (s(X,Y)&inf(Y,Z)=>inf(X,Z)) »
ce qui est syntaxiquement équivalent à « ∀X ∀Y ∀Z [(s(X,Y)&inf(Y,Z)) v s(X,Z)]=> inf(X,Z) »,
avec le « ou » des antécédents. Les clauses sont évaluées dans l'ordre (ne pas séparer les clauses définissant un même prédicat), et les hypothèses sont essayées de gauche à droite.

Sous emacs, pour se mettre en mode prolog taper : « ESC X prolog-mode »
Pour lancer Prolog, taper dans un terminal « swipl ».

Répertoire courant : « pwd. » Pour changer de répertoire « cd("Repertoire/dossier"). »
Pour charger votre programme « consult(monfichier). » ou « consult("monfichier.pl"). »
Pour quitter « halt. »
Pour suivre l'exécution « trace. » pour en sortir « notrace. », puis « nodebug. »
Interruption brutale : « CTRL C » puis « a » pour abort.

On pourra utiliser (mais avec parcimonie) les opérateurs de contrôle qui utilisent l'ordre des clauses et l'ordre des prémisses dans une clause :
« fail » : échoue toujours, renvoie « false ».
« ! » : empêche de ré-essayer cette clause avec d'autres valeurs, empêche d'utiliser d'autres clauses.
« X=Y » : unification explicite, X et Y sont unifiables ;
« X\=Y » : X et Y ne sont pas unifiables ;
« X==Y » : X et Y sont instanciés et ont la même valeur ;
« X\==Y » : X et Y sont instanciés et n'ont pas la même valeur.

Exercice A familles

Homme	Femme	Parent	Parent (suite)
homme(albert). homme(jean). homme(paul). homme(bertrand). homme(louis). homme(benoit). homme(edgar).	femme(germaine). femme(christiane). femme(simone). femme(marie). femme(sophie). femme(madeleine).	parent(albert,jean). parent(jean,paul). parent(paul,bertrand). parent(paul,sophie). parent(jean,simone). parent(louis,benoit). parent(paul,edgar).	parent(germaine,jean). parent(christiane,simone). parent(christiane,paul). parent(simone,benoit). parent(marie,bertrand). parent(marie,sophie). parent(madeleine,edgar).

On pourra compléter ces données pour que les réponses soient plus intéressantes.
Pensez à tester vos programme avec des exemple.

(A.i) Traduire les questions suivantes et vérifier les réponses.
On rendra le texte des requêtes (mais pas celles en bleu).

a) Est-ce que paul est un homme ?
   homme(paul)
b) Est-ce que benoit est une femme ?
c) Quelles sont les femmes ?
d) Est-ce que marie est la mère de sophie ?
e) Qui est la mère de jean ?
   parent(X,jean),femme(X)
f) Quels sont les enfants de paul ?
g) Quels sont les hommes qui ont des enfants ?

(A.ii) Définir les prédicats suivants :

a) mere(X,Y) :X est la mère de Y ;
   mere(X,Y) :-parent(X,Y),femme(X).
b) pere(X,Y) : X est le père de Y ;
c) grand_pere(X,Y) : X est le grand-père de Y ;
   grand_pere(X,Y) :-pere(X,P),parent(P,Y).
d) grand_pere_maternel (X,Y) : X est le grand père maternel de Y ;
e) Demi-frere(X,Y) : X est le demi-frère de Y ; X est différent de Y et X et Y ont le même père ou la même mère --- les deux clauses correspondent au « ou » des clauses car (A=>C)&(B=> C) équivaut à (AvB) => demi_frere(X,Y) : X\==Y, homme(X),mere(Z,X),mere(Z,Y).
   demi_frere(X,Y) : X\==Y, homme(X),pere(Z,X),pere(Z,Y).
f) oncle(X,Y) X est le demi-frère de la mère ou du père de Y

(A.iii) Définir (récursivement si besoin) les prédicats suivants :

a) ancetre(X,Y) : X est un ancêtre de Y ;
   ancetre(X,Y) :-parent(X,Y).
   ancetre(X,Y) :-parent(X,I),ancetre(I,Y).
b) descendant(X,Y) : X est un descendant de Y ;

Les listes (non typées) en Prolog en deux mots (voir les transparents ou Learn Prolog Now):

[PremierElement | ListeDesElementsSuivants].
[a|[b,c]]=[a,b,c]
[X|Y]= [a|[b,c]] -> X=a, Y=[b,c]

Exercice B

On rendra quelques copies d'écran (il est aisé de copier à la souris depuis l'interpréteur vers un fichier word ou texte). Par exemple :

```
trace, (q(U,[7,3])).
Call:q(_4396, [7, 3])
Call:p(_4396, _4718)
Exit:p([1|_4718], _4718)
Call:p(_4718, [7, 3])
Exit:p([1, 7, 3], [7, 3])
Exit:q([1, 1, 7, 3], [7, 3])
U
[1, 1, 7, 3]
```

Sur un interpréteur « normal » on lance trace. dans l'interpréteur.
Sur https://swish.swi-prolog.org il faut taper « trace,requête » dans l'interpréteur et cliquer sur « step into » un certain nombre de fois, jusqu'à ce que le calcul se termine.

En mode trace les expressions `_5678` sont les noms en machine des variables.
« `?-trace.` » déclenche le mode trace. « ?- `notrace.` » conduit au mode debug. « ?- `nodebug.` » sort de debug et de trace.

Faire tourner le programme suivant :

- q(X,Z) :– p(X,Y),p(Y,Z).
  p([1|Z],Z).

- avec le but q(U ,[]) puis avec le but q(U,[1,2 ,3]).

  Observer en mode trace ce que Prolog fait.

Vous constatez que PROLOG suit la méthode de RÉSOLUTION.
C'est en ce sens que PROLOG est une illustration du cours de logique

Même question avec le programme :

```
lg([],0).
lg([_|L1],N1):-lg(L1,N), N1 is N+1.
```

- Et les buts lg([a,b,c],2). Puis lg([a,b,c],3). Puis lg([a,b,c,d],P).

Exercice C Listes en prolog

On rendra les clauses définissant les prédicats non traités.

On pensera à tester les programmes en les appelant et avec des listes et avec des variables. Pour définir une liste écrire liste1([a,2,b,3,4]). Les lettres a et b ne sont pas des variables ni des chaines de caractères mais des atomes (des constantes au sens logique.

(C.i) Définir le prédicat appartient(X,L) qui est vrai lorsque l'élément X appartient à la liste L.

```
appartient(X,[X|_]).
appartient(X,[_|L]) :-appartient(X,L).
```

(C.ii) Définir le prédicat non_appartient(X,L) qui est vrai lorsque l'élément X n'appartient pas à la liste L — on peut utiliser fail et !.

```
non_appartient(X,L) :-appartient(X,L), !,fail.
non_appartient(_,_).
```

ou :

```
notin(_,[]).
notin(X,[Y|L]):-X\==Y,notin(X,L).      avec X\==Y   : X et Y syntaxiquement différents
```

la dernière clause s'écrit aussi notin(X,[Y|L]):-X\=Y,notin(X,L). avec X\=Y : X et Y non unifiables

(C.iii) Définir le prédicat sans_repetition(L) qui est vrai lorsque la liste L ne contient pas deux fois le même élément.

```
sans_repetition([_]).
sans_repetition([X|L]) :-non_appartient(X,L),sans_repetition(L).
```

(C.iv) Définir le prédicat ajout_tete(X,L1,L2) qui est vrai lorsque L2 est la liste obtenue à partir de L1 par ajout en tête de l'élément X.

(C.v) Définir le prédicat ajout_queue(X,L1,L2) qui est vrai lorsque L2 est la liste obtenue à partir de L1 par ajout en queue de l'élément X.

(C.vi) Définir le prédicat supprimer(X,L1,L2) qui est vrai lorsque L2 est la liste obtenue à partir de L1 en supprimant la première occurrence de X s'il y en a une, et, lorsqu'il n'y en a pas lorsque L1=L2.

(C.vii) Définir le prédicat supprimer_fin(L1,L2) qui est vrai lorsque L2 est la liste obtenue à partir de L1 en supprimant son dernier élément, ou lorsque L1=L2=[].

(C.viii) Définir le prédicat fusion(L1,L2,L3) qui est vrai lorsque L3 est obtenue à partir de L1 et L2 en prenant alternativement un élément dans L1 et un dans L2 et en adjoignant en queue les éléments non encore utilisés de la liste la plus longue parmi L1 et L2.

(C.ix) Définir le prédicat concatener(L1,L2,L3) qui est vrai lorsque L3 est la liste dont les éléments sont d'abord ceux de L1 puis ceux de L2.

(C.x) Définir le prédicat inverser(L1,L2) qui est vrai lorsque L2 est constituée des même éléments que L1 mais en sens inverse.

(C.xi) Définir le prédicat commun(L1,L2,L3) qui est vrai lorsque L3 est la liste sans répétition des éléments communs à L1 et à L2. telle que l'ordre d'apparition des éléments dans L3 est l'ordre de leur première apparition dans L1 suivie de L2.

(C.xii) Définir le prédicat ens(L1,L2) qui est vrai lorsque L2 est obtenue à partir de L1 par suppression de toutes les occurrences d'un élément *sauf la dernière*. La liste L2 a les même éléments que L1, mais sans répétition.

(C.xiii) Définir le prédicat reunion(L1,L2,L3) qui, en supposant que L1 et L2 sont sans répétition, est vrai lorsque L3 est sans répétition et contient tous les éléments de L1 et de L2 dans l'ordre où ils apparaissent dans L1 suivie de L2.

(C.xiv) Définir le prédicat reunionbis(L1,L2,L3) qui est vrai lorsque L3 contient tous les éléments de L1 et de L2, au plus une fois, et dans l'ordre dans leur dernière apparition dans L1 suivie de L2. Ce prédicat ne requiert pas que L1 et L2 soient sans répétition.