TP 1 - Prolog - S4

Exercice 1 (**Allons au restaurant**)

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

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

1. Créer le fichier menu.pl avec les faits ci-dessus et le charger dans Prolog à l'aide du prédicat consult/1 (l'argument est le nom du fichier, soit menu). On pourra utiliser listing/0 pour obtenir la liste des clauses.
2. Demander la liste des hors d'œuvre disponibles. Utiliser `;` après la première solution pour obtenir la solution suivante ou `a` pour les avoir toutes d'un seul coup.
3. Définissez la relation plat qui exprime qu'un plat est à base de viande ou de poisson.
4. Définissez la relation repas/3 qui précise qu'un repas est constitué d'un hors d'oeuvre, d'un plat et d'un dessert
5. Demander la liste des repas possibles.
6. Considérons maintenant les valeurs caloriques des différents aliments. Par exemple (valeurs fantaisistes) :

   ```
   calories(artichauts, 150).
   calories(crevettes, 250).
   calories(oeufs, 200).
   calories(grillade-de-boeuf, 500).
   calories(poulet, 430).
   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.
7. Définissez la valeur calorique d'un repas. Comment demander cette valeur ?
8. Définissez la relation repas-equilibre (valeur calorique inférieure à 900) et demandez la liste des repas équilibrés à base de viande.
9. Complétez le programme de façon qu'un repas comporte une boisson à choisir parmi le vin, l'eau minérale et la bière.

Exercice 2 (**Listes**)

Si x_1, ... , x_n sont des termes, $[x_1, ..., x_n]$ désigne la liste de ces termes, et si x est un terme et l une liste, $[x|l]$ désigne la liste qui a x comme premier élément (« head ») et l comme reste (« tail ») (par exemple $[1|[2|[]]] = [1, 2]$, [] étant la liste vide).

Donner le résultat de chacune des requêtes suivantes (travailler sur papier, puis lancer la requête pour vérifier).

1. `?- [a,[a]]=[H|T].`
2. `?- [[a,b],c]=[[H|T1]|T2].`
3. `?- [a,b,[c]]=[H1|[H2|[H3|T]]].`

Exercice 3 (**Quelques prédicats sur les listes**)

Pour chaque prédicat à définir, vous prendrez le soin de tester cette définition sur différents cas.

1. Définir le prédicat app(X,Y,Z), vrai si Z est la concatenation de X et Y (défini en cours).
2. Définir le prédicat dernier(X,L) qui réussit si X est le dernier élément de L. Donner deux versions de ce prédicat, avec et sans utilisation de append.
3. Écrire le prédicat mem/2 qui définit l'appartenance d'un élément (1er argument) à une liste (2ème argument).
4. Ecrire le prédicat double/2 défini par :
   double(L, S) est vrai si chaque élément de L apparaît deux fois consécutivement dans S.
   Exemple :

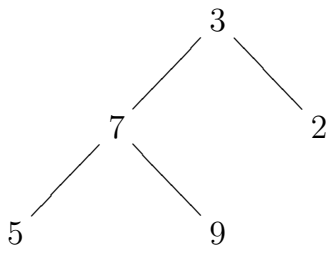
   ```
   ?- double([a,b,a], A).
   A=[a, a, b, b, a, a]

   ?- double([a,b,a], [a, a, b, a, a])
   no
   ```

5. Définir deux prédicats (mutuellement récursifs) longueurpaire/1 et longueurimpaire/1 qui réussissent si leur argument est une liste avec un nombre pair (impair) d'éléments (n'utilisez pas d'opérations arithmétiques. Ne calculez pas la longueur de la liste).
6. Définir le prédicat rev/2 pour inverser une liste. Par exemple rev([a,b,c],L) donne L = [c,b,a]. Est-ce que votre programme marche aussi avec rev(L,[a,b,c]) ?
7. Définir le prédicat prefixe(L1,L2) vrai si la liste L1 est un préfixe de la liste L2. Vous en écrirez deux versions, l'une utilisant app et l'autre non.
8. En utilisant les prédicats (de second ordre ou méta-prédicats), `setof/3` ou `bagof/3` (cherchez la documentation), écrivez une question qui permettent d'obtenir la liste de tous les préfixes de la liste [1, 2, 3].

   De la même façon, comment obtenir tous les couples de listes dont la concaténation donne la liste [1,2,3].
9. Écrire le prédicat nodouble/1 qui réussit si son argument, une liste, contient des doublons.

Exercice 4 (**Quelques prédicats arithmétiques**)

FIGURE 1 – Un exemple d'arbre binaire étiqueté



1. Écrire le prédicat fact/2 qui calcule la factorielle d'un entier. Calculer factorielle 4. Peut-on calculer N tel que sa factorielle soit égale à 2 ?
2. Écrire le prédicat division/4 défini par : division(A, B, Q, R) est satisfait si Q (resp. R) est le quotient (resp. reste) dans la division euclidienne de A par B.
3. Définir la fonction `sumlist(L, N)` satisfait si N est la somme des éléments de la liste d'entiers L.
4. Écrire un prédicat sorted qui réusiit si son argument est une liste d'entiers triée dans l'ordre croissant.
5. Écrire un prédicat ternaire merge(L, M, N) qui réussit si N est la liste triée contenant les éléments des listes triées L et M.

   Testez et vérifiez en utilisant le prédicat précédent qie si L et M sont triées alors N est triée.

Exercice 5 (**Arbres**)

On définit les arbres binaires étiquetées par des entiers à l'aide du constructeur de terme `arbre`. Ainsi l'arbre de la figure 1 est représenté par arbre(3,arbre(7,5,9),2).

Écrire un prédicat somme(Arbre,Somme), pour calculer la somme des étiquettes d'un arbre.

Exercice 6 (**Des prédicats Prolog sur des listes**)

1. Écrire un prédicat `insert(X, L, P)`. Il est satisfait si `P` désigne la liste d'entiers triée dans l'ordre croissant obtenue en insérant `X` dans la liste d'entiers `L` supposée triée dans l'ordre croissant.

   Exemple d'utilisation :

   ```
   ?- insert(7,[3,5,13],R).
   R = [3,5,7,13] ?
   yes
   ```

   Rappel : le terme `[T|L]` dénote une liste dont le premier élément est `T` et le reste est la liste `L`.
2. Écrire un prédicat `tri(L, R)` satisfait si `R` est la liste triée dans l'ordre croissant correspondant à la liste d'entiers `L`. On utilisera le prédicat précédent `insert` pour réaliser un tri par insertion.

   Exemple d'utilisation :

```
?- tri([5,3,13,1],R).
R = [1,3,5,13] ?
yes
```

3. Rappel : le terme `[P,D|R]` dénote une liste dont le premier élément est `P`, le deuxième est `D` et le reste est la liste `R`.

   Écrire un prédicat `trie(L)` qui réussit si l'argument est une liste triée dans l'ordre croissant, échoue sinon.

   Exemples d'utilisation :

```
?- trie([6,13,90]).
  yes
?- trie([4,2]).
  no
```

4. Si l = $[n_1, ..., n_k]$ est une liste d'entiers de longueur k, alors la liste des intervalles de l est la liste d'entiers $[n_2 - n_1, n_3 - n_2, ..., n_k - n_{k-1}]$, de longueur $k - 1$ (pour $k \leq 1$, la liste des intervalles est vide).

   Écrire un prédicat `intervalles(L, R)` satisfait si `R` est ls liste des intervalles de la liste d'entiers `L`.

   Exemples d'utilisation :

```
?- intervalles([7,9,5,47],R).
R = [2,-4,42] ?
yes

?- intervalles([7],R).
R = [] ?
yes
```