4. La récursivité sous différentes facettes *(VIDEO-PPT 2.4)*

Dans ce chapitre quelques principes de bases de la programmation en Prolog sont analysés. Les techniques mises en œuvre s'appuient sur les fondements du langage, l'unification, la réversibilité et la récursivité.

4.1. La récursion dans les données

- **Listes**

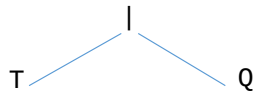
Rappel : les listes ont déjà été introduites dans leur formulation en extension

- *Exemples* :
  - `[aa,2,'abc']` est une liste (hétérogène, Prolog n'est pas typé) à 3 atomes
  - `[1, [aa,2,'abc'],'de']` est une liste à 3 éléments dont le 2eme élément est lui-même une liste

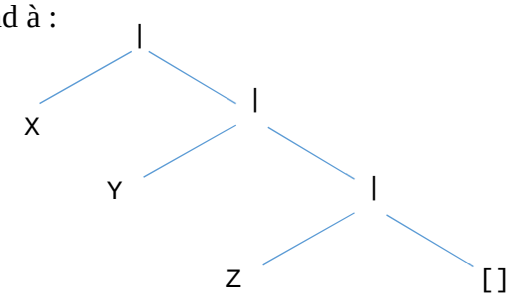
Passons à leur définition en compréhension par récursion :

Une liste `L` en Prolog s'écrit comme :

- la liste vide `[]`
- une liste ayant au moins un élément dans un arbre binaire `[T|Q]`, `T`(ete) étant l'élément de tête et `Q` étant la liste initiale `L` privée de son 1er élément (c'est donc aussi une liste).



Une telle liste est un terme composé binaire dont le foncteur est la barre verticale | et qui se développera dans sa définition récursive sur le fils droit `Q`(ueue). Ainsi `[X |[Y |[Z | []]]]` correspond à :



Les listes sont des arbres binaires « dégénérés » au sens où ils se développent sur leur fils droit. Les listes sont parfois qualifiées de « peignes ».

Raccourcis syntaxiques

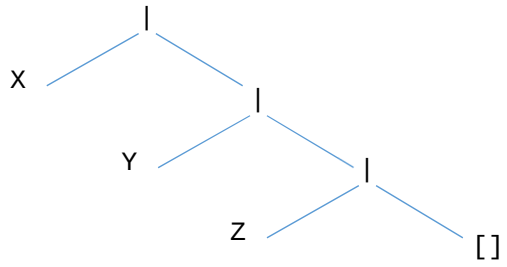
La liste a un élément `[X]` est un raccourci pour `[X|[]]`.

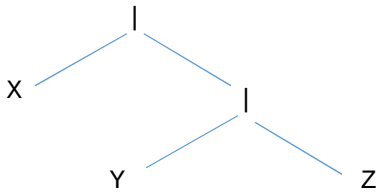
L'opérateur « virgule » déjà rencontré est en fait un raccourci syntaxique « d'emboîtement » :

- `[X,Y]` pour `[X|[Y]]` : liste à 2 éléments
- `[X,Y,Z]` pour `[X|[Y|[Z]]]` : liste à 3 éléments
- `[X, Y | Z]` pour `[X|[Y | Z]]` : liste (pour peu que `[]` la termine) qui contient au moins 2 éléments

☞ *Attention* : `[X, Y, Z]` est différent de `[X, Y | Z]`.

La première liste représente une liste à 3 éléments, la seconde représente une liste (pour peu que `[]` la termine) qui contient au moins 2 éléments (`Z` correspondant à la liste moins ses deux premiers éléments). Comparez :





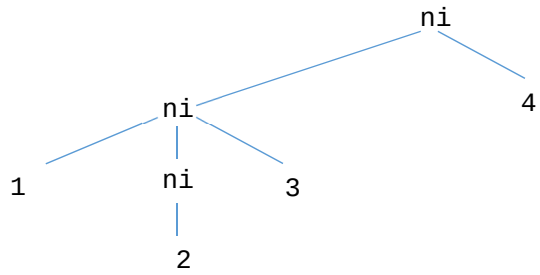
➢ *Exemples* : en supposant l'existence d'un prédicat `eq(X,Y)` qui réussit si `X` est unifiable à `Y` :

- `eq( [1, 2, 3], [X|Y])` fournira `{X=1, Y=[2,3]}`
- `eq( [1, 2, 3], [X,Y])` échouera (impossible d'unifier des listes à 3 et 2 éléments)
- `eq( [1, 2, 3], [X, Y | Z])` fournira `{X=1, Y=2, Z=[3]}`
- `eq([1, 2, 3], [1, 2 | [3]] )` réussira

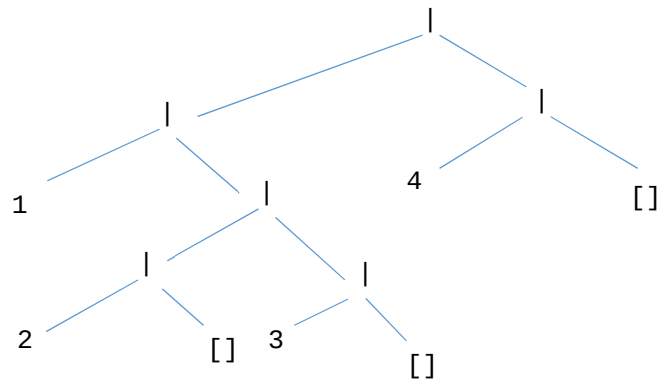
- **Arbres binaires d'énumération**

La structuration de données peut être nécessaire dans plus d'une dimension d'énumération, il suffit alors de « *mettre des listes dans des listes* » :

➢ *Exemples* : le terme composé `ni(ni(1,ni(2),3),4)` (de foncteur `ni` pour « nœud interne ») représente l'arbre « intuitif » suivant d'arité variable ne portant des valeurs que sur ses feuilles



Pour une gestion avec des listes, il se codera en Prolog `[[1,[2],3],4],` soit graphiquement :



- *miniTP :*
  a) *En vous aidant si besoin d'un schéma, donnez le résultat de l'unification des listes suivantes :*
     1. `[X, Y, Z]` et `[a, b, c]`
     2. `[X, Y | Z]` et `[a, b, c]`
     3. `[X, Y | [ Z]]` et `[a, b, c]`
     4. `[X | [ Y]]` et `[a, b, c]`
     5. `[X | [ Y, Z]]` et `[a, b, c]`
     6. `[X | Y ]` et `[[a, b], c]`
     7. `[X, Y, Z]` et `[[a, b], c]`
     8. `[X, Y | Z]` et `[[ a, b], c]`
     9. `[X, b, Z]` et `[a, Y, c]`
     10. `[X, Y, X]` et `[a, b, c]`
     11. `[[X, Y] | Z]` et `[[ a, b], [c, d], e]`

  b) *Écrire le prédicat eq avec deux arguments qui est vrai si le premier et le deuxième argument s'unifient. Vérifiez vos unifications de listes à l'aide de ce prédicat.*