TP1 – Prolog

Chloé Conrad, Nathalie Guin, Marie Lefevre & Maëva Somny

LANCER SWI-PROLOG

Avec l'éditeur de votre choix, créez un fichier avec une extension .pl dans lequel vous écrirez votre programme (par exemple tp1.pl). Attention à ne pas utiliser de majuscule pour la première lettre du nom du fichier.

Lancez Prolog :

- Sous Windows : lancez l'application Swi-prolog puis déplacez-vous dans le répertoire où figure votre fichier ;
- Sous Unix: placez-vous dans le répertoire où se trouve votre fichier puis lancez Prolog grâce à la commande prolog ou swipl ou pl.

Chargez votre programme grâce à l'instruction [tp1]. À chaque modification du programme, n'oubliez pas d'enregistrer et de recharger le fichier dans Swi-Prolog.

DEFINIR DES FAITS, DES REGLES ET LES EXPLOITER

Pour découvrir Swi-Prolog, nous allons reprendre l'exemple de la Généalogie vu en TD.

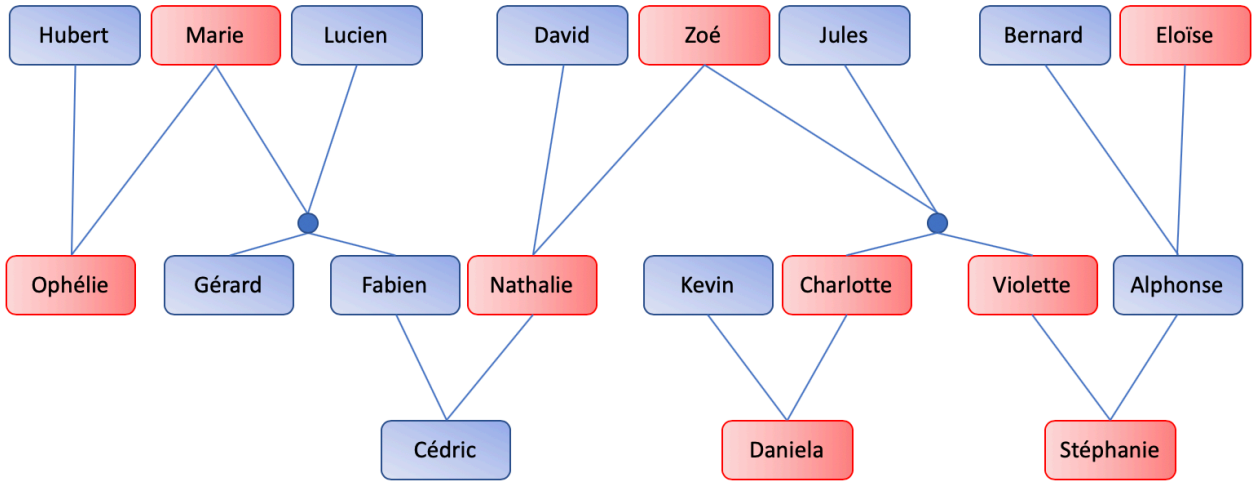
Sur la page de l'UE, récupérez le fichier « genealogie_base_faits.txt » contenant les faits suivants :

```
/* les hommes */
homme(alphonse).
homme(bernard).
homme(cedric).
homme(david).
homme(fabien).
homme(gerard).
homme(hubert).
homme(jules).
homme(kevin).
homme(lucien).

/* les femmes */
femme(charlotte).
femme(daniela).
femme(eloise).
femme(marie).
femme(nathalie).
femme(ophelie).
femme(stephanie).
femme(violette).
femme(zoe).
```

```
/* les relations de parenté */
/* où enfant(X,Y) signifie que X est enfant de Y */
enfant(alphonse,bernard).
enfant(alphonse,eloise).
enfant(cedric,fabien).
enfant(cedric,nathalie).
enfant(daniela,kevin).
enfant(daniela,charlotte).
enfant(fabien,lucien).
enfant(fabien,marie).
enfant(gerard,lucien).
enfant(gerard,marie).
enfant(nathalie,david).
enfant(nathalie,zoe).
enfant(ophelie,hubert).
enfant(ophelie,marie).
enfant(stephanie,alphonse).
enfant(stephanie,violette).
enfant(violette,zoe).
enfant(violette,jules).
enfant(charlotte,zoe).
enfant(charlotte,jules).
```

Ces faits représentent l'arbre généalogique ci-dessous :



Q1 : Posez des questions fermées afin d'obtenir une réponse oui/non. Par exemple « Cédric est-il un homme ? », « Cédric est-il un enfant de Charlotte ? ».

Q2 : Posez des questions ouvertes avec une seule variable. Par exemple « Qui sont les enfants de Zoé ? ».

Pour comprendre les réponses, utilisez le prédicat `trace`. Par exemple :

```
?- trace, enfant(X,zoe).
   Call: (9) enfant(_2524, zoe) ? creep
   Exit: (9) enfant(nathalie, zoe) ? creep
X = nathalie ;
   Redo: (9) enfant(_2524, zoe) ? creep
   Exit: (9) enfant(violette, zoe) ? creep
X = violette ;
   Redo: (9) enfant(_2524, zoe) ? creep
   Exit: (9) enfant(charlotte, zoe) ? creep
X = charlotte.

[trace]  ?- notrace.
true.

[debug]  ?- nodebug.
true. */
```

Q3 : Posez des questions ouvertes à plusieurs variables et tracez leur exécution. Par exemple « Qui est parent de qui ? », « Quels couples homme/femme ont eu un enfant ensemble ? ».

Q4 : Définissez les règles suivantes où `truc(X,Y)` signifie X est truc de Y, en n'oubliant pas d'interroger votre base de faits pour tester chacune de vos règles.

```
parent(X,Y) :- …
pere(X,Y) :- …
mere(X,Y) :- …
```

```
grand_parent(X,Y) :- …
grand_pere(X,Y) :- …
grand_mere(X,Y) :- …
```

Q5 : Définissez la règle récursive `ancetre(X,Y)` signifiant X est un ancêtre de Y, puis interrogez votre base de faits. Par exemple « Quels sont les ancêtres de Cédric ? ».

TRAVAILLER SUR LA RECURSIVITE

Q6 : Définissez un prédicat qui teste si un nombre est pair, sans utiliser la fonction *modulo*.

Q7 : Définissez un prédicat qui calcule la factorielle d'un nombre.

Q8 : Définissez un prédicat qui calcule la valeur u_n de la suite de Fibonacci sachant que

$u_0 = 1$, $u_1 = 1$, $u_n = u_{n-1} + u_{n-2}$

MANIPULER DES LISTES

Q9 : Définissez le prédicat `affiche(L)` qui affiche tous les éléments de la liste L.

Q10 : Définissez le prédicat `afficheInv(L)` qui affiche en ordre inverse tous les éléments de la liste L.

Q11 : Définissez le prédicat `premier(L)` qui affiche le premier élément de la liste L (et aucun autre).

Q12 : Définissez le prédicat `premier(L,X)` qui trouve le premier élément d'une liste.

Q13 : Définissez le prédicat `dernier(L)` qui affiche le dernier élément de la liste L (et aucun autre). Faites une version sans utiliser append, puis une en utilisant append.

Q14 : Définissez le prédicat `dernier(L,X)` qui trouve le dernier élément de L. Faites une version sans utiliser la fonction *append*, puis une en utilisant la fonction *append*.

Q15 : Définissez le prédicat `compte(L,N)` qui calcule le nombre d'éléments dans la liste L, sans utiliser la fonction *length*.

Q16 : Définissez le prédicat `somme(L,N)` qui calcule la somme des éléments de la liste d'entiers L.

Q17 : Définissez le prédicat `nieme(N,L,X)` qui trouve le N-ème élément de la liste L.

Q18 : Définissez le prédicat `occurrence(L,X,N)` qui calcule le nombre de fois où X est présent dans la liste L.

Q19 : Définissez le prédicat `substitue(X,Y,L1,L2)` qui est vrai si L2 est le résultat du remplacement de X par Y dans L1.