Expression Logique et Fonctionnelle . . . Évidemment

TP5 : récursivité terminale

1 Concaténation

Dans le TP précédent, vous avez utilisé le prédicat accoler/3 pour diviser une liste en deux listes consécutives (Q10) de la manière suivante :

```
?- accoler(X,Y,[a,b,c]).
```

On utilise des variables à la place des deux premiers arguments, que Prolog cherche alors à instancier. Comme l'exemple le montre, on peut trouver *toutes* les façons de couper une liste en deux par des retours en arrière enforcés, c.à.d. par la saisie du point-virgule. Cette façon d'utiliser accoler/3 rend la définition d'autres prédicats utiles très facile. Par exemple, on peut obtenir tous les préfixes d'un mot (représenté par la liste de ses caractères) de la manière suivante :

```
prefixe(P,L) :- accoler(P,_,L).
```

Question 1 Énumérez tous les préfixes de la liste [u,n,p,e,t,i,t,t,e,s,t].

Question 2 De la même manière, définissez un prédicat suffixe/2 qui trouve les *suffixes* d'une liste.

Question 3 Faites des traces de prefixe/2 et suffixe/2. Pourquoi prefixe/2 trouve-t-il les listes les plus courtes en premier, alors que suffixe/2 trouve d'abord les listes les plus longues ?

Question 4 Comment peut-on définir un prédicat qui trouve des *sous-listes* d'une liste ? Formulez un prédicat sousliste/2. L'idée est que sousliste(SubL,L) est satisfait s'il existe un suffixe S de L dont SubL est un préfixe.

Question 5 Observez le comportement de sousliste/2 en faisant des traces. Expliquez pourquoi ce prédicat génère *toutes* les sous-listes possibles, mais génère certaines sous-listes *plus d'une fois*.

2 Récursivité terminale et emballeurs

Il est important de savoir utiliser accoler, et de comprendre qu'il peut devenir une source d'inefficacité. Examinez l'exemple suivant, ou il s'agit d'inverser une liste à l'aide du prédicat inv/2.

```
inv([],[]).
inv([T|Q],R) :- inv(Q,InvQ), accoler(InvQ,[T],R).
```

Question 6 Expliquez, en français, le fonctionnement du prédicat inv/2.

Question 7 A l'aide d'une trace, déterminez le nombre d'étapes pour inverser une liste de longueur 10. Quel est le point sur lequel on gaspille du temps ?

Une meilleure stratégie de programmation consiste en l'utilisation d'un *accumulateur*, qui est une variable supplémentaire utilisée pour accumuler le résultat. Le deuxième paramètre de accInv/3 représente l'accumulateur. Observez son comportement :

```
accInv([T|Q],A,R):-accInv(Q,[T|A],R).
accInv([],A,A).
inverser(L,R):-accInv(L,[],R).
```

Le prédicat inverser/2 sert d'*emballeur* pour accInv/3. Il initialise l'accumulateur (le deuxième argument) avec une liste vide []. Typiquement, lorsqu'on écrit un predicat récursif terminal en Prolog, on l'accompagne d'un emballeur. Ceci facilite son utilisation par un programmeur, qui n'a plus à ce soucier de la gestion de l'accumulateur.

Question 8 Notez, sous forme d'un tableau, le contenu de la liste et de l'accumulateur lorsque vous tracez l'appel à inverser([t,e,s,t],R), étape par étape. Comparez le moment où le résultat devient disponible, à celui de l'appel à inv([t,e,s,t],R).

3 Evaluation d'expressions arithmétiques

Dans cet exercice, vous ferez connaissance du prédicat *is* de Prolog, qui force l'évaluation de l'expression arithmétiques à sa droite (supposant que toutes les variables de cette expression sont instanciés), puis associe le résultat de ce calcul à la variable à sa gauche.

Question 9 Testez (et comprenez !) les requêtes suivantes, puis *expliquez* ce que vous observez :

- ?- X = 3 + 2.
- ?- X is 3 + 1.
- ?- X is 5 + 4 * 3 - 2 + 1.
- ?- 3+2 is X.
- ?- Y is X + 1.
- ?- 3 is 2+1.
- ?- 4 is 2+1.
- ?- 2+1 is 2+1.

N'hésitez pas à consulter la doc en faisant appel à help(is).

Question 10 Formulez, en français, comment le prédicat long/2 suivant détermine la longueur d'une liste.

```
long([],0).
long([_|L],N):- long(L,V), N is V + 1.
```

Question 11 Analysez l'efficacité du prédicat long/2.

Dans le programme suivant, l'astuce est de stocker des résultats intermédiaires dans des variables. Ce principe est bien connu d'autres langages de programmation.

```
accLong([],Acc,Acc).
accLong([_|L],AccV,Res):-
    AccN is AccV + 1, accLong(L,AccN,Res).
longueur(Liste,Res):- accLong(Liste,0,Res).
```

Question 12 Expliquez l'utilisation de la variable AccN/AccV en deuxième position de accLong/3. Expliquez également son initialisation par l'emballeur longueur/2.

Question 13 Comparez l'efficacité du prédicat longueur/2 à celle de long/2.

Question 14 Formulez avec vos propres mots comment reconnaître si un prédicat est récursif terminal, à l'aide d'une trace.

4 Arithmétique avec vecteurs

Nous passons à un autre programme utilisant de l'arithmétique. Le prédicat ajouteUn/2 dont le premier argument est une liste d'entiers, et le deuxième une liste d'entiers obtenus en ajoutant 1 à chaque entier de la première liste.

```
ajouteUn([],[]).
ajouteUn( [H | T ],[ H1 | T1]):- ajouteUn(T,T1),
H1 is H + 1.
```

Question 15 Testez ce prédicat. Le résultat désiré est-il obtenu ?

Question 16 Inspectez une trace. Le programme est-il récursif terminal ? Le cas échéant, modifiez-le !

Le prédicat accMax/3 renvoit le maximum d'une liste d'entiers.

```
accMax([H|T],A,Max):- H > A, accMax(T,H,Max).
accMax([H|T],A,Max):-   H =< A , accMax(T,A,Max).
accMax([],A,A).
```

Question 17 Expliquez le fonctionnement de accMax/3.

Question 18 En changeant légèrement le code, transformez-le en un prédicat accMin/3 qui renvoit le minimum d'une liste d'entiers.

Question 19 Écrivez un emballeur min/2 pour accMin/3.

En mathématiques, un vecteur de dimension n est une liste de n valeurs. Par exemple [2,5,12] est un vecteur tridimensionnel et [45,27,3,-4,6] est un vecteur de dimension 5. L'une des opérations élémentaires sur les vecteurs et la multiplication scalaire. Dans cette opération, toutes les coordonnées du vecteur sont multipliées par un nombre. Ainsi, si on fait la multiplication par un scalaire du vecteur tridimensionnel [2,7,4] par 3 le résultat est un vecteur tridimensionnel [6,21,12].

Question 20 Écrivez un prédicat multiScal/3 dont le premier argument est un entier, le deuxième une liste d'entiers, et le troisième le résultat de la multiplication scalaire des deux premiers. Par exemple,

```
?- multiScal(3,[2,7,4],Resultat).
Resultat=[6,21,12]
```

Question 21 Après vous être assuré que votre prédicat est correct, inspectez une trace. Est-il récursif terminal ?

5 Parcours de graphes

Question 22 Faites l'exercice 4 du TD 5, le parcours de graphes à l'exemple du TER Nord-Pas-de-Calais.