TP IA 2 : Prolog

Ph. Preux
Licence GMI
Université du Littoral Côte d'Opale
Calais

mars 2001

1 Une très courte introduction à `gprolog`

On utilise le système libre `gprolog` sous Linux. Pour l'utiliser, on construit tout d'abord une base de connaissances avec un éditeur de textes. Cette base de connaissances doit être mise dans un fichier d'extension `.pl` ; appelons-la `base.pl`. Ensuite, elle doit être compilée à l'aide de la commande `gplc base.pl -o base`. Cette commande est de la me forme que la commande `gcc` : elle génère un fichier exécutable dont le nom est indiqué par l'argument `-o` (ici, `base` donc). On peut ensuite taper cette commande ce qui a pour effet de lancer `gprolog` en chargeant la base de connaissances correspondante. On peut ensuite interroger la base de connaissances.

Pour quitter `gprolog`, il faut taper le caractère fin de fichier (Ctrl-D).

Suite à une interrogation, s'il y a plusieurs réponses, `gprolog` affiche la première puis attend une action de la part de l'utilisateur ; celui-ci peut soit frapper `;` pour obtenir la solution suivante, soit taper retour-chariot pour arrêter là. Dans le cas où le calcul de la réponse à une interrogation dure longtemps, on peut la stopper par `^C`, puis `a` au prompt qui suit.

Dans un fichier `.pl`, toute ligne qui débute par un `%` est un commentaire.

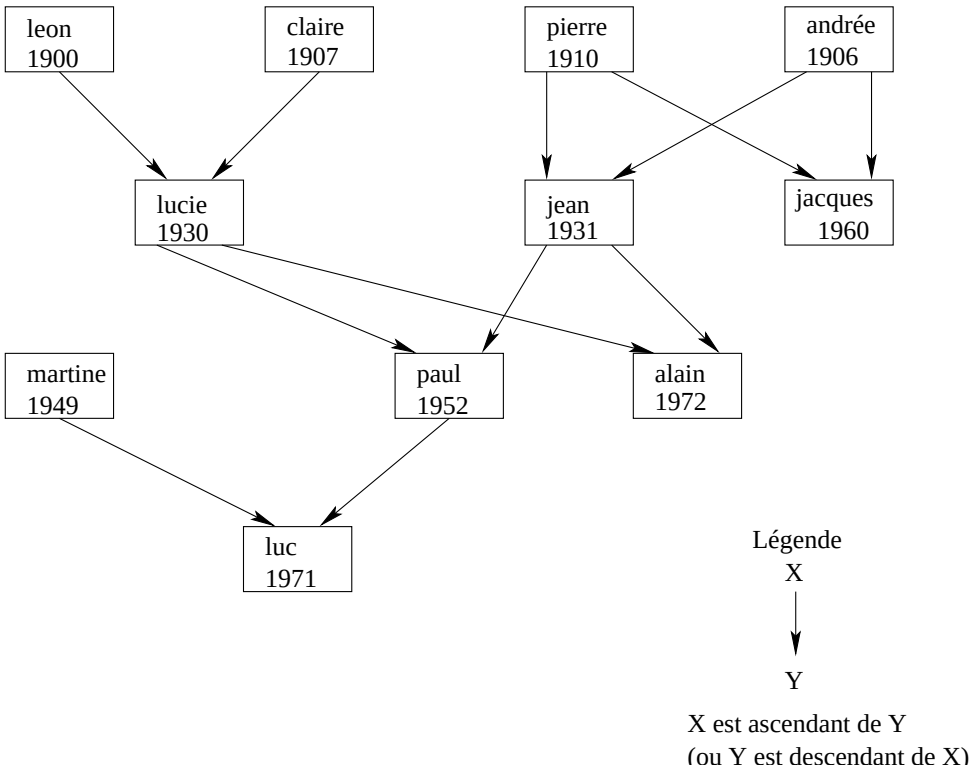
2 Histoires de famille

On reprend l'exemple vu en cours concernant les relations familiales.

Une base de connaissances minimale décrivant cette famille est disponible à l'url `http://www-lil.univ-littoral.fr/~preux/ensg/ia` (fichier famille).

Ajouter les prédicats suivants vus en TD :

- `estParent(X,Y)`
- `estEnfant(X,Y)`
- `estGrandPere(X,Y)`



- `estGrandMere(X,Y)`
- `estGrandParent(X,Y)`
- `estFrere(X,Y)`
- `estSoeur(X,Y)`
- `estPetitEnfant(X,Y)`
- `estPetitFils(X,Y)`
- `estPetiteFille(X,Y)`
- `estOncle(X,Y)`
- `estTante(X,Y)`
- `estAncetre(X,Y)`
- `estDescendant(X,Y)`
- `estCousin(X,Y)`
- `estCousine(X,Y)`
- `ontMemeGrandPere(X,Y)`

Concevoir les nouveaux prédicats suivants :

- `nombreEnfants(X,N)` qui est vrai si `X` a `N` enfants ;
- `nombreDescendants(X,N)` qui est vrai si `X` a `N` descendants ;
- `aEuUnEnfantAvantVingtAns(X)` qui est vrai si `X` a eu un enfant avant d'avoir 20 ans ;
- `onclePlusJeuneQueNeveu(X)` qui est vrai si `X` a un neveu ou une nièce plus jeune que lui ;

– `coupleDontLeMariEstLePlusJeune(X,Y)` qui est vrai si `X` est un homme marié avec une femme plus vieille que lui.

3 Traitement de listes

Écrire les prédicats suivants :

– `membre(X,L)` qui est vrai si la valeur `X` appartient à la liste `L`. Par exemple :

```
membre(3,[1,3,2]).
```

donnera `yes`,

```
membre(25,[1,3,2]).
```

donnera `no`,

```
membre(X,[1,3,2]).
```

donnera :

```
X = 1
X = 2
X = 3
```

Ensuite, tester l'interrogation suivante et interpréter :

```
membre(3,L).
```

– `longueur(L,N)` qui est vrai si la longueur de la liste `L` est `N`. Par exemple :

```
longueur([1,3,2],3).
```

donnera `yes`,

```
longueur([1,3,2],5).
```

donnera `no`,

```
longueur([[1,3],2],2).
```

donnera `yes`,

– `concat(L1,L2,L3)` qui est vrai si la liste `L3` résulte de la concaténation des listes `L1` et `L2`. Par exemple,

```
concat([1,2,3],[4,5],[1,2,3,4,5]).
```

donnera `yes`,

```
concat([1,2,3],[4,5],L).
```

donnera

```
L = [1,2,3,4,5]
```

Tester ensuite les interrogations suivantes :

```
concat(L,[4,5],[1,2,3,4,5]).
concat(L1,L2,[1,2,3,4,5]).
```

– `renverse(L,R)` qui est vrai si la liste `L` contient les mêmes éléments que la liste `R` dans le sens contraire. Par exemple :

```
renverse([1,3,2],[2,3,1]).
```

donnera `yes`,

```
renverse([1,3,2],L).
```

donnera

```
L = [2,3,1]
```

et

```
renverse([[1,2,3],4,5,[6,[7,8]]],L).
```

donnera

```
L = [[6,[7,8]],5,4,[1,2,3]]
```

- `prefixe(L,M)` qui est vrai si la liste `L` est le début de la liste `M` ;
- `sousListe(L,M)` qui est vrai si la liste `L` est une sous-liste de la liste `M` ;
- `nieme(L,N,X)` qui est vrai si le N^e élément de la liste `L` est `X` ;
- `minimum(L,X)` qui est vrai si la plus petite valeur de la liste `L` est `X` (`L` est forcément une liste d'entiers) ;
- `maximum(L,X)` qui est vrai si la plus grande valeur de la liste `L` est `X` (`L` est forcément une liste d'entiers) ;
- `somme(L,X)` qui est vrai si la somme des éléments de la liste `L` est `X` (`L` est forcément une liste d'entiers) ;
- `retireDoublon(L,M)` qui est vrai si les éléments de la liste `M` sont les mêmes que ceux de la liste `L` dont les doublons ont été retiré (`M` est un ensemble) ;
- `longueurBis(L,N)` qui est vrai si `N` est le nombre d'élements de `L` en comptant également ceux des sous-listes. Par exemple,

  ```
  longueurBis([[1,2,3],4,5,[6,[7,8]]],N).
  ```

  donnera

  ```
  N = 8
  ```

  alors que

  ```
  longueur([[1,2,3],4,5,[6,[7,8]]],N).
  ```

  donne

  ```
  N = 4
  ```

- `membreBis(X,L)` qui est vrai si `X` appartient à `L`, notamment à une sous-liste de `L`. Par exemple :

  ```
  membre(6,[[1,2,3],4,5,[6,[7,8]]]).
  ```

  donnera `no`, alors que

  ```
  membreBis(6,[[1,2,3],4,5,[6,[7,8]]]).
  ```

  donnera `yes`.
- `renverseBis(L,R)` qui est vrai si la liste `L` contient les mêmes éléments que la liste `R` dans le sens contraire et récursivement, les sous-listes de `L` et `R` sont inversées. Par exemple :

  ```
  renverseBis([1,3,2],L).
  ```

  donnera

  ```
  L = [2,3,1]
  ```

et

```
renverseBis([[1,2,3],4,5,[6,[7,8]]],L).
```

donnera

```
L = [[[8,7],6],5,4,[3,2,1]]
```

à comparer avec les exemples donnés pour `renverse` ;

- `miseAPlat(L,M)` qui est vrai si les éléments de la liste `M` sont les mêmes que ceux de la liste `L` dont les sous-listes one été « mises à plat ». Par exemple, `miseAPlat[2,4,[6,7,[8,1],3],2,[1,2]],[2,4,6,7,8,1,3,2,1,2]` est vrai ;
- `tri(L,M)` qui est vrai si la liste `M` est composée des éléments de la liste `L` triés par ordre croissant. Le tri est réalisé de la manière la plus simple qui soit : on cherche le plus petit élément que l'on met en premier élément de la iste résultat, on cherche l'élement le plus petit parmi ceux qui restent ce qui donnele deuxième élément de la liste résultat, ... ;
- `triRapide(L,M)` qui est vrai si la liste `M` est composée des éléments de la liste `L` triés par ordre croissant. Le tri est réalisé par l'algorithme de tri rapide.

4 À rendre

Le fichier `.pl` qui contient tous ces prédicats avec votre nom en commentaire en début de fichier.