

TP 1 – Initiation à Prolog

Prolog est un langage de *programmation logique*. Ceci veut dire qu'un programme ne se présentera pas sous forme d'une suite d'instruction (comme dans la *programmation impérative*), mais sous la forme d'un certain nombre d'« affirmations sur le monde » (des faits et des règles), formant une *base de connaissances*.

L'utilisation du programme se fera alors en *interrogeant* cette base de connaissances. Prolog dispose d'un puissant "moteur" lui permettant de déduire certains faits à partir de ceux que contient sa base de connaissances.

Une autre manière de voir Prolog est de dire qu'il s'agit d'un système expert générique en chaînage arrière.

Nous utiliserons SWI-Prolog¹, qui est une implémentation libre (sous licence LGPL) du langage Prolog.

Exercice 1

Soit le fait `bidule(toto, x, pierre, y)`. Quels sont les noms de relations, leur arité, les noms d'objets et les variables apparaissant dans ce fait ? Ce fait est-il clos ?

Exercice 2

Représenter à l'aide de faits les connaissances suivantes :

1. 5 est impair
2. 5 est supérieur à 3
3. Snoopy est un chien
4. Le cours de programmation logique a lieu le mercredi de 13 :30 à 15 :30, en grande amphi.

Exercice 3

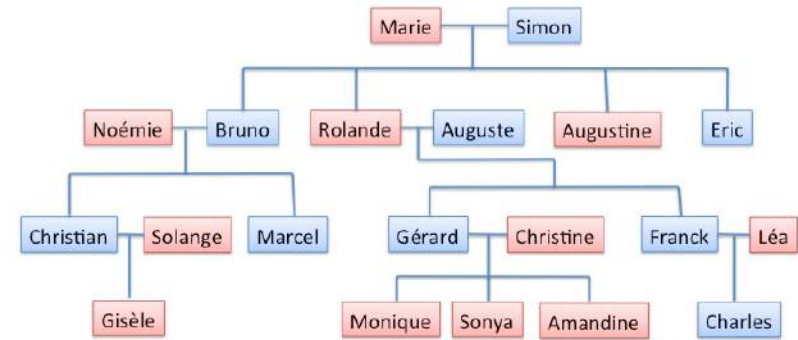
Exprimer, quand c'est possible, les connaissances suivantes sous forme de clauses Prolog :

1. Dominique est le père de Jeanne.
2. Tous les chiens sont des animaux.
3. Les chiens et les hommes sont des mammifères.
4. Le successeur d'un entier est un entier.
5. La relation frère(X, Y) (X est frère de Y) est symétrique.

¹ <http://www.swi-prolog.org/download/stable>

Exercice 4 – L'arbre généalogique

Soit l'arbre généalogique suivant :



1. Quels sont les prédicats (et leur arité) dont vous avez besoin pour définir cet arbre généalogie ?
2. Définir la base de faits décrivant cet arbre généalogique dans un fichier .pl et le charger dans Prolog.
Remarque : s'il y a des erreurs dans le chargement de votre fichier, vérifier bien que vous ayez mis des points à la fin de chaque ligne.
3. Calculer l'ensemble M(P) de votre programme.
4. Poser les questions suivantes à Prolog et prévoir ses réponses :
 - a. Augustine est-elle une femme ?
 - b. Léa est-il un homme ?
 - c. Jennifer est-elle une femme ? Que constatez vous ?
 - d. Quelles sont les mères ?
 - e. Quels sont les hommes ?
5. Ajouter au programme les définitions des relations suivantes grâce à des règles. N'oubliez pas de tester après chaque définition avec des questions vraies et fausses.
 - a. `parent(X, Y)` : X est parent de Y si X est mère de Y ou si X est père de Y.
 - b. `grand_parent(X, Y)` : X est un grand-parent de Y.
 - c. `est_mere(X)` : X est une mère.
 - d. `est_pere(X)` : X est un père.
 - e. `enfant(X, Y)` : X est enfant de Y.
 - f. `fille(X, Y)` : X est une fille de Y.
 - g. `filis(X, Y)` : X est un fils de Y.
 - h. `soeur(X, Y)` : X est une sœur de Y.*Remarque :* X et Y doivent être différents ($X \neq Y$).
 - i. `frere(X, Y)` : X est un frère de Y.
 - j. `frere_ou_soeur(X, Y)` : X est un frère ou une sœur de Y.

- k. `oncle(X, Y)` : X est un oncle de Y.
- l. `tante(X, Y)` : X est une tante de Y.
- m. `cousin(X, Y)` : X est un cousin de Y.
- n. `ancetre(X, Y)` : X est un ancêtre de Y.

Remarque : pensez à la récurrence.

6. Poser les questions suivantes à Prolog et prévoir ses réponses :
- a. Est-ce que Eric est un fils de Simon ?
 - b. Est-ce que Rolande est une sœur de Bruno ?
 - c. Est-ce que Rolande est une mère ?
 - d. Monique est-elle la cousine de Marcel ?
 - e. De qui Franck est-il le cousin ?
 - f. Marie et Eric ont-ils des enfants ensemble ?
 - g. Marie et Simon ont-ils des enfants ensemble ?

Exercice 5 – Les carnivores et les herbivores

Traduire en Prolog les énoncés suivants :

- Un animal cruel est carnivore.
- Un animal carnivore mange de la viande.
- Un animal herbivore mange de l'herbe.
- Un animal carnivore mange des animaux herbivores.
- Les herbivores et les carnivores boivent de l'eau.
- Un animal consomme ce qu'il boit ou ce qu'il mange.

Exercice 6 – Les substitutions

Montrer que :

- `sur(a, b)` est une instance de `sur(X, Y)`.
- `sur(X, Y)` est une instance de `sur(U, V)`.
- `sur(a, X)` n'est pas une instance de `sur(Y, b)`.

Exercice 7 – Arbres binaires entiers

Un **arbre binaire entier** est soit un arbre vide (représenté par l'atome `nil`), soit un arbre de valeur `v` (`v` est un entier), ayant 0, 1 ou 2 sous-arbres binaires ordonnés (gauche ou droite). On appelle **nœud** le triplet (sous-arbre gauche, valeur, sous-arbre droit), et on le représente par le type `n(G, V, D)`.

1. Définir le type arbre à l'aide de la relation `arbre/1`.
2. Définir la relation `element(X, A)` qui définit l'appartenance d'une valeur `X` à un arbre `A`.
3. Deux arbres `T1` et `T2` sont isomorphes si `T2` peut être obtenu à partir de `T1` en réordonnant les branches des sous-arbres de `T1`. Définir la relation `isomorphe(A1, A2)`.
4. Définir la relation `sous_arbre(S, A)` vraie si `S` est un sous-arbre de `A`.
5. La relation `preordre(A, Xs)` construit la liste `Xs` des valeurs contenues dans les nœuds de l'arbre `A` parcouru en préordre (i.e. en visitant d'abord la racine, puis le sous-arbre gauche, puis le sous-arbre droit).
6. Définir la relation `somme(A, S)` : `A` est un arbre binaire dont les nœuds sont des entiers, `S` est la somme des valeurs des nœuds de l'arbre.

Exercice 8 – Les listes

1. Quels sont les éléments des listes suivantes :

- a. `[]`
- b. `[[X, Y]]`
- c. `[[jean], paul]`
- d. `[[[]]]`
- e. `[[], []]`.

2. Ecrire la forme générale des listes suivantes :

- a. Listes à un élément, telles que cet élément est une liste à deux éléments.
- b. Listes à deux éléments, telles que le deuxième élément est la liste vide.
- c. Listes à trois éléments, telles que le deuxième élément est une liste à un élément.

3. Quelles sont les têtes et les queues des listes de la question 1 ?

4. Ecrire la forme générale des listes suivantes :

5. Représenter les listes suivantes en utilisant la notation `structure[X|Xs]` :

- a. La liste énumérée `[X1, X2, ..., Xn]` ;
- b. Les listes `[[X, Y]]` et `[[jean], paul]`.

Exercice 9 – Relation sur les listes

1. Définir de deux façons la relation `adjacent(X, Y, Zs)` : `X` et `Y` sont deux éléments consécutifs (dans cet ordre) de la liste `Zs`.
2. Définir de plusieurs manières les relations :
 - a. `prefixe(Xs, Ys)` : la liste `Xs` est une préfixe de la liste `Ys`.
 - b. `suffixe(Xs, Ys)` : la liste `Xs` est un suffixe de la liste `Ys`.
3. Définir la relation `double(Xs, Ys)` où `Ys` est la liste obtenue à partir de `Xs` en dupliquant les éléments de `Xs`; ainsi `double([a, b, c], [a, a, b, b, c, c])` est vrai.

Exercice 10 – Puzzle logique

Trois personnes, de nationalités différentes et pratiquant des sports différents, habitent dans trois maisons de couleurs distinctes. Ces trois maisons sont situées dans la même rue ; une des maisons est située au début de la rue, l'autre au milieu et la troisième au bout de la rue. Sachant que dans la maison verte on pratique la natation, que la maison verte est située dans la rue avant la maison de l'espagnol, que l'anglais habite la maison blanche, qui est située dans la rue avant celle où on pratique le football et que le tennisman habite au début de la rue, quel sport pratique le français et quelle est la nationalité de la personne qui habite la maison bleue ?

Pour répondre à la question posée, il faut trouver les caractéristiques (nationalité de l'habitant, couleur de la maison, sport pratiqué par l'habitant) de chacune des trois maisons. On choisit de représenter une solution par une liste de trois maisons : `[maison(N1,C1,S1), maison(N2,C2,S2), maison(N3,C3,S3)]`, les maisons étant ordonnées

suivant leur emplacement dans la rue, la première maison de la liste étant la première dans la rue. Le terme `maison(N,C,S)` représente une maison de nationalité `N`, de couleur `C` et de sport. Pour appliquer la méthode du « generate and test », on veut que la solution du problème soit fournie par l'interpréteur Prolog comme réponse à la question `trouver(Reponse)`, avec

```
trouver(Reponse) :- inconnue(Tableau),
                    renseignements_sur(Tableau),
                    questions(Tableau, Reponse).
```

1. Définir la relation `inconnue/1` qui détermine si un terme est bien de la forme choisie pour représenter une solution.

2. On suppose que la variable `Tableau` est de la forme ci-dessus. Définir les relations suivantes :

- `maison(Tableau, Maison)` : `Maison` est une maison de `Tableau`.
- `nationalite(Maison, N)` : le terme `N` est la nationalité de l'occupant de la maison `Maison`.
- `couleur(Maison, C)` : `C` est la couleur de `Maison`.
- `sport (Maison, S)` : `S` est le sport pratiqué par la personne habitant la maison `Maison`.
- `precede(Maison1, Maison2, Tableau)` : la maison `Maison1` de `Tableau` est située dans la rue avant la `Maison2` de `Tableau`.
- `premiere(Tableau, Maison)` : `Maison` est la maison de `tableau` située au début de la rue.

3. En utilisant ces relations, définir la clause `renseignements_sur(Tableau)` traduisant les contraintes de l'énoncé.

4. Définir de même la clause `questions(Tableau, [Sport_Français, Nationalite_maison_bleue])` qui extrait du tableau solution les réponses à la question posée.

5. Ecrire le programme complet, et répondre à la question : « Quel sport pratique le français et quelle est la nationalité de la personne qui habite la maison bleue ? ».

Memento Prolog

1. Quelques commandes utiles

```
listing.           Affiche le contenu de la base de connaissance.
listing(f).        Affiche les faits/règles avec f.
chdir(rep).         Change de répertoire de travail.
halt.              Sortir de l'interpréteur
```

2. Opérations arithmétiques

```
Res is Op1+Op2.     Addition
Res is Op1-Op2.     Soustraction
Res is Op1*Op2.     Multiplication
Res is Op1/Op2.     Division
Res is Op1//Op2.    Division entière
Res is Op1 mod Op2. Reste de la division entière
```

3. Opérations de comparaison

```
X<Y.               Si val(X) < val(Y) alors réussite sinon échec
X>Y.               Si val(X) > val(Y) alors réussite sinon échec
X<=Y.              Si val(X) ≤ val(Y) alors réussite sinon échec
X>=Y.              Si val(X) ≥ val(Y) alors réussite sinon échec
X:=Y.              Si val(X) = val(Y) alors réussite sinon échec
X\=Y.              Si val(X) ≠ val(Y) alors réussite sinon échec
```

4. Entrées/sorties

```
write(X).           Ecriture de la représentation du terme défini par X
print(X).           Ecriture de la valeur de la chaîne de caractères
nl.                 Saut de ligne
tab(n).             Imprime n caractères espace
consult(nom_fichier). ou [nom_fichier].
                    Charge le contenu du fichier nom_fichier
reconsult(nom_fichier).
                    Recharge le contenu du fichier avec écrasement
```

5. Modification de programmes

```
abolish(nom,arite).
                    Permet d'enlever de la base toutes les clauses spécifiées par nom
                    et d'arité arite
assert(clause).     Permet de rajouter la clause spécifiée à la fin de la base de données
                    courante
asserta(clause).    Permet de rajouter la clause spécifiée en tête du paquet de clause
                    définissant un prédicat
assertz(clause).    Permet de rajouter la clause spécifiée à la fin du paquet de clause
```

définissant un prédicat

6. Débogage

Pour comprendre ce qui se passe, cela sert notamment lorsqu'il y a des erreurs, vous pouvez utiliser les capacités de débogage de SWI-Prolog.

Si vous saisissez : **trace** , `parent(x)`.

Ceci va lancer la requête `parent(x)`., mais en demandant en plus une trace d'exécution. Prolog vous répondra quelque chose comme `Call: (8) parent(_G157)?`, ce qui signifie qu'il a projet d'appeler le prédicat `parent` avec une variable comme argument.

Pour savoir quelles sont vos possibilités à ce moment précis, tapez **h**. Prolog vous donnera alors la liste des commandes disponibles.

Pour afficher l'étape suivante de l'exécution, tapez sur la barre d'espacement (ou *enter*).

Continuez à suivre pas à pas l'exécution de votre programme en frappant la barre d'espacement chaque fois que nécessaire, jusqu'à la fin de l'exécution. Vérifiez que Prolog passe bien par les étapes que vous lui avez demandées.

Le debug en mode texte est un plaisir pour les yeux, mais SWI-Prolog (dans sa version pour Windows seulement) propose aussi une version graphique du débogueur. Dans le menu "Debug" choisissez "Graphical Debugger" (ou tapez simplement `guitracer`.), puis relancez la commande ci-dessus. Une belle fenêtre apparaît vous permettant de suivre l'exécution de votre programme.