

## Introduction à la programmation logique Le langage Prolog.

### 1. Origine :

Le langage Prolog a été conçu par Alain Colmerauer. C'est un langage de très haut niveau, dit de cinquième génération, il a été consacré en 1982, comme langage de support du projet japonais de 5<sup>ème</sup> génération

Prolog est un démonstrateur de théorèmes.

L'idée de base de la création de prolog :

- Mise en forme de la logique sous forme de clauses
- Introduction d'une règle d'inférence.

Les domaines d'applications de Prolog sont :

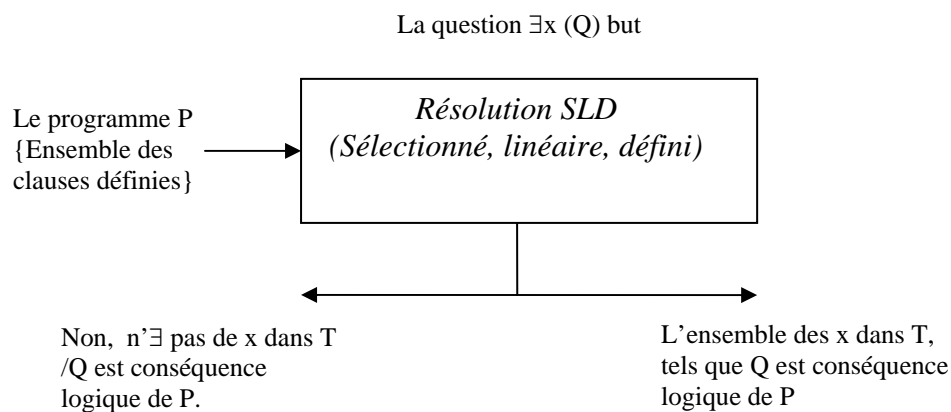
- ✓ automatisation des preuves de théorèmes,
- ✓ pour résoudre les problèmes d'analyse et de compréhension de la langue naturelle.
- ✓ Son application s'est étendue à l'intelligence artificielle,
- ✓ pour la représentation des connaissances et des systèmes experts.
- ✓ Prototypage, BD, BC etc ....

Prolog est un langage conversationnel, son code source est plus concis que Pascal, mais il consomme plus de temps et de mémoire.

Le langage Prolog est le plus souvent interprété, mais certaines versions Prolog disposent de compilateurs

### Programmer en Prolog consiste à [CM]:

- Déclarer des faits ou assertions sur des objets et leurs interactions
- Définir des règles sur des objets et leurs interactions
- Poser des questions (requêtes) sur des objets et leurs interactions.



*Schématique d'exécution Prolog*

## 2. Structure d'un programme Prolog :

Un programme  $\equiv$  des axiomes et des règles.

On se restreint aux sous-ensembles des clauses de Horn, définies dans Chap0. on n'admet que des formules du type :

$$\begin{aligned}
 & (lp1 \wedge lp2 \wedge lp3 \dots \wedge lpn) \rightarrow lp \\
 & \neg lp1 \wedge \neg lp2 \wedge \neg lp3 \dots \wedge \neg lpn \vee lp
 \end{aligned}$$

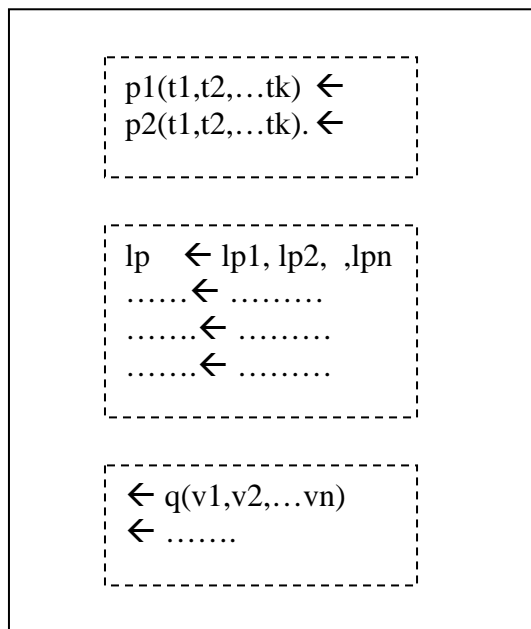
Qu'on écrit en syntaxe Prolog

$lp \leftarrow lp1, lp2, \dots, lpn.$

$lp :- lp1, lp2, \dots, lpn.$  Syntaxe d'une clause

Où  $lpi$  représente un littéral positif ou formule atomique,  $lpi = pi(t1, t2, \dots, tn)$   $pi$  : prédicat.

Autrement, une conjonction d'hypothèses entraîne une seule conclusion.



*Faits*

*Règles*

*But*

*En prolog ← s'écrit :-*

*Structure d'un programme Prolog*

### 2.1. Les faits (assertions) :

Un fait (formule atomique) est la représentation d'une connaissance exprimant une propriété d'objet ou une relation entre des objets.

*Exemple :*

filles(jamila) est la connaissance « jamila est une fille » qu'on écrit en Prolog

filles(jamila) ← ou bien filles(jamila).

pere(alii, jamila) représente la connaissance « alii est le père de jamila » en prolog

pere(alii,jamila) ← ou bien pere( alii, jamila).

mere(fatma, jamila)

Les prédicats sont pere , mere, fille et les termes sont jamila ali.

## 2.2. Les règles :

Les règles permettent de définir des relations à partir d'autres relations.

$lp \leftarrow lp1, lp2, \dots, lpn$  est une connaissance exprimée en logique mathématique par :

«  $\forall x1, x2, \dots, xn, (lp1 \text{ et } lp2 \text{ et } \dots \text{ et } lpn \Rightarrow lp)$  »

En prolog standard la règle s'écrit :

$$lp \quad :- \quad lp1, lp2, \dots, lpn$$

tête de clause        :-        corps de la clause

$lp1, lp2, \dots, lpn$  les prémisses

les quantificateurs universels sont omis, implicites, la conjonction « et » est remplacée par la virgule.

Exemple :

parent(X,Y) :- pere(X,Y).

grand\_parent( X,Z) :- parent(X,Y) , parent (Y,Z).

Soit la connaissance suivante :

« un parent d'une personne est son père ou sa mère »

Comment la représenter en logique mathématique ? Puis en prolog ?

$\forall x, \forall y (parent(x, y) \Leftrightarrow ((père(X,Y) \text{ ou } mere(X,Y) )$ ).

Equivalent à 2 énoncés :

- 1- si x parent de y, alors x est père de y ou x est mere de y.(non repr en prolog ,conclu contient ou)
- 2- si x est père de y ou x est mere de y alors x parent de y.(contient ou en prémisses)

En prolog 2- équivaut à:

parent(X,Y) :- pere(X,Y).

parent(X,Y) :- mere(X ,Y).

On peut ainsi définir de nouvelles règles telles que la règle « sœur » et enrichir ainsi la base des connaissances.

## 2.3. Les questions (requêtes) ou résolution de but :

Une fois que nous avons défini la base de connaissance, on peut lancer une requête

**q(v1,v2,...,vn).**

Les buts peuvent être simple ou composés (conjonctions d'atomes).

La requête provoque l'exécution du programme prolog qui va :

- Chercher un axiome qui convient
- Chercher si une règle peut être appliquée, c'est-à-dire si sa conclusion convient

(Chercher une assertion qui correspond à l'assertion de la question, deux assertions ont identiques si leurs prédicats sont identiques et si leurs arguments sont les mêmes dans l'ordre)

Si ça marche prolog répond **yes** sinon il répond **no**

Exemples de requête :

« Est-ce que ali est un parent de jamila » s'écrit en prolog par

?- parent(ali,jamila) → Yes

« Quels sont les parents de jamila ? »

?- parent(X, jamila) → X =ali → X= fatma

### 2.5. Conclusion :

Un programme logique est un ensemble (conjonction) fini de définition de prédicats.

## 3. Syntaxe des objets Prolog :

Les noms d'objets et de relations doivent suivre une syntaxe précise : en général un objet a une syntaxe alphanumérique, et il est recommandé d'utiliser des noms mnémoniques afin de reconnaître et nous rappeler le sens des objets. Ex pere au lieu de p.

- ✓ Les noms de relations et des objets doivent commencer par une lettre minuscule.
- ✓ Les faits doivent toujours se terminer par un point.
- ✓ La relation est écrite en premier, et les objets sont entre ( ).
- ✓ Les noms des variables doivent être une majuscule ou commencer par le blanc souligné « \_ » (tiret du 8) X \_som Flene ...
- ✓ les commentaires multilignes, comme en C, compris entre /\* et \*/ sur une seule ligne, introduits par le symbole %
- ✓

### 3.1. Un exemple de programme Prolog :

F1 mere(fatema, soraya).  
 F2 mere(fatema,jamila).  
 F3 mere(zohra, ali).  
 F4 pere(ali,jamila).  
 F5 pere(ali,kader).  
 F6 pere(hadj,ali).  
 F7 pere(houari,fatema).

R1 grandpere(X,Z) :- pere(X,Y), pere(Y,Z).  
 R2 grandpere(X,Z) :- pere(X,Y), mere(Y,Z).

R3 grandmere(X,Z) :- mere(X,Y), pere(Y,Z).  
 R4 grandmere(X,Z) :- mere(X,Y), mere(Y,Z).

.....  
 Déterminer les termes, prédicats,

On se propose de rajouter la nouvelle relation « frere » décrire la règle :

frere(X,Y) :- pere(Z,X),pere(Z,Y) , mere(Z,X), mere(Z,Y).

Les questions :

?-grandpere(hadj,kader). → yes

?-grandpere(X,jamila). → X=houari → X=hadj

?-frere(jamel,Y).

Autre règle : cousin

cousin(X,Y) :- pere(T,X), frere(T,U), pere(U,Y).

#### 4. Exécution d'un programme Prolog. Résolution de but.

Soit l'écriture généralisée d'un programme P :

$$p_1(t_1, t_2, \dots, t_k) .$$

$$p(t_1, t_2, \dots, t_k) :- p_1(t_1, t_2, \dots, t_k), p_2(t_1, t_2, \dots, t_k), \dots, p_n(t_1, t_2, \dots, t_k).$$

On exécute le programme P par la soumission d'une requête  $q(v_1, v_2, \dots, v_n)$  ;

La réponse de Prolog à la question  $q(v_1, v_2, \dots, v_n)$  est  $\rightarrow$  yes , s'il peut déduire que  $q(v_1, v_2, \dots, v_n)$  est vraie à partir de la connaissance du programme P constituée des faits et des règles.

$q(v_1, v_2, \dots, v_n)$  sera donc une conséquence logique de P.

Ex - ?grandpere(hadj,kader). Peut être déduit des faits F5 et F6 plus la règle R1.

**La résolution de but** est basée sur le **principe d'Unification**, qui fait appel à la **substitution**.

##### 4.1. Définition substitution S

C'est un ensemble fini de paires de la forme  $X_i = t_i$  où les  $X_i$  sont des variables et les  $t_i$  sont des objets, termes.

Par ex  $S = \{ X = ali, Y = fatema \}$  est une substitution.

La substitution vide (Identité) est  $\{ \}$  où l'ensemble des couples est vide (aucune variable à remplacer ou à instancier).

##### 4.2. Le principe d'Unification.

Consiste à mettre en correspondance  $q(v_1, v_2, \dots, v_n)$  et  $p(t_1, t_2, \dots, t_n)$  on unifie les deux termes sachant qu'un prédicat est identifié à la fois par son nom et par le nombre de ses arguments.

Le principe d'« unification » affecte une valeur à une variable.

Une fois qu'une valeur est affectée à une variable, on dit « variable liée ».

L'unification calcule une substitution S qui rend les termes égaux. On fait « une résolution d'équations symboliques »

$$q(v_1, v_2, \dots, v_n) = p(t_1, t_2, \dots, t_n)$$

Exemples d'unifications des deux termes :

$p(X, a, c) = p(a, a, Z)$	$\rightarrow$ substitution solution $S = \{ X=a, Z=c \}$
$f(X, g(a, d))$ et $f(h(c, Z), g(a, Y))$	$\rightarrow S = \{ X = h(c, Z), Y = d \}$
$f(g(X))$ et $f(h(Y))$ : $S = \phi$	
$pere(X, jamila)$ et $pere(ali, jamila)$	$\rightarrow S = \{ X = ali \}$

Conclusion	requête	solution
$c(X, Y)$	$c(f(a), g(a, b))$	$X = f(a), Y = g(a, b)$
$c(f(a), g(a, b))$	$c(X, Y)$	$X = f(a), Y = g(a, b)$
$f(T, c(e), d)$	$f(a, X, d)$	$T = a, X = c(e)$
$f(a, b, X)$	$f(Y, c, d)$	echec
$c(X, Y)$	$c(Z, T)$	$X = Z, Y = T$ (ou $T = Y$ , ou $Z = X \dots$ )
$c(X, a, b)$	$c(c, X, b)$	problème mal pose
$f(a)$	$f(a)$	oui
$p(X, c, X)$	$p(a, Y, a)$	$X = a, Y = c$
$f(X, g(X))$	$f(Z, Z)$	échec

### 4.3. Exemples de résolutions de buts.

Revenons au programme prolog famille.

Ex1 Soit la requête ?-grandpere(hadj,kader).

Comment procède Prolog pour la résolution de ce but ?

1. La requête correspond à la tête de la règle **grandpere** la var X sera instanciée à hadj et Z à kader
2. Prolog doit satisfaire les deux sous buts de la règle **grandpere**, qui sont **pere(X, Z)** et **pere(Z, Y)** avec les instanciations de X et Z égales à hadj et kader.
3. prolog recherche **pere(hadj, Z)** et **pere(Z, kader)** dans les faits, la variable Z sera instanciée à ali par conséquent :
4. Prolog répond → yes.
5. la substitution S est vide (Identité) car pas de variable à calculer.

Ex2 soit la requête ?-grandpere(X,jamila). → grandpere(hadj, jamila)  
grandpere(houari, jamila)

Prolog procède par les étapes suivantes :

1. cette requête correspond (sera unifiée) à la conclusion de la règle **grandpere**, la var X de la question est libre comme la variable X de la règle R1 grandpere → ces 2 variables sont unifiées.
2. on passe à la résolution des sous buts : **pere(X,Z)** et **pere(Z,jamila)** engendrés par la règle **grandpere** pour construire progressivement la substitution solution)
3. pour le premier sous but de la règle **pere(X,Z)** comme on ne connaît pas Z, le second sous but est **pere(Z,jamila)** et va correspondre à **pere(ali,jamila)** donc Z est instancié par ali
4. on revient au 1er but **pere(X, ali)** ce qui correspond à **pere(hadj, ali)** pour X= hadj.

Unifier ces deux sous termes revient à engendrer des substitution(s) solution(s) S qui les rendent égaux.

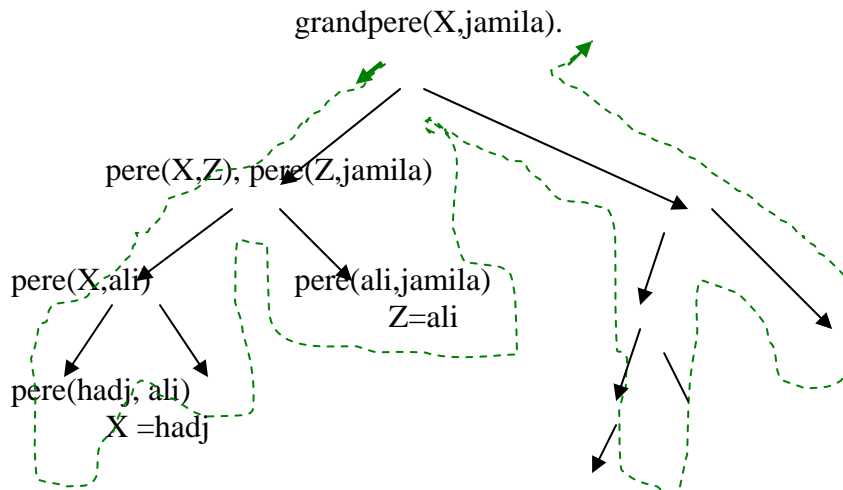
$S1 = \{X = hadj\}$

La résolution du but se poursuivra de la même manière avec la seconde règle R2 de **grandpere** qui permettra d'engendrer la substitution  $S2 = \{X = houari\}$

Le parcours de l'unification est arborescent, du fait qu'on part d'un but à résoudre pour remonter à des sous buts résolus. Ceci est la décomposition du problème en sous problèmes et ainsi de suite...

Prolog fonctionne en retour arrière (*backtracking*)! Un mécanisme qui permet de remonter dans l'arbre pour trouver d'autres solutions.

De ce fait il existe plusieurs algorithmes d'unifications.



*Mécanisme de backtracking.*

Autre exemple : représenter à l'aide de faits les connaissances suivantes :

6 est pair

6 est le double de 3

Le cours de PL a lieu le mardi de 13h à 14h30.

3.4. Exemple :

0 est un nombre naturel

Si N est un nombre naturel alors succ(N) est naturel

Natural(0).

Natural(S(N)) :- natural(N).

?- natural (s(s(s(0)))).

Yes

?-natural(mardi)

No

?-natural(X)

→ X=0

→ X= S(0)

→ X= S(S(0))

.....

## 5. Les outils :

Il existe différents outils pour vous permettre de programmer en Prolog :

- SWI Prolog  
*Possède un débogueur graphique ainsi que plusieurs solveurs de contraintes*  
[www.swi-prolog.org](http://www.swi-prolog.org)
- GNU Prolog  
*Propose un solveur de contraintes sur domaine fini*  
<http://gnu-prolog.inria.fr>

### 5.1. Installation

- Sous Windows :  
Télécharger l'un des programmes mentionnés précédemment et l'installer

### 5.2. Edition

Tout éditeur de texte : Bloc notes, Wordpad, ou Word l'extension d'un programme Prolog est : **.pro ou .pl**

#### **Lister le programme :**

Avec le prédicat **listing**.

Avec argument : permet d'éditer la liste de définition du prédicat argument

Sans arguments : permet de lister tout le programme tout le programme.

```
3 ?- listing('pere').
```

```
pere(ali, jamila).
```

```
pere(ali, med).
```

```
pere(med, sam).
```

```
true.
```

**Editer** permet d'ouvrir la fenêtre d'édition de l'interpréteur Prolog.

```
?- edit(famille).
```

```
true.
```

### 5.3. Exécution.

Lancez votre interpréteur Prolog :

Chargez votre programme grâce à la commande *consult* :

Utilisez le prédicat *consult* avec comme argument le chemin d'accès du programme, sous forme d'une chaîne :

```
| ?- consult('le_nom_de_votre_programme.pro').
```

Tous les prédicats sont alors évalués et on peut alors les exécuter

On peut visualiser le contenu suivant :

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.62)
```

```
Copyright (c) 1990-2008 University of Amsterdam.
```

```
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,  
and you are welcome to redistribute it under certain conditions.
```

```
Please visit http://www.swi-prolog.org for details.
```

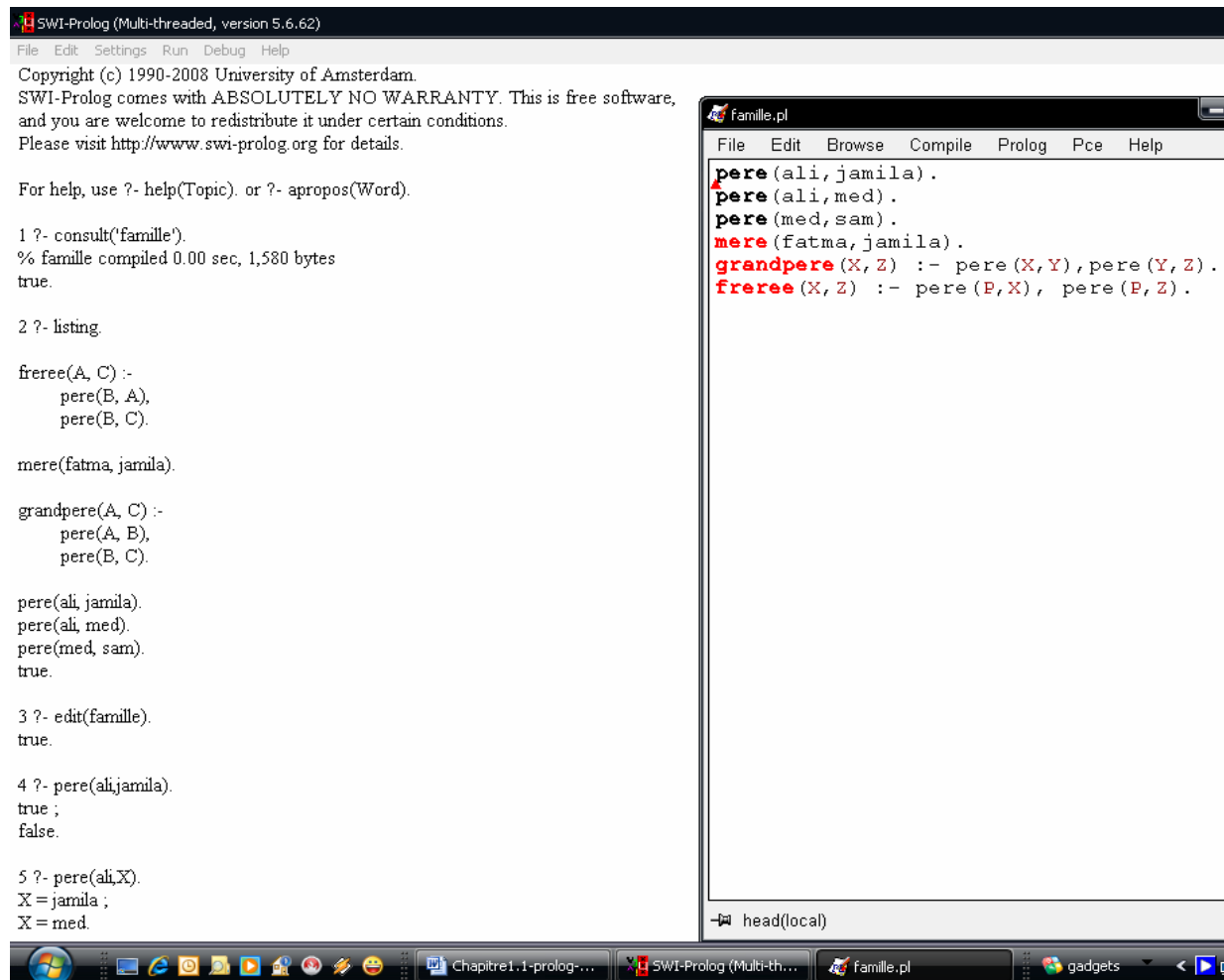
```
For help, use ?- help(Topic). or ?- apropos(Word).
```

1 ?- consult('famille.pl').  
 % famille.pl compiled 0.00 sec, 1,484 bytes  
 true.

2 ?- pere(ali,X).  
 X = jamila ;

Pour passer à la solution suivante tapez la touche « ; »

X = med.



Fenêtre d'exécution et d'édition SWI-Prolog

## Références :

Programmer en Prolog. Jonhatan Elbaz Ellipses.  
 Programmer en Prolog. W.F. Clocksin C.S. Mellish. Eyrolles.  
<http://prolog.developpez.com/cours/>