# TP2 – Introduction à Prolog

INF8215 – Hivers 2023

## Consignes

- Le devoir doit être fait par groupe de 3 au maximum.
- La date de remise est le **18 mars** sur Moodle (avant minuit) pour tous les groupes.
- Vous devrez remettre un seul fichier ***.pl*** qui contient toutes les solutions aux exercices et le nommé selon le format matricule1_matricule2_matricule3_TP2.pl .
- Indiquez vos noms et matricules en commentaires en haut du fichier soumis.
- Il n'y a aucun rapport à remettre pour ce devoir.
- Toutes les consignes générales du cours (interdiction de plagiat, etc.) s'appliquent pour ce devoir.
- Il est permis (et encouragé) de discuter de vos pistes de solution avec les autres groupes. Par contre, il est formellement interdit de reprendre le code d'un autre groupe ou de copier un code déjà existant (StackOverflow ou autre). Tout cas de plagiat sera sanctionné de la note minimale pour le TP.

## 1. Introduction

Vous avez vu dans votre scolarité un certain nombre de langages de programmation comme le python, le C, le C++, le Java, le Matlab, et bien d'autres. Les langages de programmation sont classés par paradigmes. Ceux-ci vont beaucoup influencer la manière de coder. Parmi les paradigmes les plus courants, on trouve l'orienté objet. Ce paradigme permet de structurer un gros projet et d'éviter de se perdre dans les fonctions. Java, python, C et les autres reposent sur ce paradigme. Un autre paradigme est le fonctionnel. Dans ce cas-ci, chaque fonction va faire appel à d'autres fonctions pour s'exécuter. Certains langages comme python, Matlab ou R offrent de programmer selon les deux paradigmes. Prolog, quant à lui, est aussi un langage fonctionnel. La particularité de prolog est son aspect logique. Nous allons voir ce que cela veut dire. Prolog est dit fonctionnel logique, parce qu'il ne se présente pas comme une suite d'instructions, mais comme un certain nombre d'affirmations sur le monde. L'utilisation du programme se fera par interrogation d'une base de connaissance. Prolog parcourra cette base de connaissance par lui-même pour trouver la solution.

## 2. Ouverture de Prolog

Sur les machines du laboratoire, ouvrir un terminal (ctrl+alt+t) et taper 'swipl', pour quitter taper 'halt.' (ne pas oublier le point). Vous pouvez télécharger sur votre machine swi-prolog (licence libre) à partir du lien suivant : https://www.swi-prolog.org/Download.html .

# 3. Les éléments de base

## 3.1 Base de connaissance

Ouvrez maintenant un fichier texte et insérez le texte suivant (attention à la casse):

```
animal(chien).
animal(chat).
prenom(paul).
prenom(pierre).
prenom(jean).
possede(jean,chat).
possede(pierre,chien).
possede(pierre,cheval).
amis(pierre,jean).
amis(jean,pierre).
amis(jean,paul).
amis(pierre,paul).
amis(paul,jacques).
```

Ce fichier, appelé base de connaissance ou base de faits, est utilisé par prolog pour résoudre les problèmes. On peut le considérer comme toute la connaissance du monde de prolog.

Enregistrez ce fichier sous le nom *base1.pl*. Nous allons maintenant l'ouvrir sous prolog. Sur Windows, vous pouvez essayer de cliquer deux fois sur le fichier. Sinon, placez-vous avec un terminal dans le dossier contenant la base de connaissance, puis lancez prolog avec ''swipl -s base1.pl''. Le -s base1.pl permet d'ouvrir la base de connaissance au lancement. Vous pourrez l'ouvrir depuis prolog avec ''consult(nom_de_la_base)''. ou [nom_de_la_base]. (attention à ne pas oublier le point, omettre l'extension dans le nom du fichier). Si des erreurs syntaxiques se sont glissées dans le fichier, prolog vous le signalera ici. Il faudra ensuite recharger le fichier.

Quelques points pratiques:

- toutes les instructions finissent par un point (équivalent du point virgule en java)
- Pour mettre un commentaire dans la base de connaissance, utiliser %

## 3.2 Les Requêtes

Prolog se divise donc en deux parties : d'une part, la base de connaissance et d'autre part, les questions (requêtes) que l'on va lui poser. Ainsi on peut maintenant entrer (dans la console prolog) :

-? prenom(pierre).

Le -? est déjà affiché, il ne faut pas le remettre. Prolog devrait vous répondre **true**., car il est spécifié dans la base de connaissance que Pierre est un prénom. Maintenant, tapez :

-? prenom(jacques).

prolog devrait maintenant vous répondre **false**., car bien que pour vous Jacques est bien un prénom, la base de connaissance ne peut le savoir puisque cela ne lui est pas précisé. On touche ici à un point important en prolog. Prolog fait l'hypothèse de connaissance totale du monde. Ainsi tout fait qui ne lui est pas précisé est considéré comme faux. Maintenant tapez :

-? chat(tom).

Prolog affiche maintenant un message d'erreur disant qu'il ne connaît pas chat/1. Cette erreur est différente de la précédente. Tout à l'heure, on était face à une propriété connue dont on ne savait pas si elle était vraie, et qui est donc considérée fausse. Maintenant, on se retrouve face à une propriété inconnue.

## 3.3 Les Termes

Dans un langage de programmation classique, on utilise des types primitifs ainsi que des variables ou objets. Dans prolog, on manipule des termes qui peuvent être de diverses natures. Il y a les termes simples comme: chat., chien. ou pierre., les nombres (entiers) et les chaines de caractères, entourées de guillemets ('ma chaine'). Les termes simples commencent par une minuscule.

On peut ajouter ensuite les termes composés, comme prenom(pierre).. Ces termes sont comme les termes simples, mais prennent un ou plusieurs arguments. Ils commencent toujours par une minuscule.

Enfin, il y a les variables. Celles-ci vont nous permettre de poser des questions plus poussées à prolog.

Tapez prenom(X) dans prolog. Prolog devrait vous répondre X = paul. Tapez maintenant **;** Cette fois-ci, il devrait ajouter X = pierre. Si vous le refaites une troisième fois, il va afficher X = jean et vous rendre la main. La variable ici est entendue dans son sens mathématique : elle vaut tous les éléments possibles qui satisfont une certaine condition. Ici, on demande à prolog : " Donne-moi tous les X tels que prenom(x) est vrai (existe, est dans la base de connaissance) "

Prolog identifie X à une variable parce qu'il commence par une majuscule. Tout mot commençant avec une majuscule ou par _ est considéré comme une variable.

Essayez maintenant de répondre aux questions suivantes avec Prolog :

1. Qui possède le chat?

2. Quels animaux Pierre possède-t-il?

## 3.4 Les Opérateurs Logiques

Pour pouvoir faire des choses intéressantes avec Prolog, il faut pouvoir faire des opérations logiques. Les deux opérations logiques de base sont définies de la façon suivante :

ET : On utilise la virgule : predicat1,predicat2

OU : On a ici deux choix: soit on utilise le point virgule : predicat1;predicat2, soit on passe tout simplement à la ligne, en créant une nouvelle entrée dans la base de connaissance. Cela a pour conséquence pratique que les disjonctions de cas sont beaucoup utilisées pour implémenter des fonctions ou règles complexes.

Cherchez maintenant s'il y a des amitiés réciproques (utilisez des variables).

## 3.5 Les Règles

Jusqu'ici Prolog n'est pas très intéressant, et ressemble, en moins pratique et plus lent, à une base de données. Il nous manque l'essentiel pour pouvoir programmer : les règles. Retournez dans le fichier base1.pl et rajoutez la ligne suivante à la suite des autres. amis_2(X,Z) :- amis(X , Y) , amis(Y,Z).

Cette ligne signifie que la partie gauche est vraie si la partie droite l'est (mais pas inversement), c'est une implication. En français, cette ligne représente le fameux dicton "l'ami de mon ami est mon ami". Enregistrez le fichier et interrogez prolog sur les amis_2 de pierre. (tapez amis_2(pierre,X).)

## 3.6 Débogage

Prolog propose un outil de débogage, qui permet de tracer toutes les opérations effectuées. Dans la règle proposée ci-dessus, on souhaiterait que quelqu'un ne puisse pas être ami avec lui-même. Cependant, le programme nous répond que cela est le cas pour Pierre.

Tapez trace, amis_2(pierre,X).

Prolog affiche ici toutes les opérations qu'il effectue. Un call correspond à l'appel d'une fonction, un exit à une remontée dans l'arbre, un redo à une reprise de la recherche. Corrigez maintenant le programme, en ajoutant à la fin X \= Z. Testez maintenant votre base de données.

## 3.7 Exercice

Voici un arbre généalogique :

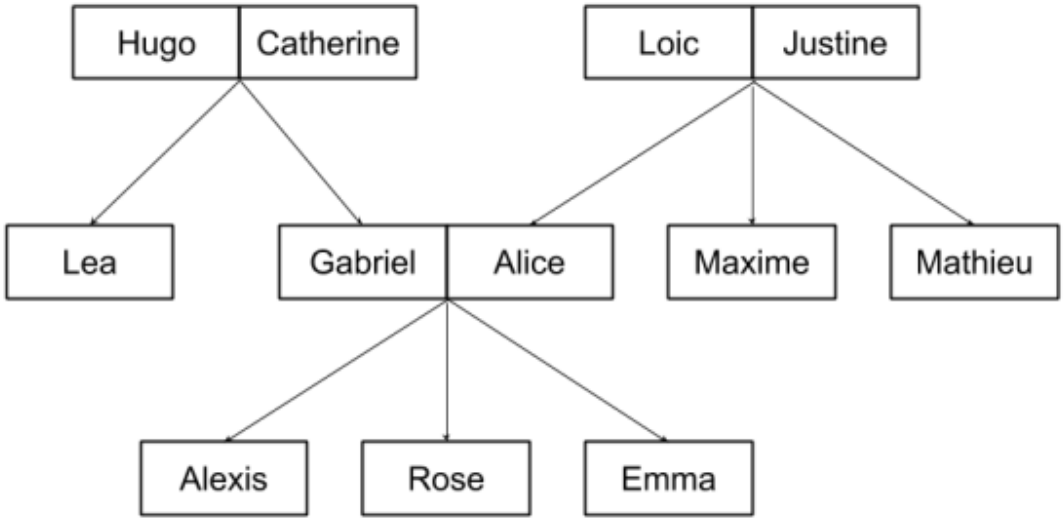


Figure 1: arbre généalogique

1. Écrivez une série de faits décrivant les relations de base de cet arbre. Il faudra définir les relations homme, femme et parent.
2. Définissez l'ensemble des règles suivantes. On conviendra que pred(X,Y) signifie que X est le 'pred' de Y.
   - enfant(X,Y)
   - fille(X,Y)
   - fils(X,Y)
   - mere(X,Y)
   - pere(X,Y)
   - oncle(X,Y)
   - tante(X,Y)
   - grand parent(X,Y)
   - grand pere(X,Y)
   - grand mere(X,Y)
   - petit enfants(X,Y)
   - petite fille(X,Y)
   - petit fils(X,Y)
   - frere(X,Y)
   - soeur(X,Y)

   Pensez à tester vos règles au fur et à mesure

# 4. Fonctionnement de Prolog et Programmation

## 4.1 Variable anonyme

On veut maintenant, toujours sur la base *base1.pl*, définir un prédicat qui est vrai si une personne a des amis. On peut alors écrire `amis(X) :- amis(X,Y).` Lorsque l'on écrit cela, à la compilation, prolog affiche un warning, disant que l'on a une variable singleton. Cela signifie simplement qu'une variable n'apparaît qu'une fois dans une formule et qu'il est donc inutile de lui donner un nom. Prolog nous incite alors à la renommer avec un "_". Ce warning peut aussi être déclenché par une erreur dans le code qu'il nous faudrait corriger. C'est pour cela qu'il est intéressant de remplacer qu'une fois les variables utilisées en variables anonymes (_). Remplacez alors la règle par la suivante: `amis(X) :- amis(X,_).`

**Remarque** : Chaque variable anonyme est indépendante ainsi, `amis(X,X)` cherchera les personnes qui sont amies avec elles-mêmes alors que `amis(_,_)` cherchera toutes les combinaisons d'amis possibles.

## 4.2 Unification

Les variables manipulées par prolog sont différentes que celles utilisées dans un langage classique. Elles se rapprochent plus des variables mathématiques. Si on tape dans prolog `X=Y, Y=2.`, prolog nous répond `X=2`, alors que dans n'importe quel autre langage, il aurait planté, parce qu'on utilise Y avant de l'avoir défini. D'autre part, l'instruction `X=Y, Y=2, X=1.` a pour résultat **false**. Dans le langage classique, `Y=2` et `X=1` correspondent à des affectations de variables et ne posent donc aucun problème. Ici, les variables sont immuables pendant toute leur durée de vie, c'est-à-dire qu'elles ne peuvent être réaffectées ou remises à zéro. Une variable ne représente pas en fait un objet particulier, mais l'ensemble des valeurs satisfaisant une ou plusieurs conditions. En fait `X=Y` se lit 'X s'unifie à Y'.

Cela permet aussi de faire du pattern matching, essayez:

- `X+Y=1+5.`
- `X+Y=f(x)*5+7/3.`
- `X*3=5*Y`

Mais cela à une contrepartie. Ainsi `1+3=2+2` conduira à une erreur. Il faudra plutôt écrire `x is 1+3` pour forcer l'évaluation. Cela conduit à avoir des opérateurs assez différents selon ce qu'on cherche à faire:

| | |
|---|---|
| `x=y` | sera utilisé pour unifier X et Y (renvoie true s'ils sont unifiables) |
| `x\= y` | renvoie true si x et y ne sont pas unifiables, ne réalise pas d'unification |
| `x==y` | renvoie true si X et Y sont égaux (pas d'unification) |
| `x\==y` | renvoie true si X et Y sont différents |
| `x=:=y` | calculera une égalité arithmétique (attention demande à ce que les variables soient instanciés) |
| `x is y` | unifie X à l'évaluation arithmétique de Y (non réversible) |
| `x>y, x>=y` | force l'évaluation de x et y et renvoie le résultat attendu |

## 4.3 Exercices

1. Écrivez une règle sum(X,Y,R) qui permet d'obtenir la somme de deux nombres (R=X+Y)

2. Écrivez une règle max2(X,Y,M) qui calcule le maximum de 2 nombres (M=max(X,Y))

3. Écrivez ensuite une règle qui calcule le maximum de trois nombres max3(X,Y,Z,M)

## 4.4 Le chainage arrière

Pour comprendre comment prolog fait pour résoudre les demandes que vous lui faites, lisez le document suivant. Il vous aidera ensuite à optimiser vos programmes, et à comprendre les bugs. http://cs.union.edu/~striegnk/learn-prolog-now/html/node20.html

## 4.5 Exercices

On vous demande de remplir, à l'aide de prolog, la grille ci-dessous avec les mots suivants de façon à ce que chaque case ne contienne qu'une lettre :

- abalone
- abandon
- enhance
- anagram
- connect
- elegant

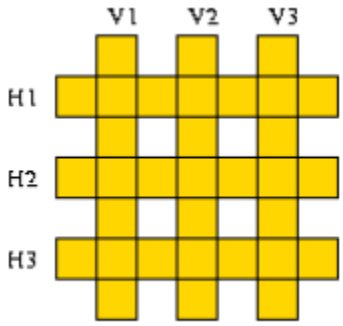


Figure 2: grille à compléter

Définissez pour cela un prédicat crossword(V1,V2,V3,H1,H2,H3) qui donne la solution du problème. Voici un exemple d'exécution et de solution attendu :

```
1 ?- crossword(V1,V2,V3,H1,H2,H3).
V1 = abalone,
V2 = anagram,
V3 = connect,
H1 = abandon,
H2 = elegant,
H3 = enhance
```

## 4.6 Les listes

Les listes sont un élément essentiel en prolog. Une liste peut être assimilée à un prédicat binaire de la forme liste(premier élément, liste(deuxième élément, .... liste(dernier élément, liste vide)...)) Pour rendre son utilisation plus pratique, un objet liste a été intégré à prolog. Il se présente de la manière suivante:

- [] est la liste vide
- [H] est une liste contenant un seul élément H
- [H|T] est une liste, où H est le premier élément et T la suite de la liste
- [H1, H2, H3, ...|T] est la liste qui commence par H1, ... ,Hn et fini par T
- les éléments d'une liste peuvent être complètement différents. [f(x)] est une liste. On peut mélanger des prédicats, des nombres, et des chaînes de caractères.

## 4.7 La récursivité

Avec les listes, s'ajoute la notion de récursivité, essentielle en prolog. En effet, prolog ne présente pas de boucle for ou while comme dans les autres langages. Pour boucler, on utilise alors une approche récursive. Cela demande une gymnastique mentale, mais la plupart des raisonnements que l'on peut faire sur une boucle for sont transposables sur un raisonnement récursif. Par exemple, si on veut calculer la longueur d'une liste, on écrira :

- longueur([],0).
- longueur([ |T],N) :- longueur(T,N1),N is N1+1.

On remarque que dans ce raisonnement, il y a toujours au moins deux cas : le (ou les) cas de base (ici la liste est vide) et le cas récursif qui appelle le cas inférieur. Un raisonnement récursif se base le plus souvent sur une règle simple, comme le fait que la longueur d'une liste L vaut 1 + la longueur de L privée de son premier élément. Attention à ne pas oublier les cas de base.

## 4.8 Exercices

- Écrivez une méthode qui calcule le maximum max(L,M) d'une liste (utilisez la fonction définie plus tôt)
- Écrivez une fonction qui calcule la somme somme(L,S) des éléments d'une liste
- Écrivez une fonction qui renvoie le nième élément d'une liste nth(N,L,R)
- Écrivez une fonction zip(L1,L2,R) qui renvoie une liste de couples par exemple zip([1,2,3],[a,b,c],R) retourne R = [[1,a],[2,b],[3,c]]
- Écrivez une fonction enumerate(N,L) qui renvoie une liste avec tous les nombres de 0 à N-1

• **Bonus :** Écrivez une fonction rend_monnaie(Argent,Prix) qui affiche les pièces à rendre si on paie le prix en espèces. Pour afficher du texte utilisez les fonctions write() et nl. Vous devrez afficher quelque chose qui ressemble à

```
A rendre :
1 piece de 0.05
1 piece de 0.10
1 piece de 0.25
1 piece de 1
1 piece de 2
true
```

Vous pouvez utiliser la fonction floor qui renvoie la partie entière. Attention aussi aux erreurs d'arrondi.

# 5. Barème de correction

Seul les exercices 3.7, 4.3, 4.5 et 4.8 seront corrigés. Un seul fichier ***.pl*** doit être remis qui contient les solutions à tous ces exercices. Tout travail en retard sera pénalisé de 10% par jour de retard.

**Barème :**

Exercice 3.7 : 5 points

Exercice 4.3 : 3 points

Exercice 4.5 : 5 points

Exercice 4.8 : 5 points

Respect des consignes : 2 points

* Ce TP a été écrit par Pierre-Emmanuel Savoie, inspiré du travail de Pierre Hulot et Matthieu Miguet.