TP 1 : prise en main avec prolog

Introduction

Prolog est un langage de programmation logique. Ceci veut dire qu'un programme ne se présentera pas sous forme d'une suite d'instruction (comme dans la programmation impérative), mais sous la forme d'un certain nombre d'"affirmations sur le monde", formant une base de connaissances. L'utilisation du programme se fera alors en interrogeant cette base de connaissances. Prolog dispos d'un puissant "moteur" lui permettant de déduire certains faits à partir de ceux que contient sa basede connaissances.

Exercice 1

Base de faits

1/Dans un fichier texte, entrez le code suivant

```
homme( jean) .
homme(alain).
femme( lucie) .
femme( nelly) .
femme(martine).
parent( jean, lucie) .
parent( nelly, lucie) .
parent( lucie, alain) .
parent(alain, martine).
```

Une telle base s'appelle base de faits, parce qu'elle regroupe des faits que l'on déclare être vrais.
Remarques :

- Pas de majuscule pour les noms (ce sont des symboles de constantes)
- Tous les faits se terminent par un point.

Sauvez le fichier sous le nom base1.pl.
Si l'extension .pl est associée au compilateur swi-prolog, vous pouvez simplement double-cliquer sur le fichier ci-dessus pour lancer Prolog. Sinon, trouvez le nom de l'exécutable (probablement pl) et tapez une commande du type swipl -s base1.pl
Prolog vous indique qu'il a chargé et compilé votre fichier et vous présente son invite de commande.

```
% base1.pl compiled 0.00 sec, 2,964 bytes
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.55)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.
For help, use ?- help(Topic). or ?- apropos(Word).
?-
```

Si vous avez fait une erreur de syntaxe, celle-ci sera signalée à ce moment-là.

Requêtes

Lorsque vous vous trouvez face à l'invite de Prolog, vous pouvez lui poser des questions. Dans la terminologie Prolog, on les appelle des requêtes.
Entrez :

```
? – femme( nelly) .
```

Si tout va bien, Prolog devrait vous répondre

```
true.
```

et revenir à l'invite de commandes.
Essayez maintenant la requête

```
? – femme( laura) .
```

Que répond Prolog ?
Vous constatez donc que Prolog considère que tout ce qu'il ne peut pas déduire de la base de faits est faux .
Pour quitter Prolog :

```
? – halt.
```

Variables

Il est possible d'utiliser des variables. Leurs identificateurs doivent commencer par une majuscule. Relancez Prolog en ouvrant votre fichier base1.pl.
À l'invite de commandes, tapez

```
? – homme( X) .
```

Prolog vous répond

```
X = jean
```

et ne revient pas immédiatement à l'invite. Tapez un espace, Prolog affiche X = alain. et revient cette fois-ci à l'invite de commandes.

Nous nous sommes pour l'instant limités à satisfaire un seul but à la fois, mais on peut aussi tenter de les combiner : si l'on entre plusieurs buts séparés par des virgules, Prolog essaiera de les satisfaire tous simultanément. Ainsi :

```
? – femme( X) ,parent( jean, X) .
```

Nous donne bien le X = lucie;
Que donne la requête suivante ?

```
? – femme( X) ,homme( X) .
```

Règles

Jusque-là, Prolog ressemble à une sorte de base de données, mais en moins pratique et plus lent. Pour en faire un langage de programmation, il manque un ingrédient essentiel : les règles.
Ouvrez à nouveau votre éditeur de texte et chargez le fichier base1.pl. À la suite des lignes existantes, rajoutez les deux suivantes :

```
ancetre( X,Y)  : – parent( X, Y).
%on lit : ancetre(X,Y) si parent(X,Y)

ancetre(X,Y) :-parent(X,Z), ancetre(Z,Y).
%on lit : ancetre(X,Y) s'il y a un Z tel que  parent(X,Z) et ancetre(Z,Y)
```

Une règle signifie que la partie gauche (avant le :-) est vraie si la partie droite est vérifiée. Elle doit aussi finir par un point.
Enregistrez votre fichier, relancez Prolog sur cette nouvelle base et essayez de l'interroger sur les ancêtres de Martine. Pour recharger ce nouveau programme, vous avez plusieurs solutions à choix :

1. Quitter Prolog et double-cliquer à nouveau sur base2.pl.
2. Utiliser la commande : consult('o:/path/to/base2.pl').
3. (Sous Windows) Dans le menu "file", choisir "consult" puis sélectionner votre fichier.
4. (Sous Windows) Dans le menu "file" toujours, choisir "reload modified files".

Testez votre nouvelle base :

- les descendants de alain ?
- les decendants de jean qui sont des hommes ?

Variable anonyme

Supposons maintenant que l'on veuille écrire un prédicat qui soit vrai si son argument est un père, sans s'intéresser à qui est l'enfant. On va naturellement essayer quelque chose comme

```
pere( X) : – pere( X, Y).
```

Ce qui veut dire quelque chose comme "X est père (tout court) si c'est le père de quelqu'un". . .
Si vous testez ce prédicat, Prolog affiche un Warning à la compilation qui n'est pas forcément une erreur, mais qui signifie : nous avons utilisé une variable Y mais dont la valeur n'apparait pas dans le résultat. Ce warning peut disparaître en utilisant une variable anonyme, qui est notée par un soulignement (_). Notre règle devient alors :

```
pere( X) : – pere( X, _) .
```

Tester cette règle.

Remarque : Chaque occurrence de la variable anonyme est indépendante.

Rajoutez à votre base des règles pour définir les prédicats suivants :
a) enfant(X,Y) : "X est un enfant de Y"
b) fils(X,Y) :"X est un fils de Y"
c) fille(X,Y) : :"X est un fille de Y"
e) mere(X,Y) :"X est une mère de Y"
f) grand_parent(X,Y) :"X est un grand-parent de Y"

Exercice 2

Ecrire le programme du calcul du carré d'un nombre entier comme suit :

```
lire(X) :- write('donner un entier '), nl, read(X), nl, write('votre entier est '),write(X),nl, nl.

calcul_carre(X,Y):-  Y is X * X.

ecrire_resultat(X,Y) :- write('le carré de '), write(X), write(' est '),write(Y), nl,nl.

aller :- lire(X), calcul_carre(X,Y), ecrire_resultat(X,Y).
```

1. Tester ce programme et ses prédicats un à un.
2. Réécrire ce programme en utilisant un seul prédicat sans variables « carre ».
3. Faire une boucle de calcul de carrés de nombres entiers tant que leurs valeurs soient différentes de zéro.
4. Enrichir ce programme en ajoutant un nouveau prédicat factorielle qui permet de calculer la factorielle d'un entier.