TP PROLOG

Pour lancer Prolog :

GNU Prolog est un langage de programmation logique avec contraintes développé par Daniel Diaz à l'INRIA. Il est libre et vous pouvez le télécharger sur vos machines. GNU Prolog est installé sur les machines Linux de l'Enssat.

Créez un répertoire de travail dans lequel vous enregistrerez vos programmes.

mkdir Prolog
cd Prolog

Pour activer l'interpréteur, tapez :
gprolog

Vous êtes sous l'interpréteur ...

Pour définir une base de connaissances, ouvrez une fenêtre éditeur de texte dans laquelle vous écrirez votre code. Enregistrez votre fichier source avec l'extension .pl (ex : *famille.pl*).
pere(marcel, gwendoline) .

Pour compiler un fichier source en bytecode et le charger dans la base de connaissance de l'interpréteur, tapez (sous l'interpréteur) :

consult('famille.pl').

Pour lister toute la base de connaissance :

listing.

Pour lister un paquet de clauses :

listing(pere).

Pour interroger la base de connaissances :

pere(marcel, X). */* une variable commence par une majuscule */*

Gprolog affiche la première solution. Tapez *;* pour obtenir la solution suivante, *a* pour toutes les solutions, *rc* pour arrêter.

Lorsqu'on charge la base de connaissance décrite dans un fichier .pl, celle-ci s'ajoute à la base de connaissance de l'interpréteur. Cependant, les prédicats définis dans le fichier chargé écrasent les prédicats de même nom dans la base de connaissances.

Pour sortir de l'interpréteur :
halt. */* ou ctrl d*/*

Pour arrêter un programme qui boucle :
ctrl c

Problème n° 1 : Base de données familiales

A partir de faits du type :

```
pere(jean, jacques).
pere(jean, thierry).
pere(patrick, isabelle).
pere(jacques, michel).
pere(jacques, henri).

mere(magali, jacques).
mere(veronique, isabelle).
mere(veronique, lucie).
mere(isabelle, catherine).
```

écrire le prédicat
grandpere(X, Y)
qui est vrai si X est le grand pere de Y.

Expliquez l'exécution et indiquez le résultat de la question :

grandpere(jean, X).

Ecrire les prédicats demifrere(X, Y), soeur(X, Y), oncle(X, Y), tante(X, Y)...

Ecrire le prédicat
ancetre(X, Y)
qui est vrai si X est un ancêtre de Y.

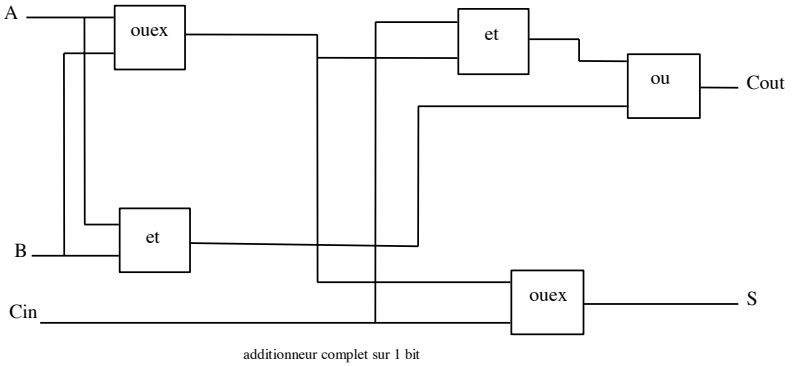
Problème n° 2 : **Unification**

Donnez le résultat de l'unification des listes suivantes :

[X, Y, Z] et [aa, bb, cc]
[X, Y | Z] et [aa, bb, cc]
[X, Y | [Z]] et [aa, bb, cc]
[X | [Y]] et [aa, bb, cc]
[X | [Y, Z]] et [aa, bb, cc]
[X | Y] et [[aa, bb], cc]
[X, Y, Z] et [[aa, ba], cc]
[X, Y | Z] et [[aa, bb], cc]
[X, bb, Z] et [aa, Y, c]
[X, Y, X] et [aa, bb, cc]
[[X, Y] | Z] et [[aa, bb] , [cc, dd], ee]

Problème n° 3 : **Simulation d'un circuit**

On veut simuler un additionneur pour des entiers codés sur 4 bits à partir du schéma suivant :



additionneur complet sur 1 bit

1) Simuler à partir de faits les portes *et*, *ou*, et *ouex*.

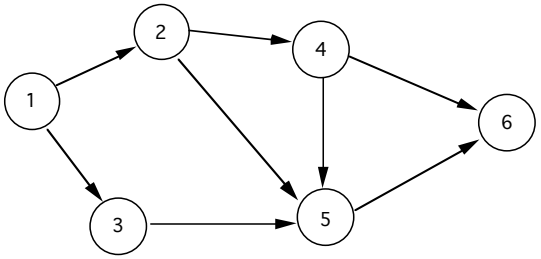
2) À partir des faits précédents, écrire le prédicat *addi(A, B, Cin, S, Cout)* qui simule l'additionneur sur un bit.

3) En déduire le prédicat *addi4([A1, A2, A3, A4], [B1, B2, B3, B4], [S0, S1, S2, S3, S4])* qui simule l'addition de deux entiers de 4 éléments binaires (les bits de poids forts sont à gauche et S0 représente le débordement).

Problème n° 4 : Parcours de graphe

Un graphe sera représenté par une base de faits contenant l'ensemble des transitions entre les sommets du graphe. Le prédicat *arc(X, Y)* est vrai s'il y a un arc du sommet X vers le sommet Y.

1) Modéliser le graphe suivant :



2) Écrire le prédicat *chemin(X, Y)* qui est vrai s'il existe un chemin menant du sommet X au sommet Y (on supposera que le graphe ne contient pas de cycle).

3) Tester votre programme en posant les questions : chemin(1, 5), chemin(1, Y), chemin(X, 6), chemin(X, Y).

Problème n° 5 : Les entiers

Les entiers naturels sont habituellement représentés par 1, 2, 3, ... La classe des entiers admet une construction récursive : un entier est l'entier 0 ou le successeur d'un entier. Soit ss le nom d'une structure à un argument, alors la suite : 0, ss(0), ss(ss(0)), ss(ss(ss(0))), ..., peut être une représentation des entiers. Cette représentation est en pratique inutilisable, nous l'utiliserons cependant pour définir des opérations sur les entiers, car dans le cadre de Prolog "pur" il n'y a pas d'autres solutions.

1) Écrire un prédicat entier(X) qui est vrai si l'objet x représente un entier. Tester votre programme en posant les questions : entier(ss(ss(ss(0)))), entier(X).

2) Écrire le prédicat plus(X, Y, Z) qui est vrai si X + Y = Z. Tester votre programme en posant les questions suivantes : plus(0, ss(0), Z). plus(X, ss(0), ss(ss(0))). plus(X, 0, ss(0)). plus(X, Y, ss(ss(ss(0)))).

3) Écrire le prédicat fois(X, Y, Z) qui est vrai si X * Y = Z.

Problème n° 6 : **Manipulation d'ensembles**

Dans cet exercice, on travaille sur des ensembles représentés par des listes. Une liste ne pourra dons pas contenir deux fois le même élément.
Écrire les prédicats suivants :

— *membre(X, Y)* qui est vrai si l'élément X appartient à l'ensemble Y.

— *inclus(X, Y)* qui est vrai si l'ensemble X est inclus dans l'ensemble Y.

— *enlever(X, Y, Z)* où Z est l'ensemble Y privé de l'élément X.

— *egal-ens(X, Y)* qui est vrai si X et Y sont des ensembles égaux.

— *intersection(X, Y, Z)* où Z est l'intersection de X et de Y.

— *union(X, Y, Z)* où Z est l'union de X et Y.

— *disjoint(X, Y)* qui est vrai si X et Y sont des ensembles disjoints.

— *soustraction(X, Y, Z)* où Z = Y - X (soustraction ensembliste).

On pourra utiliser le prédicat prédéfini not(P), qui est vrai si P n'est pas démontrable.

Problème n°7 : Tris

Exercice n° 1 : tri par insertion
Écrire le prédicat *insérer(X, L, S)* qui insère l'entier *X* dans la liste triée *L* pour produire la liste triée *S*. En déduire le prédicat *tri_ins(L, S)*.

Exercice n° 2 : tri par permutation
Écrire le prédicat *permut(L, S)* où *S* est une permutation de la liste d'entiers L. On songera pour cela à utiliser le prédicat *enlever* du précédent TP. En déduire le prédicat *tri-perm(L, S)* qui est vrai si S correspond à la liste L triée. Pour ce faire on génère les permutations de la liste *L* jusqu'à avoir trouvé la bonne permutation, celle correspondant à la liste triée *S*.

Exercice n° 3 : tri rapide
Écrire le prédicat *eclater(p, L, L1, L2)* qui décompose la liste d'entiers alphanumériques *L* par rapport à l'entier *p*, en 2 sous-listes *L1* et *L2*, la première contenant les éléments de *L* qui sont inférieurs à *p*, et la deuxième, les éléments supérieurs à *p*.
En déduire le prédicat *tri1(L, S)* où *S* est la liste triée des éléments de *L*. On écrira et utilisera à cette fin le prédicat de concaténation de listes.