Travaux pratiques de programmation logique

Philippe Morignot, sur une base de François Fages et al. – Décembre 2022. Revu en mai 2023.

Installer SWI-Prolog sur sa machine depuis l'adresse https://www.swi-prolog.org/

Avoir le manuel de référence (Documentation > Manual) dans un butineur web.

Manipulation de listes

Exercice 1 (membre) : écrire le prédicat **membre(E, L)** qui est vrai si l'élément **E** appartient à la liste **L**.

Vérifier qu'on peut énumérer les éléments d'une liste avec ce prédicat.

Votre écriture de ce prédicat doit probablement être incomplète parce que si un élément est membre plusieurs fois d'une liste, il y aura plusieurs réponses :

| ?- membre(2, [1,2,3,2,4,2,5]).

true ? ;

true ? ;

true ? ;

(16 ms) no

Utiliser l'opérateur **cut « ! »** pour qu'un élément n'appartienne qu'une seule fois à une liste même avec des doublons.

| ?- membre(2, [1,2,3,2,4,2,5]).

yes

Réaliser la même chose, mais sans l'opérateur **cut** et avec l'opérateur de non-unification.

Exercice 2 (concaténation) : écrire le prédicat **concatene(L1, L2, L)** qui est vrai si la liste **L** est la concaténation des listes **L1** et **L2**.

| ?- concatene([1,2], [3,4,5], X).

X = [1,2,3,4,5]

Yes

Vérifier qu'on peut trouver toutes les décompositions en deux sous-listes d'une liste avec ce prédicat.

Exercice 3 (enlèvement) : écrire le prédicat **enleve(E, L1, L2)** qui est vrai si la liste **L2** est la liste **L1** privée de l'élément **E**.

| ?- enleve(2, [1,2,3,2,4,2,5], X).

X = [1,3,4,5] ?

Yes

Exercice 4 (renverse) : écrire en utilisant la concaténation le prédicat **renverse(L1, L2)** qui est vrai si la liste **L2** est l'image renversée de la liste **L1**.

| ?- renverse([1,2,3,4,5], L).

L = [5,4,3,2,1]

Yes

Quelle est la complexité algorithmique de ce prédicat ? En déduire le prédicat **renverse_lin** qui est la version de complexité linéaire du prédicat **renverse**.

Ecrire le prédicat **palindrome(L)** qui est vrai si la liste **L** est identique à son image renversée (par exemple, « radar » ou la ville « Senones »).

Exercice 5 (maximum) : utiliser l'opérateur **not** « **\+** » pour écrire le prédicat **maximum(L, M)** qui est vrai si l'élément **M** appartenant à la liste **L** est le nombre maximum dans cette liste. C'est-à-dire si l'élément **M** appartient à la liste **L** et s'il n'existe pas d'élément **X** appartenant à la liste **L** tel que **X > M**. En termes algébriques : $M = \max(L) \Leftrightarrow M \in L \wedge \text{non } \exists X \in L \mid X > M$

Quelle est la complexité algorithmique de ce prédicat ? En déduire le prédicat **maximum_lin** qui est la version de complexité linéaire du prédicat **maximum**.

Base de données

Exercice 6:

Philippe est le père de Alexandre et Léonard ; Anne-Françoise est leur mère.

Franck est le père de Marie et Xavier1 ; Dominique est leur mère.

Frédéric1 est le père de Romain ; Fabienne est sa mère.

Paul est le père de Philippe, Dominique et Frédéric1 ; Odile est leur mère.

Martin est le père de Rosalie et Marcel ; Laurence est leur mère.

Xavier2 est le père de Maximilien ; Nadine est sa mère.

Daniel est le père de Laurence et Xavier2 ; Jacqueline est leur mère.

Frédéric2 est le père de Mathieu et Thomas ; Barbara est leur mère.

Olivier est le père de Juliette et Oscar ; Stéphanie est leur mère.

Tony est le père de Esteban et Lola ; Véronique est leur mère.

Serge est le père de Barbara, Olivier et Véronique ; Catherine est leur mère.

André est le père de Paul, Daniel, Serge et Geneviève1 ; Suzanne est leur mère.

René est le père d'Odile et Jacques ; Geneviève2 est leur mère.

Ecrire le prédicat **father(X,Y)** qui est vrai si **X** est le père de **Y**. Faire de même pour les prédicats mother, grandfather, grandmother, brother, sister, uncle, aunt, cousin, …, ancestor. Ecrire ces prédicats en anglais et non en français, pour leur ré-utilisation dans un exercice ultérieur.

Tester. (Par exemple : qui sont les ancêtres de Romain ? Qui sont les descendants de Suzanne ? Qui sont les frères de Genevieve1 ? Qui sont les cousins et cousines de Dominique ?)

Parcours d'arbre

Exercice 7 (évaluateur arithmétique) : écrire un prédicat **evalue(X,V)** qui est vrai si **V** est la valeur du terme **X** considéré comme représentant une expression arithmétique formée avec les symboles binaires **plus**, **moins**, **mult** et **div** et les constantes.

| ?- evalue(plus(10, moins(20,5)),X).

X = 25

yes

| ?- evalue(mult(plus(4,6),moins(5)),X).

X = -50

Yes

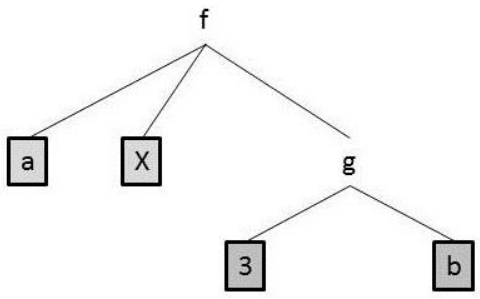
Exercice 8 (feuilles d'un arbre) : écrire un prédicat **feuilles(T,L)** qui est vrai si la liste **L** correspond à la liste ordonnée de gauche à droite des feuilles du terme/arbre **T** quelconque. Les feuilles sont les constantes et les variables libres (utiliser les prédicats **=../2** pour décomposer les termes, et **var/1** et **nonvar/1** pour tester si une variable est libre).

On fera attention au cas où il n'y a pas de reste de liste dans **=../2** puisque *a =.. L* donne *L = [a],* aussi utiliser l'implication **->/2**

| ?- feuilles(f(a,X,g(3,b)),L).

L = [a,X,3,b] ? ;

No



Récrire le prédicat **feuilles** en l'appelant **feuilles_lin** mais sans concaténation de listes, en utilisant un accumulateur (comme dans l'exercice 4).

Exercice 9 (co-routines) : modifier le prédicat **feuilles_lin** en l'appelant **feuilles_freeze** pour qu'il ne collecte que les feuilles correspondant à des constantes et qu'il bloque la collecte des feuilles sur les variables tant qu'elles ne sont pas instanciées (utiliser le prédicat de co-routinage freeze/2).

?- feuilles_freeze(f(a,X,g(b,3)),L).

L = [3, b|_A],

freeze(X, feuilles_f(X, [a], _A)).

?- feuilles_freeze(f(a,X,g(b,3)),L),X=h(5).

X = h(5),

L = [a, 5, b, 3] .