



Ministère de l'enseignement supérieur et de la recherche scientifique  
Université Djillali BOUNAAMA - Khemis Miliana (UDBKM)  
Faculté des Sciences et de la Technologie  
Département de Mathématiques et d'Informatique



## Chapitre 5

# Programmation Logique

MI-GLSD-M1-UEM213 : Paradigmes de langages de Programmation

**Noureddine AZZOUZA**

n.azzouza@univ-dbkm.dz

# Plan du Cours

**1. Introduction et Historique**

**3. Concepts principaux**

**3. Langage PROLOG**

# Introduction

# & Historique

```
class JavaProgram
{
    Pen ps_Dlugopis = new Pen(this.PS_Kolor, this.
    ps_Dlugopis.DashStyle = this.PS_RodzajLinii;
    public static void main(String[] args) throws java.lang.Exception
    {
        hplanszaGraf.DrawRectangle(ps_Dlugopis, PS_X -
        PS_Margines, PS_X - PS_Margines, PS_Y / 2, PS_Y
        public static void main(String[] args)
        {
            ps_Dlugopis.Dispose();
            BufferedReader file_reader = new BufferedReader(new InputStreamReader
            String text;
            while (!file_reader.isClosed())
            {
                text = file_reader.readLine();
                System.out.println(text);
            }
            for (int i=0; i<PS_Widoczny)
            {
                for (int j=0; j<PS_Grubosc)
                {
                    Pen ps_j_Dlugopis = new Pen(hForm1.PS_img.Back
                    this.PS_grubosc);
                    ps_Dlugopis.DashStyle = this.PS_RodzajLinii;
                    hplanszaGraf.DrawRectangle(
                    PS_Margines,
                }
            }
        }
    }
}
```

# Paradigme Logique

- ✓ La **programmation logique** est basée sur l'idée qu'un programme implémente une relation et non un mapping (correspondance).
- ✓ Les langages de programmation logique permettent de décrire le (les) résultat (s) en écrivant **la relation** qui le lie avec les données du problème.
- ✓ Ces langages ont permis d'exprimer et de résoudre des problèmes d'optimisation s'exprimant plus simplement par les contraintes que doit respecter la solution que par la méthode pour y parvenir.



## Remarques

- ✓ Bien que basé sur la logique mathématique, aucun langage de programmation logique ne peut exploiter toute la puissance de la logique mathématique.
- ✓ Les langages de programmation logiques sont restreints aux clauses de Horn
- ✓ La logique n'est pas suffisante pour rendre compte de toutes les notions dont on a besoin pour un langage de programmation



## Les relations

- ✓ Une relation binaire **R** entre deux ensembles **S** et **T** est incluse dans **SxT** telle que :  $\forall x \in S \text{ et } \forall y \in T$ 
  - ❖  $R(x, y)$  est soit **vraie** soit **fausse**.
  - ❖  $R(x, y)$  peut être vue comme un **prédicat**.
- ✓ Un **programme logique** est une implémentation de la **relation logique**. Ce programme permet par exemple de répondre aux **requêtes** suivantes :
  - ❖ étant donnée  $a$  et  $b$ ,  $R(a, b)$  est elle vraie ?
  - ❖ étant donnée  $a$ , pour quelles valeurs de  $y$ ,  $R(a, y)$  est elle vraie ?
- ✓ La relation peut être : **unaire, binaire, ternaire**.
- ✓ Les réponses sont variées : **vrai / faux / plusieurs réponses / pas de réponse**



## Historique

- ✓ **PROLOG** a été créé par A-CAULMERAUER et P.ROOSSEL à l'université de MARSEILLE et reconnu comme langage de développement en I.A en 1980.
- ✓ Plusieurs versions et extensions vers la programmation par contraintes ont été développées : **PROLOG** 1, 2, 3, **CPROLOG**, **visual PROLOG**, **EPILOG**, **Quintus turboPROLOG**, **XILOG**, **GOIDEL**, **OZ** ...etc.
- ✓ **PROLOG** a servi en particulier comme outil pratique de développement de logiciel.



# Concepts

# Principaux





# Les Faits

- ✓ Les faits : sont des affirmations qui décrivent des relations ou des propriétés.
- ✓ **Exemple**
  - masculin (ALI)
  - Féminin (SAMIA)
  - Père (ALI, SAMIA)  $\Rightarrow$  ALI est le père de SAMIA
  - Mère (FATIMA, SAMIA)  $\Rightarrow$  FATIMA est mère de SAMIA.
- ✓ - La forme générale d'un fait est : **Prédicat** (argument 1, argument 2 ...)



# Prédicat

Un prédicat est une fonction logique s'évaluant à **VRAI** ou **FAUX**, c'est un symbole qui traduit une relation.

## Arité

L'arité est le nombre des arguments du fait.

- ✓ Un **prédicat** est indiqué par son nom et son arité : **Prédicat / arité**.
  - ❑ **Exemple** : Mère / 2 de manière générale
- ✓ Un prédicat **unaire** donne propriété de l'argument.
  - ❑ **Exemple** : Le ciel est bleu : BLEU (ciel)
- ✓ Un prédicat **binaire** exprime une relation entre 2 objets.
  - ❑ **Exemple** : ALI est le père d'IBRAHIM  $\Rightarrow$  père (ALI , IBRAHIM )
- ✓ Un prédicat **d'arité supérieur à 2** établit un lien entre ses arguments.
  - ❑ **Exemple** : Auteur ( BENNABI, 1905, 1973)



# Requêtes ou Questions

✓ En mode interactif, on peut poser des questions sur les faits.

❑ **Exemple** ? - masculin (OMAR) ==> YES

? - masculin (LILI) ==> NO (il n'existe pas dans la base des connaissances)

✓ On peut aussi utiliser des variables :

❑ **Exemple** ? - masculin (x)

X = OMAR ;

X = ALI ;

✓ Le caractère « ; » permet de demander la solution suivante.

✓ Le caractère « » arrête la recherche de la solution.

✓ Les constantes commencent par une **minuscule**, les variables commencent par une **majuscule**.

# Les Atomes

- ✓ Les **atomes** : sont des chaînes alphanumériques qui
  - ❑ commencent par une lettre minuscule : omar, samia, list 1, n2.
  - ❑ Les chaînes entre ' ' ou « »
  - ❑ Les nombres 19, -25, 314, 23, -5.
- ✓ Les **variables** commencent par une majuscule X, XYZ, Xyz
  - ❑ \_X\_3\_ (ces variables sont anonymes).
- ✓ Certains PROLOG introduisent le concept de **domaine** pour distinguer les symboles (variables), les entiers (long...), les réels...etc.



# Unification

- ✓ Une formule est une application d'un prédicat à ses arguments.
- ✓ **L'unification** est le procédé par lequel on essaie de rendre 2 formules identiques en donnant des valeurs qu'elles contiennent.
- ✓ Une unification peut réussir ou échouer
- ✓ **Exemple** : père (x, x) et père (Omar, Ali) ne peuvent pas s'unifier.

? – père (x, y)

X = Omar Y = Ali.



# Substitution

- ✓ PROLOG unifie le terme de la requête au terme qui convient dans la base de données, pour ceci il réalise la **substitution** des variables  $x$  et  $y$  par des termes, ici des constantes.
- ✓ On note cette substitution  $\{x = \text{Omar}, y = \text{Ali}\}$
- ✓ On dit qu'un terme  $A$  est une **instance** de  $B$  s'il existe une substitution de  $A$  à  $B$ .
- ✓ **Exemple** : masculin (Omar) et masculin (Ali) sont des instances de masculin ( $x$ )
  - $\{x = \text{Omar}\}, \{x = \text{Ali}\}$  sont des substitutions correspondantes.



# Variables partagées

- ✓ Une même variable peut être utilisée pour contraindre deux arguments à avoir la même valeur.
- ✓ **Exemple** pour chercher un père qui a le même nom que son fils
  - ? – père (x, x)
- ✓ les questions peuvent être des conjonctions et on peut partager des variables entre les buts.
- ✓ **Exemple** pour chercher tous les fils d'un père, on partage la variable x entre père et masculin
  - ? – père (x, y) masculin (x).



# Clauses ou Règles

- ✓ Les règles permettent d'exprimer des **conjonctions de buts**.
- ✓ Leur forme générale est : <Tête> : -- C1, C2, ....Cn.
  - :-- signifie « **si** » ou « **if** »
  - , signifie « **et** »
  - ; signifie « **ou** »
- ✓ La Tête de la règle est **vraie** si **tous** les éléments de la règle C1, C2, ....Cn est **vrai**.
  - ❑ On appelle ce type de règle **CLAUSE DE HORN**.
  - ❑ Les éléments du corps sont aussi appelés des sous buts car pour démontrer la tête de la règle, il faut démontrer tous ses sous buts.



## Clauses ou Règles

✓ **Exemple** : si x est un homme alors il est fort

- ❑  $FORT(x) :- homme(x).$
- ❑  $Grand\ père(x, y) :- père(x, z) père(z, y).$

✓ **Arbre généalogique**

- ❑  $Fils(A, B) :- père(B, A), masculin(A).$
- ❑  $Fils(A, B) :- mère(B, A), féminin(B).$
- ❑  $Parent(x, y) :- père(x, y).$
- ❑  $Parent(x, y) :- mère(x, y).$
- ❑  $Grand\ père(x, y) :- parent(x, z), parent(z, y).$



## Remarque

- ✓ les **faits** sont une forme particulière de **règles** qui sont toujours **vraies**.
- ✓ Un fait est une clause dont la queue est vide.
- ✓ Une requête est une clause sans tête **? - <corps>** Signifiant : déterminer si « corps » est vraie.
- ✓ Les règles peuvent aussi être récursives.

### ❑ Exemple

ancêtre (x, y) : -- parent (x, y).

ancêtre (x, y) : -- parent (x, z), ancêtre (z, y).



# PROLOG

```
class JavaProgram
{
    Pen ps_Dlugopis = new Pen(this.PS_Kolor, this.
    ps_Dlugopis.DashStyle = this.PS_RodzajLinii;
    public static void main(String[] args) throws java.lang.Exception
    {
        hplanszaGraf.DrawRectangle(ps_Dlugopis, PS_X -
        PS_Margines, PS_X - PS_Margines, PS_Y / 2, PS_Y
        public static void main(String[] args)
        {
            ps_Dlugopis.Dispose();
            BufferedReader file_reader = new BufferedReader (new InputStreamRe
            String text;
            while (!text.equals(""));
            in = new BufferedReader (new InputStreamReader(System.in));
            for (int i=0; i<PS_Widoczny; i++)
            {
                for (int j=0; j<PS_Widoczny; j++)
                {
                    Pen ps_j_Dlugopis = new Pen(hForm1.PS_img.Back
                    z[a+j]=x[j];
                    this.PS_grubosc);
                    ps_Dlugopis.DashStyle = this.PS_RodzajLinii;
                    hplanszaGraf.DrawRectangle(
                    PS_Margines
                }
            }
        }
    }
}
```

# Introduction

- ✓ **PROLOG** : Créé vers les années 1970.
- ✓ **Utilisé pour**
  - ❑ l'interrogation de base de données
  - ❑ Réalisation de système expert
  - ❑ Compréhension du langage naturel
- ✓ **Particularité :**
  - ❑ Aucune distinction entre programme et données
  - ❑ aucune structure de contrôle.



# Principe

- ✓ Prolog implémente la règle de résolution en utilisant la technique du **Backtracking** (recherche en profondeur et retour arrière)
- ✓ Exécuter un programme revient à demander la preuve d'une expression.
- ✓ En programmation logique, nous pouvons trouver tous les objets en relation avec d'autres objets. ( propriété de l' "invertibility")



# Les types

- ✓ **Standard** : integer, real, char, symbol
- ✓ **construit** ( voir les structures)

# Les variables et les constantes

- ✓ Les **variables** sont des combinaisons de lettres, chiffres et `_`. Le premier caractère est une lettre alphabétique majuscule ou `'_'`.
- ✓ Les **constantes** peuvent être des nombres, symbole (chaîne de caractères dont la première lettre est en minuscule, ou une chaîne de caractères quelconque entre `"`.



# Les Listes

- ✓ Une liste L dont les éléments sont de type type est définie :  $L \text{ type}^*$  (Domains).
  - [] désigne une liste vide.
  - [a, b, c] liste formée des 3 éléments a, b et c.
  - [X!Y] X désigne le car et Y le cdr.
  - $[a, b, c] = [a![b, c]] = [a, b![c]] = [a, b, c![]]$

# Les Structures

- ✓ Utilise le mot clé **Domains**
- ✓ **Exemple** : `typedate = date(integer, integer, integer)`



## Portée des variables

- ✓ Les constantes ont une portée globale.

## Portée des constantes

- ✓ La portée d'une variable se limite à une clause.





# Structure d'un programme

## ***Domains***

Construction de nouveau types

## ***Predicates***

Définitions des prédicats (types des objets en relation)

## ***Clauses***

Faits et règles

## ***Goal***

But



## Exemples

### *Predicates*

*Parent(symbol, symbol)*

*Frere(symbol, symbol)*

*Ascendant(symbol, symbol)*

### *Clauses*

*Parent(aa, bb).*

*Parent(bb, cc) .*

*Parent(bb, dd).*

*Parent(dd, ee).*

*Parent(ff, dd).*



## Exemples

```
Frere(X, Y) :- Parent(Z, X), Parent(Z, Y), Not(X=Y).  
Ascendant(X, Y) :- Parent(X, Y).  
Ascendant(X, Y) :- Parent(X, Z), Ascendant(Z, Y).
```

```
/* On peut aussi écrire ascendant comme suit :  
Ascendant(X, Y) :- Parent(X, Y);  
Parent(X, Z), Ascendant(Z, Y).  
*/
```



# Exemples

- *Question : frere(dd, cc)*  
✓ *Réponse Yes*
- *Question : frere(X, dd)*  
✓ *Réponse X = cc*
- *Question : frere(dd, X)*  
✓ *Réponse X = cc*



## Exemples

➤ *Question : Ascendant( $X$ , ee)*

✓ *Réponse*

$X = dd$

$X = aa$

$X = bb$

$X = ff$

➤ *Question : Ascendant( $X$ , cc)*

✓ *Réponse*

$X = bb$

$X = aa$

