Fiche 6 : Les listes en Prolog

Durée 40 min - Travail individuel [optionnel / en fonction de vos acquis]

Objectif : Savoir utiliser les listes dans le formalise de Prolog.

Définition et syntaxe des listes en Prolog

Les listes sont des structures de données très utilisées qui permettent les traitements récursifs.

Définition d'une liste :

- la liste vide, représentée par **[ ]**, est une liste,
- si T est un terme et L une liste, le terme **.(T,L)** représente la liste de premier élément **T** (ou « tête de liste »), et **L** est la liste privée du premier élément (ou « queue de liste »).
- **.** l'opérateur binaire de séquence
- une liste non vide est représentée par **[X | Ls]**

Notations :

- **.(T, L)**
- **[T | L],** avec l'opérateur | (« cons ») de construction de liste.

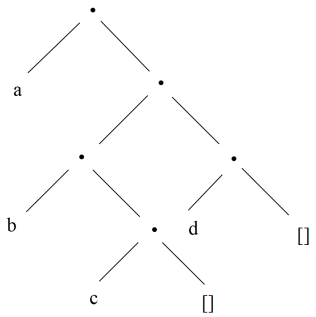
Pour simplifier l'écriture de listes imbriquées, on utilisera plutôt la notation **[T | L] .**

Exemples :

- **.(a, [ ])** est représentée aussi par [a], liste d'un élément
- **.(a, .(b, [ ]))** équivalent à **[a, b] ou [a | [b]]**
- **.(a, .(b, .(c, [ ]))** équivalent à **[a, b, c]** ou **[a | [b,c]]** ou **[a, b | [c]]**

Il est possible d'avoir plusieurs représentations de la même structure de liste. Les différentes représentations ci-dessous représentent le même arbre binaire :

- **.(a, .(.(b, .(c, [ ])), .(d, [ ])))**
- **[a, [b,c], d]**
- **[a | [[b,c], d]]**
- **[a, [b,c] | [d]]**



Unification sur les listes

L'unification sur une liste doit permettre de trouver une substitution d'une variable par un terme qui peut être lui-même une liste, une liste de listes, etc..

Exemples :

- **?- [[il, fait], beau,[a,paris]] = [X,Y].**
  - *no*
  - *une liste de 3 éléments ne peut s'unifier à une liste de 2 éléments*

- **?- [a, [b, c], d] = [X, Y, Z] .**
  - *X = a; Y = [b,c]; Z = d ;*
  - *no*

- **?- [a, b, c, d] = [a, b | L] .**
  - *L = [c,d] ;*
  - *no*

- **?- [a, [b, c], d]=[a, b | L].**
  - *no*

Exercice 1 - Listes et récursivité: recherche d'un élément

Les listes sont très utilisées pour faire de la récursivité sur un ensemble de termes à manipuler.

Soit le prédicat récursif à base de liste permettant de vérifier l'appartenance d'un terme à une liste :

```
/* condition d'arrêt de votre récursivité :
   l'élément recherché a été trouvé, quelque soit la fin de liste */
element(X,[X|_]).

/* condition générale de la récursivité : on parcourt notre liste en
   « éliminant » à chaque tour le premier élément de la liste */
element(X,[_|Ls]):-element(X,Ls).
```

- Que se passe-t-il si vous testez votre prédicat sur une liste à 3 éléments en recherchant un élément appartenant à la liste ?
- Que se passe-t-il si vous testez votre prédicat sur une liste à 3 éléments en recherchant un élément n'appartenant pas à la liste ?
- Que se passe-t-il si vous testez votre prédicat sur une liste vide ?

- Tracez les arbres de résolution pour chacun de ces cas.

Exercice 2 - Listes et récursivité: concaténation de listes

Ecrivez un prédicat permettant de concaténer deux listes ensemble :

- ✗ Combien d'arguments doit avoir votre prédicat ?
- ✗ Quelle est la condition d'arrêt ?
- ✗ Quelle est la condition générale de la récursivité ?
- ✗ Testez votre prédicat dans différents cas (listes vides, tous les arguments sont connus, seul une partie des arguments sont connus, taille de listes différentes, mauvaise concaténation, etc.) ?

Exercice 3 - S'entraîner à manipuler des listes

Ecrivez les prédicats suivants (chaque prédicat est indépendant) :

- ✗ Prédicat permettant de connaître la longueur d'une liste;
- ✗ Prédicat permettant de calculer la somme des éléments d'une liste;
- ✗ Prédicat permettant de supprimer un élément d'une liste;