



Eclipse est un langage de programmation logique (Prolog) qui a été caché par les applications *clpq*, *fd* utilisées jusqu'alors. Prolog est un langage relationnel : une fonction  $Y=f(X_1, \dots, X_n)$  est remplacée par une relation  $rf(X_1, \dots, X_n, Y)$ . En particulier quand les valeurs  $X_1, \dots, X_n$  sont données la valeur  $Y$  correspondant est  $f(X_1, \dots, X_n)$ . Les relations sont définies par :

1. des faits  $A$ . (à lire  $A$  est vrai). Ex : `homme(martin)`.
2. Ex : des règles  $B$  :  $\neg A_1, \dots, A_m$ . (à lire  $B$  est vrai si  $A_1$  et ...et  $A_m$  sont vrais). Ex : `pere(X,Y) :-parent(X,Y),homme(X)`.

Les  $A$  ou  $B$  sont de la forme `nom(t1, ..., tn)` avec `nom` un nom de relation (commence toujours par une minuscule, exemple `pere`), des variables (commence toujours par une majuscule, exemple `Xa1`), des nombres (comme 1, 2, ...), des noms (commence toujours par une minuscule, exemple `anne`), des expressions (formées avec des opérations, des noms,...). L'ensemble est un programme logique et sert ensuite à répondre à des requêtes. Une requête est de la forme `:-A`. qui s'écrit simplement `A` quand on est sous l'interpréteur. Exemple `parent(X,martin)`.

### Exercice 1 (*Fonctionnement de l'interpréteur*)

Recopier puis charger le fichier `~lugiez/TPMI/fam1.pl` qui contient le programme :

```
homme(martin).
homme(pierre).
femme(anne).
femme(martine).
parent(anne,martin).
parent(jules,martine).
parent(martin,pierre).
parent(martine,pierre).
parent(martine,jean).
etudiant(pierre,1990,info,1).
etudiant(martine,1970,math,2).
```

Que répond l'interpréteur aux requêtes :

```
homme(martin).
homme(jean).
femme(X).
parent(X,pierre).
etudiant(X,Y,Z,T).
```

Recopier puis charger le fichier `~lugiez/TPMI/fam2.pl` qui contient le programme :

```
pere(X,Y):-parent(X,Y),homme(X).
mere(X,Y):-parent(X,Y),femme(X).
ancetre(X,Y):-parent(X,Y).
ancetre(X,Y):-parent(X,Z), ancetre(Z,Y).
```

Quelles sont les réponses de l'interpréteur aux requêtes :

```
mere(Martine,X).
mere(martine,X).
ancetre(X,pierre).
```

### Exercice 2 (*Listes*)

La liste vide se note [], la liste contenant les éléments  $e_1, \dots, e_n$  se note  $[e_1, \dots, e_n]$ . La notation  $[e|L]$  désigne la liste dont le premier élément est  $e$  et le reste de la liste est  $L$ . Certains predicats sont prédéfinis : `append(L1,L2,L3)` : vrai ssi  $L3$  est la concaténation de  $L1$  et  $L2$ . `member(e,L)` vrai ssi  $e$  est dans  $L$ .

Que doit répondre l'interpréteur aux requêtes suivantes (et vérifier votre réponse) ?

- `[a,b]=[XY]` .
- `[a]=[X|Y]` .
- `[a,b,c,d]=[X,Y|Z]` .
- `[a,b,c,d]=[X,Y,Z]` .
- `[[a,b,c],d]=[X|Y]` .
- `member(a,[b,c,a,d])` .
- `member(a,[b,c,d,e])` .
- `member(X,[a,b,c])` .
- `append([a],[b,c],[a,b,c])` .
- `append(L1,L2,[a,b,c])` .

Écrire un programme `efface(X,L,R)` qui efface toutes les occurrences de  $X$  dans la liste  $L$  et met le résultat dans la liste  $R$ .

### Exercice 3 (*Différence unification = et affectation is.*)

Que répond l'interpréteur à : `X=1+1.` et à `X is 1+1.` Explication = est l'unification pour laquelle les objets sont des arbres et les symboles sont non interprétés (ce sont des lettres). L'unification de  $s$  et  $t$  instancie les variables de  $s$  et  $t$  pour rendre identique les termes  $s$  et  $t$ . `is` est un prédicat système qui cherche à évaluer l'expression arithmétique qui est à droite pour l'affecter à la variable de gauche. Vérifier cette utilisation et les opérateurs connus sur des exemples :

```
fact(0,1). %cas de base
fact(N,F):- N>0, N1 is N-1, fact(N1,FP), F is N*FP. %appel recursif
%utilisation: fact(n,F) avec n un entier
```

```

long([],0).                                     %cas de base
long([X|L],N):-long(L,N1), N is N1+1.         %appel recursif

gcd(N,0,N).                                     %cas de base
gcd(I,J,G):-J>0, R is I mod J, gcd(J,R,G).    %appel recursif
                                         %utilisation: gcd(n,m,R) avec n>m, n,m entiers

```

#### Exercice 4

Que fait le programme :

```

enum(X,I,J):-I=<J, X is I.
enum(X,I,J):-I=<J, K is I+1,enum(X,K,J).

```

```

label([],I,J).
label([X|L],I,J):-enum(X,I,J), label(L,I,J).

```

(utiliser sur des buts `label([X1,...,Xn],i,j)` avec  $i,j$  des entiers donnés.

#### Exercice 5

Reprendre l'exercice du TP2 sur les maisons, allez voir la solution sur

`~/lugiez/TPMI/maisons.pl`

et la comprendre, puis l'exécuter.

#### Exercice 6

Utiliser la librairie *fd* pour résoudre l'exercice suivant. Un carré magique de dimension 3 est un carré découpé en 9 cases (par 3 lignes, 3 colonnes) chaque case contenant un nombre et tel que la somme des nombres de chaque colonne soit la même partout et la somme des nombres de chaque ligne soit la même partout. On demande qu'en plus un nombre ne figure qu'une seule fois dans le carré et que les nombres soient entre 1 et  $N$ . Ecrire le programme pour  $N = 9$ . Trouver le plus petit  $N$  pour lequel un carré de dimension 3 existe.

#### Exercice 7

Vérifier que le solveur du module `clpr` ne sait pas résoudre des problèmes de minimisation avec des contraintes de différence et d'inégalité strictes (inventez un tel problème dont vous connaissez la solution et testez-le).