# 1 Introduction to Prolog

You are probably used to some imperative languages as python, C or Java. Programming languages are classed by paradigm, the most common one is the object oriented, as it allows to easily build big projects. Some languages as python, R or matlab allows also their user to use the functional paradigm. Prolog is functional declarative language. This means that it uses one functions and is not iterative. The specificity of Prolog is its logical aspect. Prolog uses some affirmation on the world to solve mostly combinatorial problems. TO do so it relies on a data base containing the representation of the world. This database is scanned to find the answers to some questions.

# 2 Opening of prolog

On the lab computers, open a terminal (ctrl+alt+t) and write 'swipl'. To quit prolog write 'halt.' (don't forget the point).
You can also download swi-prolog on your laptop (free license).

# 3 Basics elements

## 3.1 data base

Open a text file and insert the following text (warning : prolog is very case sensitive)

```
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).
```

This file is called knowledge base or fact base, and it is used in prolog to solve some problems. It can be considered as prolog's knowledge of the world.
Save this file as *base1.pl*. We are now going to open it into prolog. With the terminal

go to directory where the base file is, and type `swipl -s base1.pl`. The `-s base1.pl` command is used to open the database on the opening of prolog, you can also load it using `[database_name].` or `consult(database_name).` (don't forget the point, omit the extension). Prolog will signal all syntax error on the opening. If there are some errors, the file will not open.
Some practical points:

- all instruction end with a point (equivalent to the ';' in java)
- use `%` to comment

## 3.2 Requests

Prolog is divided into two parts, the first one is the database, the second one is the execution, triggered by some questions, or requests. Try to write into the console
`-? prenom(pierre).`
prolog should answer **true.**, Prolog knows that pierre is a name because it is precised into the database.
Now write :
`-? prenom(jacques).`
Prolog should answer **false.**. While we are aware that jacques is a first name, prolog is not, and prolog assumes that if he doesn't know something then this fact is false. Prolog assumes that he knows every thing about the wold.

Now Write :
`-? chat(tom).`
Prolog now print an error message saying that he doesn't know the predicate `chat/1`. This error is different from the previous one. Before we had a known property (prenom), but we didn't know if it was true or false and though considered false, now we have an unknown property.

Remark : every property written in the database is called a predicate.

## 3.3 Terms

In usuals programming languges we used primitive type variables and objects, in prolog, we use terms that can be of multiple nature. There are simple terms as `chat.`, `chien.` or `pierre.`, Numbers (integers or reals), and strings (surrounded by ")
Note : simple terms begin with a lowercase.

Then we can add composed terms as `prénom(pierre).`. These terms are similar to simple terms but can take one or more arguments. They always begin we a lowercase. The same property name can be used to indicate different properties if the number of argument differ.

Finally there are the variables. Variables makes possible to ask question to prolog.
write `prenom(X)` into prolog. Prolog should answer `X = paul`. Then type ';' or space. Now it should have displayed `X = pierre`. Do it until prolog stops printing. The last printed line

should be false. We will understand this later.

The important thing here is that a variable is intended in its mathematical sens. The value of a variable is the all the values that satisfies some property. in the previous line we asked to prolog: tell me all the values of X such that prenom(X) is true. Prolog knows that X is a variable because it begins with a capital letter.

Try yo answer the following questions with prolog:

1. who owns the cat (chat)?
2. what animals does pierre own?

## 3.4 Logic operations

To do interestant thing with prolog we need to be able to do so logical operation. The main two operations are define this way:

AND : use the coma : `predicat1,predicat2`

OR : there are two choices it is possible tu ise the ';', the other zqy is to create a new entry to the database. This means that we will use a lot case matching to implement function and rules.

Look now for reciprocal friendships (use variables)

## 3.5 Rules

The main ingredient to prolog is still missing. For now we can't do anything interesting with prolog. The rules are the equivalent of functions in standard programming. Get back to `base1.pl` and add the following line at the end of the file:
`amis_2(X,Z) :- amis(X , Y) , amis(Y,Z).`
this line means that if the right part is true, then the left one is true. it's an implication. Notice the variable in the rule. Save the file and ask prolog about amis_2 of pierre. (write `amis_2(pierre,X).`)

## 3.6 Debug

prolog proposes a debug tool, that allows us to trace of operations. For exemple the previous rule tells us that pierre is friends of pierre. We want the program to tell us all friends of pierre that are not pierre. Write on the terminal : `trace, amis_2(pierre,X)`
Prolog displays of operation, that allows us to follow the tree search performed by prolog. Add $X\bar{Z}$ at the end of the rule to correct it. Test it on your database.

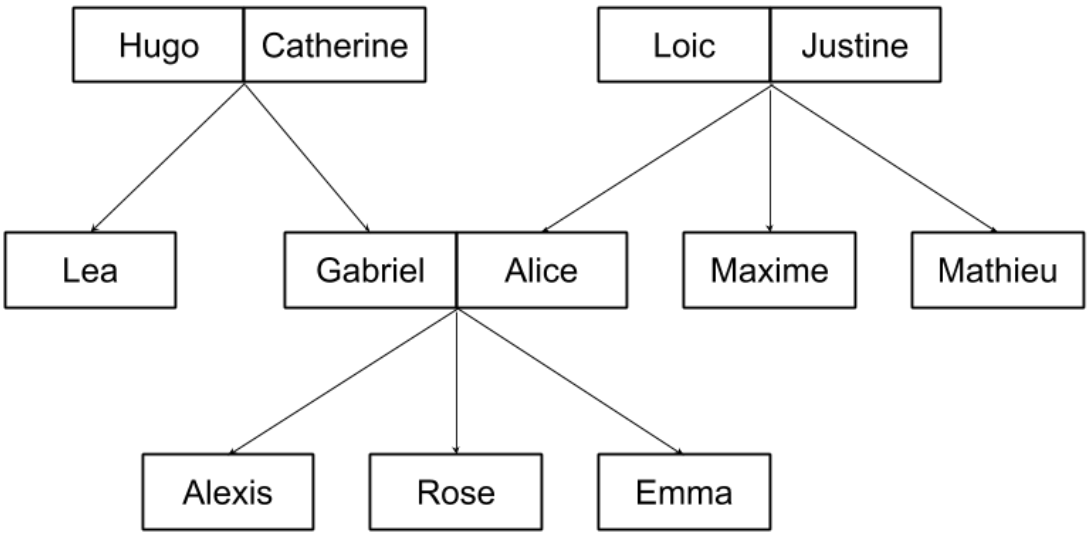


Figure 1: genealogical tree

## 3.7 Exercise

this is a genealogical tree :

1. Write a serie of facts decribing the relations of this tree, using the following relations : `man`, `woman` et `parent`.
2. defineall the follwing rules. We assume that pred(X,Y) means that X is the 'pred' of Y.
   - child(X,Y)
   - daughter(X,Y)
   - son(X,Y)
   - mother(X,Y)
   - pfather(X,Y)
   - uncle(X,Y)
   - aunt(X,Y)
   - grand_parent(X,Y)
   - grand_father(X,Y)
   - grand_mother(X,Y)
   - grand_child(X,Y)
   - grand_daughter(X,Y)

- grand_son(X,Y)
- nephew(X,Y)
- niece(X,Y)
- brother(X,Y)
- sister(X,Y)

test your rules as your writing them.

# 4 Prolog et programming

## 4.1 Anonymous Variable

Let's define a predicate in the *base1.pl* file that tells us if someone has some friends
Then we can write `amis(X) :- amis(X,Y)`. This formula is correct, but on the compilation, Prolog prompt a warning, saying that there is a singleton variable. This means that a variable appears only once is formula. This warning can be triggered for several reasons. If we've done that on purpose, we should replace the variable with an anonymous variable : "_". This warning can also be raised because we've made a mistake in our code. to keep track of these mistakes, it is important to replace anonimous variables with _. Replace the formula with following one `amis(X) :- amis(X,_)`.

**Remark** Each anonimous varaible is independant , this means that, `amis(X,X)` will search for people friends with themselfs and `amis(_,_)` will search for any two variable X, Y such that `amis(X,Y)` is true.

## 4.2 Unification

Variables in prolog are different from the ones used in iterative programming. They are more similar to mathematical ones. The instruction `X=Y, Y=2.` is correct in prolog and the result is X=2, However in a imperative language, this line will throw an error because Y is used before assignment.
Consider now the line `X=Y, Y=2, X=1.`, prolog will say that this line is false because X connot be equal to 2 and to 1. In an iterative language, this double affectation is not a problem because '=' means that X is now define as, and just erase the previous definition. In prolog variable are immutable, they cannot be reset. A prolog variable does not represent a specific object but a set of object satisfying some condition. X=Y is read as 'X unifies to Y'.
this paradigm makes the pattern matching possible :
try:

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

the counterpart of this convention is that `1+3=2+2` will be false because 1!=2 and 3!=2, to force the evaluation of an arithmetic expression we should use `x is 1+3`. There are also different operator, depending on what we want to do :

**x=y** is used to unify X and Y (return true if they are unifiable)

**x\= y** returns true if x et y are not unifiable, and doesn't unify

**x==y** returns true if X and Y are equals (no unification)

**x\==y** returns true if X and Y are different

**x=:=y** will compute an arithmetic equality (warning, variable needs to be instantiated)

**x is y** unifies X à to the arithmetic value of Y (one directional operation)

**x<y, x=<y** force the evaluation of x and y and returns the expected result

**x>y, x>=y** force the evaluation of x and y and returns the expected result

## 4.3 Exercises

1. write a rule `sum(X,Y,R)` that computes the sum of two numbers (R=X+Y)
2. write a rule `max2(X,Y,M)` that computes the maximum of two numbers (M=max(X,Y)), then a rule that computes the maximum of three numbers `max2(X,Y,Z,M)`
3. write a rule d(F,X,G) that computes the derivative of F reguarding to X : $G(X) = \frac{dF(X)}{dX}$. We suppose that F is polynomial. The first case will be d(x,x,1) is true. Use `atomic(C), C\=X` to test if C is constant. Use the derivation rules of the sum and the product. Finally compute d(5*x∧ 3+3,x,R). The result doesn't need to be simplified.

**Remark :** the previous derivation formula could be used to integrate, but some operation are not reversible.

## 4.4 Backward chaining

To understand how prolog solve the requests on the database, read the following document. This will hekp you organizing your programs and understanding bugs. `http://cs.union.edu/~striegnk/learn-prolog-now/html/node20.html`

## 4.5 Exercises

### 4.5.1 Exercise

Consider the following database:

```
pere(hugo,gabriel).
pere(gabriel,alexis).
pere(alexis,paul).
pere(paul,andre).
ancetre(X,Y):-pere(X,Y).
ancetre(X,Y):-pere(X,Z),ancetre(Z,Y).
```

write the search tree corresponding to the request

```
ancetre(hugo,X).
```

give all prolog answers. To save space you can use abbreviations. You should precise them in the report. Use _1, _2, _3, ... for the variable names.

### 4.5.2 Exercise

Fill the grid with the following words using prolog.

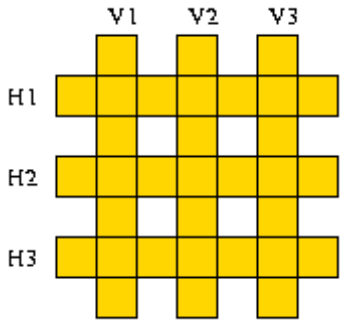


Figure 2: grid to complete

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

Define a predicate `crossword(V1,V2,V3,H1,H2,H3)` that gives the problem solution. Begin defining the following predicates:

```
word(abalone, a, b, a, l, o, n, e).
word(abandon, a, b, a, n, d, o, n).
word(enhance, e, n, h, a, n, c, e).
word(anagram, a, n, a, g, r, a, m).
word(connect, c, o, n, n, e, c, t).
word(elegant, e, l, e, g, a, n, t).
```

## 4.6 Lists

Lists are essential in prolog. A list is similar to a binary predicate that looks like : `list(first element, list(second element, .... list(last element, empty list)...))` To use it more effectively a list object is integrated in prolog :

- [] is the empty list,
- $[H]$ is a list containing one element H,
- $[H|T]$ is a list where H is the first element and T the tail of the list,
- $[H1, H2, H3, ...|T]$ is the list beginning with H1, ...Hn and with the tail T.
- The element contained in a list can be completely different :$[f(x)]$ is a list. we can mix predicates, numbers and strings.

## 4.7 Recursivity

Functional paradigm + list leads to the recursivity notion. this idea is essential in prolog. Prolog doesn't have for or while loops, so the only way of doing a loop is using recursivity. This difference needs an adaptation, but most loop reasoning are transposable to recursive paradigm. For example to compute the length of a list we will write:

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

In such a reasoning there are at least two cases : the base case and the recursive case. Most of the time the recursive reasoning uses a very simple rule (here the length of a list is 1+ the length of the list without its first element. Don't forget the base case or the program will not end.

## 4.8 Exercises

- write a method that computes the maximum over a list `max(L,M)` you can use the method defined before
- write a method that computes the sum of all elements in a list `somme(L,S)`
- write a method that returns the n-th element of a list `nth(N,L,R)`
- write a method that zips two lists `zip(L1,L2,R)`. example : `zip([1,2,3],[a,b,c],R).` gives `R = [[1,a],[2,b],[3,c]]`
- write a method that enumerates all number from 0 to N in the list L `enumerate(N,L)`

- write a method `give_change(Money,Price)` that prints the change to do if we pay Price with Money. To print use the functions `write()` and `nl`. ou should print something like:

```
A rendre :
1 coin of 0.05
1 coin of 0.10
1 coin of 0.25
1 coin of 1
1 coin of 2
true
```

  You can use the function 'floor' that returns the integer part. be careful with round-off error.

## 4.9 CLPFD and constraint programming

The clpfd library makes constraint programming intuitive with prolog. clpfd brings arithmetics and ensemble operators. Check this website to familirize your self with these operators. `http://www.swi-prolog.org/man/clpfd.html` To include the library, write at the beginning of the database file :
`?- use_module(library(clpfd)).`
Use this library to solve the following problems

### 4.9.1 problem 1

given :

- the British lives in the red house.
- The Spanish has a jaguar.
- The Japanese lives on the right of the snail owner.
- the snail owner lives on the left of the blue house.

and that all three people live separately, that each one if them owns an animal and an house, find who owns the snake.

### 4.9.2 problem 2

similarly solve the following problem :

- The British lives in the red House
- The Spanish has a dog

- In the green House, its owner drinks coffee.
- the Ukrainian drinks some tea
- the green house is immediately on the right of the white one
- the sculptor has some snails
- the diplomat lives in the yellow house
- in the middle house, its owner drinks milk
- the norvegian lives in the first house on the left
- the doctor lives in a house near the one of the fox owner
- In an house near the one of the diplomat there is an horse
- The violinist drinks some orange juice
- The Japanese is an Acrobat
- The Noregian lives near the blue house

and find who drinks water and who owns a Zebra.

# 5 Return instructions

The work can be done with teams up to 3 people. You will return your work on the moodle. You should return a pdf file with the questions answers and an explication of your implementations. You can add every pertinent remark.
The work should be return before the **Monday 7 November à 23h55**. Every day of delay will be penalized with a 10% penalization.

**marking scheme :**

| | |
|---|---|
| part 3 | 20pts |
| part 4.3 | 15pts |
| part 4.5 | 15pts |
| part 4.8 | 20pts |
| part 4.9 | 20pts |
| report : | 10pts |

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