Expression Logique et Fonctionnelle . . . Évidemment

TP3 : Récursivité

Egalité et différence

`=:=` cet opérateur teste l'égalité *numérique* entre les termes gauche et droit ;
`=\=` cet opérateur teste l'inégalité *numérique* entre les termes gauche et droit ;
`\=` teste la non unification entre les termes gauche et droit.

1 Modes, unification et évaluation d'expressions

Modes des paramètres

SWI-PROLOG utilise les notations suivantes pour spécifier le *mode* des paramètres autorisé pour un prédicat donné :

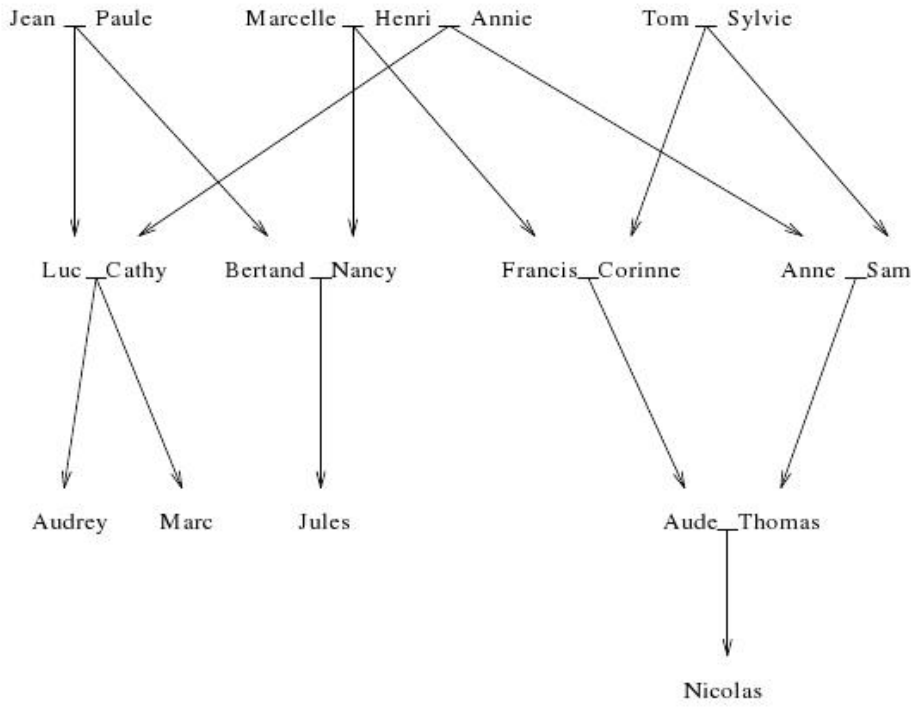
- un paramètre en "*entrée*", précédé du signe '+', est un paramètre clos (c'est-à-dire sans variable) ;
- un paramètre en "*sortie*", précédé du signe '−', est typiquement une variable ;
- un paramètre en "*entrée*" et/ou "*sortie*", précédé du caractère '?', est n'importe quel terme.

Question 1 Executez les buts suivants et comparez leur évaluation.

```
?- X = 1+1.
?- X is 1+1.
?- X = Y+1.
?- X is Y+1.
?- X+1 is 1+1.
?- X+1 =:= 1+1.
?- X+1 =\= 1+1.
?- 1+1+1 =:= 1+2.
```

Question 2 Déduisez de la question précédente le mode des paramètres des prédicats `=/2`, `is/2`, `=:=/2` et `=\=/2`.

2 Arbre généalogique



Le fichier `genealogie.pl` contient les faits permettant de définir cet arbre généalogique à partir des prédicats :

- `individu`(*?Nom, ?Sexe*) ;
- `pere`(*?Nom1, ?Nom2*), *Nom1* est père de *Nom2* ;
- `mere`(*?Nom1, ?Nom2*), *Nom1* est mère de *Nom2*.

Question 3 En utilisant la récursivité, écrivez les prédicats suivants :

- `ancetre`(*?Nom1, ?Nom2*) : *Nom1* est ancêtre de *Nom2* ;
- `descendant`(*?Nom1, ?Nom2*) : *Nom1* est descendant de *Nom2* ;
- `apparentes`(*?Nom1, ?Nom2*) : deux individus sont apparentés s'ils ont au moins un ancêtre commun.

Question 4 Testez ces buts en utilisant systématiquement la commande **`trace`** :

```
?- ancetre(marc,X).
?- descendant(X,marcelle).
?- apparentes(audes,thomas).
```

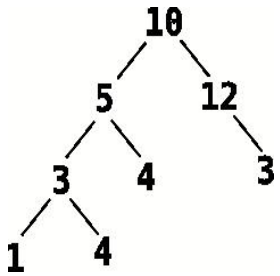
3 Récursivité et utilisation de foncteurs : les arbres binaires

Un arbre binaire est un arbre qui est soit l'arbre vide, soit un arbre ayant exactement deux sous arbres binaires qui sont eux aussi des arbre. La valeur attachée à un nœud de l'arbre s'appelle *etiquette* du nœud.

On représente :

- l'arbre binaire vide par la constante `vide` ;
- les arbres binaires non vides par des termes de la forme `noeud`(*Etiquette, Gauche, Droit*) où le terme *Etiquette* est l'étiquette de la racine, *Gauche* est le sous-arbre gauche (binaire), *Droit* est le sous-arbre droit (binaire).

L'AB[1] :



est donc représenté par :

```
noeud(10,
      noeud(5,
            noeud(3,
                  noeud(1,vide,vide),
                  noeud(4,vide,vide)),
            noeud(4,vide,vide)),
      noeud(12,
            vide,
            noeud(3,vide,vide)))
```

Le fichier `arbre.pl` contient un prédicat `exempleArbre` dont le paramètre définit des arbres comme celui ci-dessus. Vous pouvez vous en servir pour faire vos tests avec des buts de la forme :

```
?- exempleArbre(Arbre), predicat_a_tester(Arbre).
```

Par exemple, pour la question 8 :

```
?- exempleArbre(Arbre), estFeuille(Arbre).
```

Question 5 Quel but (i.e. quelle *requête*) permet de connaître la racine des différents arbres définis dans `arbre.pl` ?

Question 6 Écrivez le prédicat `racine`(*?Arbre, ?Val*) qui unifie *Val* avec l'étiquette de la racine de *Arbre*.

Question 7 Écrivez le prédicat `estVide`(*?Arbre*).

Question 8 Un arbre est une feuille s'il n'est constitué que d'un seul nœud, ses deux sous-arbres sont donc vides. Écrivez le prédicat `estFeuille`(*+Arbre*).

Question 9 Écrivez un prédicat `cherche`(*+Val,+Arbre*) qui est satisfait si *Val* est l'étiquette d'un nœud de l'arbre *Arbre*.

Question 10 Écrivez un prédicat `taille`(*+Arbre,-NbNoeuds*) qui unifie *NbNoeuds* avec la taille de l'arbre c'est-à-dire le nombre de nœuds (différents de l'arbre vide) de l'arbre *Arbre*.

Question 11 Écrivez un prédicat `hauteur`(*+Arbre,-Hauteur*) qui unifie *Hauteur* avec la hauteur de l'arbre. La hauteur correspond au nombre de nœuds de la plus longue branche (entre la racine et une feuille). L'arbre vide aura la hauteur 0 et une feuille la hauteur 1.

[1] sans l'affichage des arbres vides

Question 12 Écrivez un prédicat `nbOccurrences`(*+Val,+Arbre,-NbOcc*) qui unifie *NbOcc* avec le nombre d'occurrences de la valeur *Val* dans l'arbre *Arbre*.

Question 13 Écrivez le prédicat `somme`(*+A,-N*) qui calcule la somme *N* des valeurs des étiquettes de l'arbre *A*.

Question 14 Écrivez le prédicat `valeurMax`(*+A,-N*) qui calcule la plus grande des valeurs des étiquettes de l'arbre *A* et l'unifie avec *N*. Le terme `max`(*+A,+B*) est prédéfini (voir l'aide).