INF 320 : Fiche de TD N°2

Étienne KOUOKAM

Année académique 2014-2015

N.B. Tous ces exercices peuvent être traités en TP, directement dans Prolog.

Exercice 1: Au restaurant

Considérez la base de données suivante qui représente la carte d'un restaurant.

```
hors-d-oeuvre(artichauts).
hors-d-oeuvre(crevettes).
hors-d-oeuvre(oeufs).
viande(grillade-de-boeuf).}
viande(poulet).
poisson(loup).
poisson(sole).
dessert(glace).
dessert(tarte).
dessert(fraises).
```

Cela constitue un programme Prolog à base de règles. Ces règles définissent des relations en introduisant à la fois les objets (crevettes par exemple) et leur classification (hors-d-oeuvre).

a. Comment nomme-t-on ce type de règles ?

b. Comment demander la liste des hors d'œuvre disponibles ?

c. Quel sera le résultat affiché ?

d. Définissez la relation plat qui dit : un plat est à base de viande ou de poisson.

e. Définissez la relation repas qui dit que un repas est constitué d'un hors d'œuvre, d'un plat et d'un dessert.

f. Comment demander la liste des repas possibles ?

g. Comment demander la liste des repas comprenant du poisson ?

h. Considérons maintenant la valeur caloriques des différents aliments. Prenons par exemple (valeurs fantaisistes) :

```
calories(artichauts, 150).
calories(crevettes, 250).
calories(oeufs, 200).
calories(grillade-de-boeuf, 430)
calories(poulet, 500).
calories(loup, 250).
calories(sole, 200).
calories(glace, 300).
calories(tarte, 400).
calories(fraises, 250) .
```

Demandez l'affichage de la valeur calorique des hors d'œuvre.

i. Définissez la valeur calorique d'un repas.

j. Comment demander cette valeur ?

k. Définissez la relation repas-equilibre (valeur calorique inférieure à 900 et demandez la liste des repas équilibrés à base de viande.

l. Traduisez les questions suivantes :

1. quels sont les repas comportant des crevettes ?
2. quels sont les repas ne comportant pas de fraises ?

m. Complétez le programme menu de sorte qu'un repas comporte une boisson à choisir parmi le vin, l'eau minérale ou la bière.

Exercice 2: Ecrire le programme prolog suivant dans un fichier fact.pl :

```
fact(0,1).
fact(N,R):-N>0,M is N-1,fact(M,T),R is N*T.
```

a. Que calcule le prédicat binaire fact ?

b. Dans le même répertoire, lancer une session prolog, et taper consult(fact).

c. Interroger le programme, en évaluant, par exemple, les requêtes qui suivent.

d. Essayer d'interpréter les résultats obtenus, pour chacune des requêtes suivantes :

```
fact(5,120).
fact(3,7).
fact(6,X).
fact(10,X).
fact(X,1).
fact(X,6).
fact(X,Y).
```

Exercice 3: Agence matrimoniale.

Nous sommes dans une agence matrimoniale qui possède un fichier de candidats au mariage organisé par les assertions suivantes :

```
homme(N, T, C, A).
femme(N, T, C, A).
```

où N est le nom d'un homme ou d'une femme, T sa taille (grande, moyenne ou petite), C la couleur de ses cheveux (noirs, blonds, bruns, roux, châtains), A son âge (jeune, mur ou vieux).

```
gout(N, M, L, S).
```

qui indique que la personne N aime le genre de musique M (classique, pop, jazz), le genre de littérature L (aventure, science-fiction, policier), et pratique le sport S (football, tennis, jogging).

```
recherche(N, T, C, A).
```

qui exprime que la personne N recherche un partenaire de taille T, ayant des cheveux de couleur C et dont l'âge est A. On considère que deux personnes X et Y de sexes différents sont assorties si X convient à Y et si Y convient à X. On dira que X convient à Y si d'une part X convient physiquement à Y (la taille, l'âge et la couleur des cheveux de X sont ceux que Y recherche) et si d'autre part les goûts de X et Y en matière de musique, littérature et sport sont identiques.

a. Donner un ensemble d'assertions représentant le fichier des candidats.

b. Ecrire les règles définissant *convient-physiquement(X, Y)*, puis les règles définissant *ont-memes-gouts(X, Y)*.

c. En déduire le programme qui détermine les couples assortis.

Exercice 4: Manipulations de listes

Les structures de listes sont supportées nativement dans le langage Prolog. Elles représentent un élément fondamental du langage. Elles sont notées entre crochets (ex. : [a, b, c]). La notation [a, b|L] désigne la liste L augmentée en tête des éléments a et b. Par exemple [a|[a, b]] est identique à [a, a, b].

a. Dans cette partie, il s'agit de définir un certain nombre de prédicats simples sur les listes. Certains sont déjà présents dans les bibliothèques standards du langage, d'autres non. (On pourra se servir des prédicats présents, notés entres crochets, pour vérifier la validité des prédicats définis). Définir les prédicats suivants :

1. prem(X, Y) : Y est la tête de la liste X.

2. rest(X, Y) : Y est la queue de la liste X.
3. der(X, Y) : Y est le dernier élément de la liste X [last].
4. nieme(N, X, Y) : Y est le N-ième élément de la liste X.
5. elem(Y, X) : Y est un élément de la liste X.
6. concat(X, Y, Z) : Z est la concaténation de X et Y [append].
7. long(X, N) : N est longueur de la liste X [length].
8. Écrire paire et impaire qui permettent de tester la parité de la longueur d'une liste.

b. Et maintenant :

1. Écrire le prédicat **extraire**(X, Y, E) : Y est la liste X dont on a extrait l'élément E.
2. Essayer les buts :

```
extraire([3], [], 3).
extraire(L, [1,2,3,2,1], 2).\\
extraire([1,2,3,4,5], [1,2,4,5], X).\\
extraire([1,2,3,4,5], L, X).\\
extraire([1,2,4,5], L, 3).
```

3. Commentez.
4. Utiliser **concat** pour écrire **sous_liste** :
   $sous\_liste(ListeIncluse, ListeContenant) : -\cdots$
5. Écrire **palindrome** qui permet de tester si une liste est un palindrome. On pourra penser à utiliser un accumulateur, c'est-à-dire une liste dans laquelle on stocke les éléments lus au moment de l'appel récursif.

c. En réalité, les listes en Prolog permettent de représenter beaucoup plus d'objets que ce qu'on entend par "liste" dans le langage courant. En effet, les listes en Prolog ne sont pas "typées", i.e. elles peuvent contenir des éléments hétérogènes. En particulier, il est possible de construire des listes qui contiennent d'autres listes.

Par exemple, la liste **[a, [a, b, c], [[d, e], f], g, [h, i]]** est totalement valide.

Une conséquence intéressante est qu'on peut représenter simplement des arbres (ex : **[tete, fils1, fils2, ... , filsn]**). Dans la suite, nous travaillerons uniquement sur des arbres binaires dont les nœuds ont soit deux fils (appelés respectivement fils gauche et fils droit), soit aucun fils (dans ce cas, le nœud est appelé feuille).

1. Écrire un prédicat qui vérifie qu'une liste est un arbre binaire.
2. Écrire un prédicat qui calcule la profondeur d'un arbre binaire.
3. Écrire un prédicat qui calcule le nombre de nœud d'un arbre binaire.