INTRODUCTION AU PROLOG

1- Présentation

11) Le langage Prolog est un langage de **programmation déclaratif** en opposition aux langages de **programmations procéduraux** (pascal, c++).

Programmation procédurale *: série d'instruction qui indiquent à la machine comment elle doit procéder pour résoudre un problème, en fait le programmeur raisonne toujours en termes d'opérations à effectuer.*

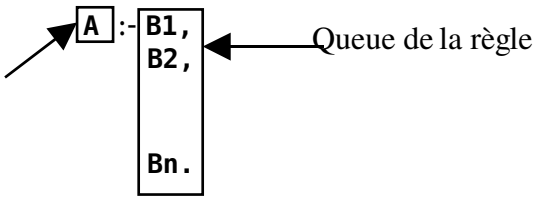
Programmation déclarative *: Le programmeur ne se préoccupe plus exclusivement de la marche à suivre pour résoudre le problème. Il se préoccupe surtout de définir les objets manipulés par son programme et les relations qui existent entre ces objets. Ce type de langage passe en revue toutes les solutions du problème.*

12) Prolog est un langage de programmation logique.
Son mécanisme de résolution est l'unification.

2- Syntaxe du Prolog

21) Structure d'un programme Prolog

Un programme Prolog est constitué de clauses de la forme :



Cette syntaxe signifie qu'on a **A si on a B1 et B2 et … et Bn**.

Définitions :

- **règles** : *les règles permettent de définir les prédicats.*
- **conclusion** : *C'est la tête d'une règle.*
- **prémisse** : *C'est la queue d'une règle.*

22) La syntaxe des prédicats

Un prédicat est défini par son nom et son nombre d'arguments :

```
nom_du_prédicat(arg1,…,argn)
```

La valeur du prédicat est **vrai** ou **faux**.
Les arguments peuvent être des variables, des constantes, des prédicats, ou d'autres termes.

Définitions :

- **constantes** : *représente un objet, une valeur.*
- **variables** : *une variable est une inconnue comme dans une formule mathématique dont le système Prolog est chargé de déterminer la ou les valeurs. Elles commencent par une majuscule ou un underscore.*
- **prédicats** : *un prédicat spécifie et représente une relation entre arguments. Ils sont définis par un nom commençant par une lettre minuscule. Toutes les clauses qui commencent par le même prédicat sont mises ensemble.*
- **termes** : *les constantes et les variables sont des termes, l'application d'une fonction à un terme est un terme.*
- **variables muettes** : *c'est une variable sans affectation.*
- **littéraux** : *un littéral est l'application d'un prédicat aux arguments qui sont des termes.*
- *Les **chaînes de caractères** sont entre quotes*
- *Les **commentaires** sont précédés de %.*
- *Prolog distingue les majuscules des minuscules.*

Exemples : **`fille(X,Y)`** X est la fille de Y.
`fille(julie, jean)` julie est la fille de jean.

Une clause sans prémisse (de la forme A.) est toujours vraie :
`fille(julie, jean).`

Les buts ou questions sont toujours de la forme B1,B2,…,Bn.

3- Utilisation de GNU-Prolog

Pour lancer le prolog : **`gprolog`**
Vous êtes alors avec le prompt **?-** derrière lequel vous tapez les buts et questions.
Pour charger un programme : **`[nom_prog].`**
Ensuite lancer le but ou la question.
On peut avoir la trace d'un prédicat en lançant le mode déboggage: **`trace.`**
Pour sortir du mode déboggage: **`notrace.`**

4- Premier exercice :généalogie miniature

Ecrire en Prolog les faits suivants :

Jean est le père de Pierre et Jacques.
Pierre est le père de Paul.
Jacques est le père de Rémi.

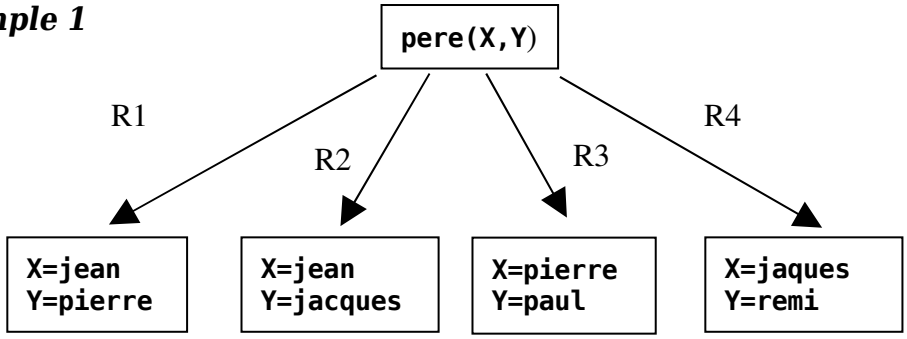
Définir le prédicat grand-père.

solution :

Dans le cas où l'on utilise la règle qu'avec le grand-père paternel, on peut simplifier et ne garder que la règle :

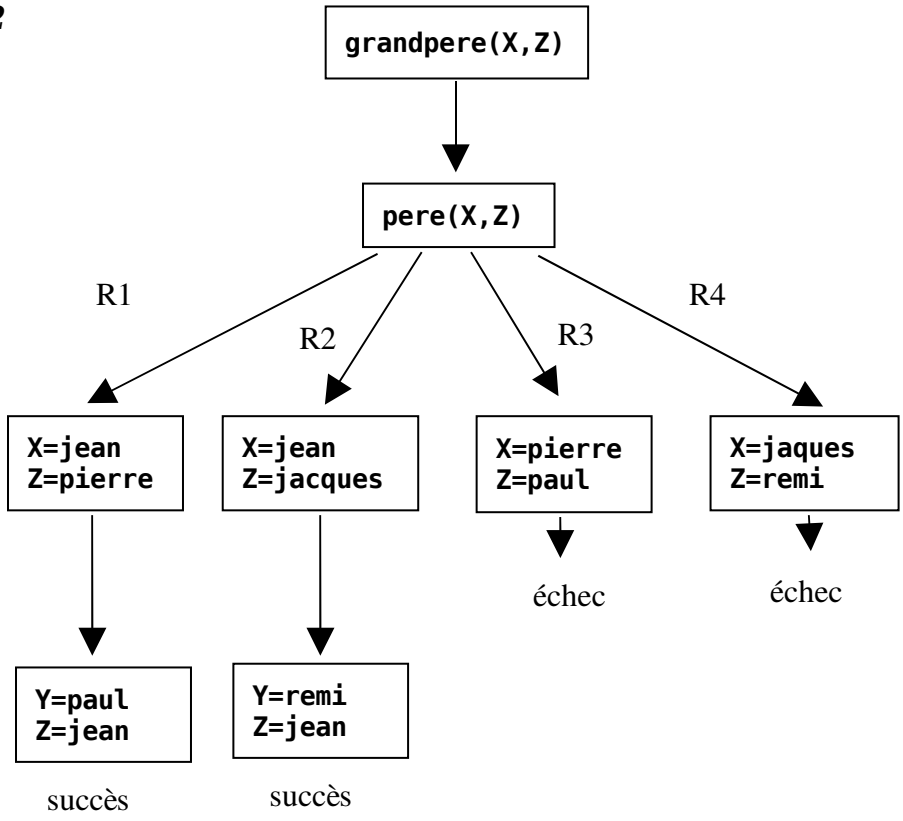
Tester les prédicats père et grand-père (**`pere(X,Y).`** ou **`pere(Rémi,X).`** …).
Les mécanismes de résolution mis en œuvre lorsqu'on donne un but à Prolog sont (stratégie Prolog) :

exemple 1



Prolog veut effacer le but **`pere(X,Y)`**. Pour cela, il cherche toutes les règles dont la tête s'unifie avec le but et les règles **`R1,…,Rn`**, il prend d'abord la première règle, il cherche à unifier les 2 prédicats en unifiant des variables avec des variables ou des constantes.
Dans notre cas, il va unifier pour la première règle **`X=jean`** et **`Y=pierre`** et remonte dans l'arbre afin de connaître toutes les solutions.

exemple 2



Mécanisme général de Prolog lorsqu'il a un but à vérifier :

Dans notre cas, le but à vérifier est **`grandpere(X,Y)`**. Prolog va donc chercher les règles qui peuvent s'unifier à ce but, seule la règle **`R5`** correspond à ce cas. Il essaie ensuite d'unifier les deux prédicats en unifiant éventuellement des variables avec des variables ou des constantes.

Si la clause utilisée est un fait et si le but s'unifie avec le but s'unifie avec le fait alors le but s'efface, c'était le cas pour **`pere(X,Y)`**.

Ici le but s'unifie avec la règle **`R5`**, Prolog essaie donc de vérifier ce nouveau but, il recherche les règles pouvant s'unifier avec le premier but **`pere(X,Z`**), il y a quatre règles, il les vérifie une à une.

Une fois arrivé à un but vide, il remonte et essaie les autres possibilités et ainsi de suite jusqu'à n'avoir ni but ni autre clause possible.

5- Deuxième exercice : âge

Coder les faits suivants :

Paul a 20 ans.
Pierre a 30 ans.
Julie a 15 ans.

Définir un prédicat plus_vieux à deux arguments et qui donne **vrai** si la première personne est plus vieille que la seconde.

solution :

6- Troisième exercice : factoriel

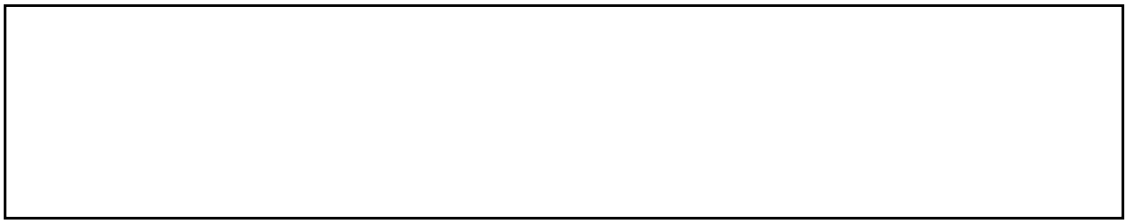
Récursivement on peut définir factoriel comme suit :
fact(0)=1
*fact(n)=n*fact(n-1)*

Ainsi le prédicat fact(N,F) sera vrai si F=N*R avec fact(N-1,R) vrai.
Coder factoriel de n :
solution :

7- Quatrième exercice : négation et différence

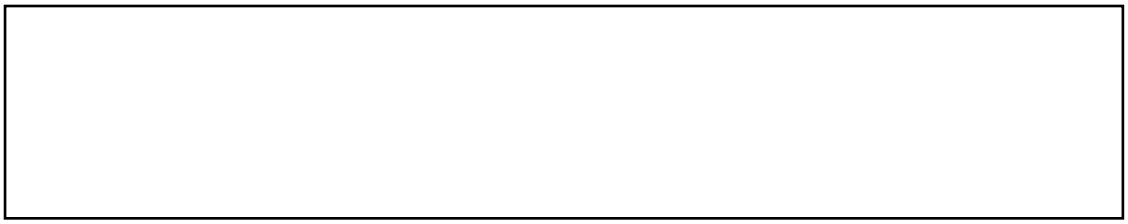
Redéfinir le prédicat non à un argument qui prend la valeur de vérité « opposée » à celle de son argument.

solution :



Ecrire en Prolog le prédicat **`diff(arg1,arg2)`** qui renvoie **vrai** si et seulement si **`arg1`** est différent de **`arg2`**.

solution :



8- Les listes

Prolog utilise une structure particulière de données : les listes.

Une liste est une structure linéaire entre crochets [ ] dont le premier argument est la tête de la liste et le reste la queue de la liste, chaque élément étant séparé par une virgule :

exemples :

```
[1,2]
[ ]
[2|[3,4,5]]
[X|Y]
```

exemples de prédicats sur les listes :

Test si l'argument est une liste :

```
liste([]).
liste([X|Y]) :- liste(Y).
```

Sous-liste de tête :

```
listeT([],L).
listeT([D|X],[D|Y]) :- listeT(X,Y).
```

Sous-liste de queue :

```
listeQ(L,L).
listeQ(L,[X|Y]) :- listeQ(L,Y).
```

9- Cinquième exercice : appartenance à une liste

Ecrire un prédicat qui teste si un élément appartient à une liste ou non.

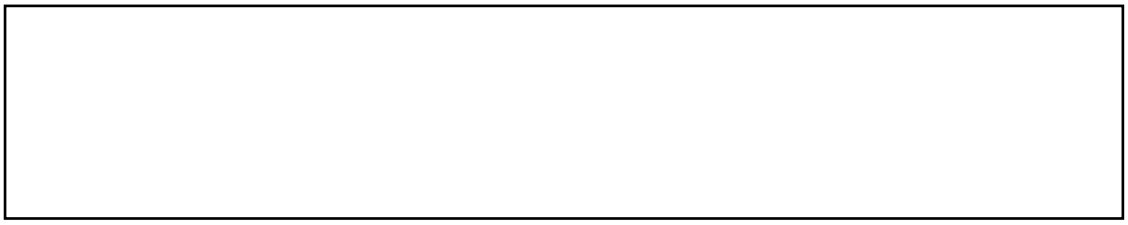
solution :

Que se passe-t-il si on ajout un **cut** ?

10 - Sixième exercice: Concaténation de deux listes

Donner les règles Prolog effectuant la concaténation de deux listes.

solution :



exemples :

```
      concat([1,2,3],[4,5],X).
            { X=[1,2,3,4 ,5] }

      concat(L,[4,5],[1,2,3,4,5]).
            { L=[1,2] }

      concat(X,Y,[1,2,3]).
      { (X=[ ] , Y=[1,2,3]) , (X=[1] , Y=[2,3]) , (X=[1,2] , Y=[3]) ,
(X=[1,2,3] , Y=[ ]) }

      concat(X,[1,3],Y).
```

Boucle infinie : nombre de possibilités infini

11 - Septième exercice : Longueur d’une liste

Définir un prédicat à deux arguments, le premier reçoit une liste et le second retourne la longueur de cette liste.

solution :

12 - Huitième exercice : i^{ème} élément d’une liste

Définir un prédicat à trois arguments, le premier reçoit une liste ,le deuxième reçoit un indice (compris entre 1 et la dimension de la liste) et le troisième retourne l’élément de la liste correspondant à l’indice donné.

solution :