TP - Le langage Prolog/SWI-Prolog

Table des matières

Travail demandé

L'objectif de ce TP est de vous familiariser avec le langage Prolog, grâce à l'environnement SWI-Prolog [1]. Pour cela, vous allez tester des programmes vus en TD, modifier des programmes existants, ou en écrire de nouveaux.
Le test d'un programme suppose :

- le choix d'un jeu de données.
- la réalisation par vous-même de l'arbre de résolution supposé construit par Prolog, comme fait en TD.
- l'exécution du programme avec ces valeurs, en mode pas-à-pas et une comparaison de l'arbre que vous avez réalisé avec la trace de Prolog. Vous devez être capable d'expliquer les différences éventuelles.

A l'issue de ce TP, *et dans un délai maximal de 48h*, vous déposerez sur Moodle/LO12 votre compte-rendu. Celui-ci décrira, pour les exercices : 3, 4 et 6 :

- le code du programme testé
- La (ou les) requête(s) évaluée(s) (imposée(s) ou proposée(s) par vous-même).
- L'arbre de résolution construit par vos soins
- La trace d'exécution de Prolog
- L'explication des différences éventuelles constatées et une proposition de solution quand une différence est la cause d'une anomalie de fonctionnement

Exercice 1

Dans l'exemple de la famille (cours : D25), ajouter la ou les clause(s) permettant à Prolog de déterminer si un individu X est beau-parent d'un individu Y. Exemple : *beauParent(jules, jeanne)*. Prolog répond que c'est vrai.

Exercice 2

Exécuter : *reverse([3,2,1],[],L)* de l'exercice 3 du TD Prolog.

Exercice 3

A propos du programme de l'exercice 4 du TD (incrémentation d'une liste)

1. Test du programme

Interrogez Prolog avec des requêtes ayant pour arguments une variable et une liste "constante", puis uniquement des constantes.

2. Modification

Modifiez le programme de manière à ce qu'il échoue quand il tente d'incrémenter une liste vide. Testez ce nouveau programme.

1. `https://www.swi-prolog.org/`

Exercice 4

Executez pas à pas *ancetre(paul, lili)*, de la diapositive 21 du cours. Comparez avec l'arbre de résolution fait en cours. Même chose avec la variante du programme proposée dans la même diapositive.

Exercice 5

Testez le programme de l'exercice 5 du TD (ajout d'un élément à une liste sans doublon)

Exercice 6

Ecrivez et testez le programme de l'exercice 7 du TD (différence ensembliste de 2 listes)

Exercice 7

Ecrivez et testez le programme de l'exercice 9 du TD (séparation d'une liste de nombres, en une liste des positifs et une liste des négatifs)

Exercice 8

Testez le programme qui compte le nombre d'occurrences d'un élément dans une liste.

Travail demandé

L'objectif de ce TP est de développer vos compétences en programmation Prolog. Les programmes demandés vont permettre d'évaluer votre compréhension de la résolution réalisée par Prolog. L'écriture d'un programme suppose :

- l'écriture d'un court algorithme (comme vu en TD)
- l'écriture du programme Prolog correspondant
- l'évaluation du programme écrit :
  - le choix d'un jeu de données.
  - la réalisation par vous-même de l'arbre de résolution supposé construit par Prolog, comme fait en TD.
  - l'exécution du programme avec ces valeurs, en mode pas-à-pas et une comparaison de l'arbre que vous aviez réalisé avec la trace de Prolog. Vous devez être capable d'expliquer les différences éventuelles.

A l'issue de ce TP, *et dans un délai maximal de 48h*, vous déposerez sur Moodle/LO12 votre compte-rendu. Celui-ci décrira, pour les exercices : 1, 2, et 5 :

- votre programme Prolog
- La (ou les) requête(s) évaluée(s) (imposée(s) ou proposée(s) par vous-même).
- L'arbre de résolution construit par vos soins
- La trace d'exécution de Prolog
- L'explication des différences éventuelles constatées et une proposition de solution quand une différence est la cause d'une anomalie de fonctionnement

Exercice 1

Vérifiez si une liste est ordonnée. Exemple : *ordonnee([1,2,4,6]). True*

Exercice 2

Calculez le maximum d'une liste. Exemple : *Max([2,4,8,1,7],8).*

Exercice 3

Ajout d'un élément à la fin d'une liste. Exemple : *ajouteFin([2,4,8,1,7],6, [2,4,8,1,7,6]).*

Exercice 4

Calculez la somme d'une liste. Exemple : *Somme([1,5,10,14], 30).*

Exercice 5

Réalisez la concaténation de 2 listes. Exemple : *Conc([a,b,c],[d,e,f,g],[a,b,c,d,e,f,g]).*

Exercice 6

Calculez les permutations d'une liste.

Exemple *permutation([1,2],X).*
X=[1,2]
X=[2,1].

Indications Soit la liste [1,2]. Ses permutations sont : [1,2] et [2,1].
Les permutations de la liste [1,2,3] se construisent à partir de celles de la liste [1,2]...

Exercice 7

Un automate fini non déterministe est une machine abstraite qui lit une chaîne de symboles, qu'elle décide d'accepter ou de rejeter. L'automate possède un nombre fini d'états, et se trouve toujours dans un état donné. Il peut en changer en passant d'un état à un aure. Dans l'exemple de la figure 1, E1, E2, E3 et E4 sont les états de l'automate. Partant de l'état initial (E1 dans notre cas), l'automate passe d'un état à un autre au fur et à mesure qu'il lit la chaîne qu'on lui fournit en entrée. Si le premier symbole lu est a, alors il peut rester dans $E1$ ou transiter vers $E2$.Chaque transition est fonction du symbole lu, étiquetant chaque arc du graphe de transition.

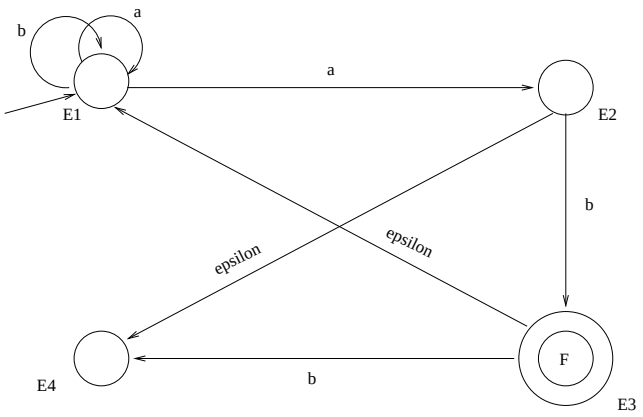

FIGURE 1 – Automate non déterministe

Quelques arcs portent l'étiquette *epsilon*. Ces arcs dénotent un passage invisible de l'automate vers un nouvel état, sans qu'aucun symbole ne soit lu. Ce passage est invisible car un observateur considérant l'automate comme une "boîte noire" ne pourrait se douter qu'une transition a eu lieu. L'état E3 est doublement entouré, ce qui signifie qu'il est un *état de satisfaction*. On dit que l'automate accepte la chaîne qu'on lui propose s'il existe une suite de transitions dans le graphe telle que les transitions successives effectuées grâce aux symboles de la chaîne, aboutissent à l'état de satisfaction.

On dit que l'automate accepte la chaîne qu'on lui propose s'il existe une suite de transitions dans le graphe telle que :

1. elle commence par l'état initial (ici, *E1*),

2. elle se termine par un état de satisfaction, et
3. les étiquettes des arcs du chemin corespondent aux symboles contenus dans la chaîne lue.

En prolog, parmi les clauses nécessaires, trois d'entre elles seront consacrées à la définition de la structure de l'automate :

— une clause définissant les états de satisfaction de l'automate
— une clause définissant les états de transition, telle que *trans(E1, X, E2)* signifie que l'on peut transitez de l'état *E1* vers l'état *E2* lorsqu'on lit le symbole *X*,
— une clause epsilon(*E1*, *E2*) indiquant une "epsilon-transition" depuis l'état *E1* vers *E2*.

Définition des clauses

Définir les clauses *satisfaction(X)*, *trans(E1, X, E2)*, *epsilon(E1, E2)* pour l'automate décrit.

Formulation des clauses

Ecrire les clauses *accepte(Etat, Chaîne)*. L'évaluation d'une telle clause est vraie lorsque, partant de l'état *Etat*, la lecture successive des symboles contenus dans *Chaîne*, permet d'aboutir dans un état de satisfaction.

Exemples :
accepte(e2,[b]). false.
accepte(e1,[a,b]). false.