[IA02] TD feuille 7 – Listes en Prolog

Exercice sur les listes ☆ (exercice clé)

Prédicats de base

Définir les prédicats Prolog suivants :

1. `tete(+L, -H)` qui unifie la variable `H` avec la tête de la liste `L`.
2. `reste(+L, -R)` qui unifie la variable `R` avec le reste de la liste `L`.
3. `vide(+L)` qui est vrai si la liste `L` est vide.
4. `element(?X, ?L)` qui est vrai si `X` est présent dans la liste `L`.
5. `dernier(+L, -X)` qui unifie `X` avec le dernier élément de la liste `L`.
6. `longueur(+L, -Lg)` qui unifie `Lg` avec la longueur de la liste `L`.
7. `nombre(+L, +X, ?N)` qui compte le nombre de fois où `X` apparaît dans `L` et unifie le résultat avec `N`.
8. `concat(+L1, +L2, -L3)` qui effectue la concaténation de la liste `L1` avec la liste `L2` et l'unifie avec `L3`.
9. `inverse(+L, -R)` telle que la liste `R` soit l'inverse `L`.
10. `sous_liste(+L1, +L2)` qui vérifie que `L1` est une sous liste de `L2`.
11. `retire_element(+L, +X, -R)` qui retire la première occurrence de l'élément `X` dans `L` et place le résultat dans `R`.

Tri en Prolog ☆☆☆

On veut implémenter un tri sous Prolog. On utilisera un algorithme de tri de type *quicksort*. On suppose que `L` est une liste d'entiers à trier. Soit `X` un élément de `L`. On considère `L1 = { Y ∈ L\X tel que Y ≤ X }` et `L2={ Y ∈ L\X tel que Y > X }`. Alors la liste `L` triée est égale à : `[ liste L1 triée..., X , liste L2 triée...]`.

1. Définir `partition(+X, +L, -L1, -L2)` qui place dans L1 les éléments de `L` qui sont inférieurs ou égaux à `X`, et dans `L2` les éléments de `L` qui sont strictement supérieurs à `X`.
2. Définir `tri(+L1, ?L2)` qui trie la liste `L1` et unifie le résultat avec `L2`.

Les ensembles ☆

On souhaite représenter les ensembles en Prolog comme des listes sans doublons. Définir l'ensemble des prédicats suivants.

1. `retire_elements(+X, +L, -R)` qui retire toutes les occurrences de `X` dans `L` et place le résultat dans `R`.
2. `retire_doublons(+L, -E)` qui transforme la liste `L` en un ensemble `E` (sans redondance).
3. `union(+E1, +E2, -E)` qui effectue l'union de l'ensemble `E1` avec l'ensemble `E2` et place le résultat dans `E`.
4. `intersection(+E1, +E2, -E)` qui effectue l'intersection de l'ensemble `E1` avec l'ensemble `E2` et place le résultat dans `E`.