UPMC
Année 2017-2018

LRC - TME n°6

M1 ANDROIDE
M1 DAC

1 Quelques éléments de prolog

Listes Une liste est une suite finie d'objets, de taille quelconque. Elle est représentée entre crochets, les éléments étant séparés par des virgules : `[1, 2, 3, 4]` pour une liste d'entiers, `[a, b, c]` pour une liste de caractères, `[]` pour la liste vide. Cette structure peut être définie récursivement par

- la liste vide, représentée par `[]`.
- si `T` est un terme et `L` une liste, `[T | L]` représente la liste de premier élément `T` (tête de liste), suivi de `L` (queue de liste). L'opérateur `|`, appelé **cons** permet de construire les listes.

Ainsi, les symboles `[]` et `|` peuvent être vus comme des symboles de fonction permettant de construire des termes.

Remarque : un élément d'une liste peut être un terme quelconque, par exemple une liste (voir les exemples ci-dessous).

Unification L'opérateur `=` a un sens spécial en prolog : il ne correspond pas à une instruction d'affectation comme c'est le cas classiquement, mais à un *test d'unification*. Le test est satisfait s'il existe une instanciation des variables présentes qui rend égaux les deux expressions, il échoue sinon.

Par exemple, la requête `toto(X, a) = toto(b,c)` échoue, car aucune valeur de `X` ne permet de rendre les deux expressions égales. Par contre, `toto(X,c) = toto(b,c)` réussit et rend `X=b`.

2 Exercices

Exercice 1 - Listes et unification

Calculer sur papier le résultat que prolog renvoie pour les requêtes suivantes, selon les principes de construction de liste et d'unification résumés ci-dessus, puis vérifier, en utilisant prolog, que vos réponses sont exactes.

1. `?- [a, [b, c], d] = [X].`
2. `?- [a, [b, c], d] = [X, Y, Z].`
3. `?- [a, [b, c], d] = [a | L].`
4. `?- [a, [b, c], d] = [X, Y].`
5. `?- [a, [b, c], d] = [X | Y].`
6. `?- [a, [b, c], d] = [a, b | L].`
7. `?- [a, b, [c, d]] = [a, b | L].`
8. `?- [a, b, c, d | L1] = [a, b | L2].`

Exercice 2 - Prédicats de manipulation classique

Il existe en prolog les prédicats `append/3`, `delete/3`, et `reverse/3`. Il est demandé ici de les reprogrammer (et donc de ne pas les utiliser), en veillant à considérer des récursions terminales.

Dans chaque cas, tester les prédicats en faisant varier les statuts entrée/sortie des arguments.

1. Ecrire le prédicat `concatene`/3 tel que `concatene(X,Y,Z)` est satisfait si `Z` est la concaténation des deux listes `X` et `Y`. Ainsi, `concatene([a,b,c],[d],L2).` doit conduire à l'unification `L2 = [a,b,c,d]`. Remarquer que ce prédicat permet de décomposer une liste donnée en sous-listes quand on fait varier le statut entrée/sortie des paramètres.
2. Ecrire le prédicat `inverse`/2 tel que `inverse(L1,L2)` est satisfait si la liste `L2` est l'inverse de la liste `L1`. Ainsi, `inverse([a,b,c,d],L2).` doit conduire à l'unification `L2 = [d, c, b, a]`.
3. Ecrire le prédicat `supprime`/3 tel que `supprime(X,Y,Z)` est satisfait si `Z` est la liste `X` de laquelle les occurrences de `Y` ont été supprimées. Ainsi `supprime([a,b,a,c], a, L).` doit unifier `L` avec la liste `[b,c]`.
4. Ecrire le prédicat `filtre`/2 tel que `filtre(L1,L2,L3)` est satisfait si `L3` est la liste `L1` obtenue après suppression de tous les éléments qui apparaissent dans la liste `L3`.

   Ainsi `filtre([1,2,3,4,2,3,4,2,4,1],[2,4], L)` doit unifier `L` avec la liste `[1, 3, 3, 1]`.

Exercice 3 – Palindromes

Un palindrome est une mot (ou une portion de phrase) qui se lit de la même façon de gauche à droite et de droite à gauche, comme par exemple “non”, “Laval”, “Esope reste ici et se repose” (on ne compte pas les espaces, ni les majuscules ni les accents). On souhaite écrire ici des prédicats permettant de tester si des listes de lettres correspondent à des palindromes ou non.

1. Définir un prédicat `palindrome/1` qui est satisfait si un mot, représenté par la liste de ses lettres (minuscules, sans espaces ni accents), est un palindrome. Ainsi `palindrome([l,a,v,a,l])` doit réussir, `palindrome([n,a,v,a,l])` doit échouer.
2. Définir un prédicat `palindrome2/1` identique à `palindrome` mais qui n'utilise pas la fonction `inverse` : il n'est en fait pas nécessaire de parcourir toute la liste pour l'inverser, on peut s'arrêter dès qu'on rencontre un problème.