

---

# Outils pour la programmation logique par contraintes

Cours commun Master ILI & SIA

**Lakhdar SAÏS**

[Sais@cril.fr](mailto:Sais@cril.fr)

<http://www.cril.fr/~sais/>

2009-2010

# Plan du cours

- **Programmation logique et Prolog (PL)**
  - SWI-Prolog, Sicstus
- **Programmation logique par contraintes (PLC)**
  - Sicstus
- **Problèmes de satisfaction de contraintes (CSP/PC)**
  - Choco

# PL : plan du cours

- Les origines de Prolog
- Deux exemples de programmes
- Syntaxe
  - Constantes, variables, termes, prédicats.
  - Assertions, Règles, Buts.
- Sémantique
  - Unification
  - Arbres de dérivation
- Le langage

# PROGRAMMER EN LOGIQUE

Les origines :

- 1972 : création de Prolog par A. Colmerauer et P. Roussel à Marseille.
- 1980 : reconnaissance de Prolog comme langage de développement en Intelligence Artificielle
- Depuis plusieurs versions, dont une tournée vers la programmation par contraintes.
- Depuis, un grand nombre d'implémentations. Voir par exemple:

<http://fr.wikipedia.org/wiki/Prolog>

La version que nous allons utiliser: SWI-Prolog, Sicstus3

# 1er exemple : « biographie »

Un programme prolog: **assertions** et **règles**

```
/*bio(nom,sexe,ne en,dcd en,pere,mere)*/  
bio(louis13,h,1601,1643,henri4,marie_medicis).  
bio(elisabeth_france,f,1603,1644,henri4,marie_medicis).  
bio(louis14,h,1638,1715,louis13,anne_autriche).
```

```
/*pere(pere,enfant)*/  
pere(X,Y):- bio(Y, , , ,X, ).  
/*age(personne,age)*/  
age(X,Y):-bio(X, ,Z,T, , ),Y is T-Z.
```

# 1er exemple : « biographie »

On interroge le programme: **but**s

Quel est la date de naissance de Louis XIV?

**bio(louis14, \_ ,X,\_, \_ , \_).**

Qui est le père de Louis XIII?

**pere(X,louis13)....**

mais aussi...

**bio(louis13, \_ , \_ , \_ ,X, \_).**

Combien d'année Louis XIV a survécu à son père?

**bio(louis14, \_ , \_ ,Y,Z, \_ ), bio(Z, \_ , \_ ,T, \_ , \_ ), R is Y-T.**

Parmi les personnes dont on dispose de la « biographie »,  
qui sont les hommes?

**bio(X,h, \_ , \_ , \_ , \_).**

# 1er exemple : « biographie »

```
/* bio.pl */
```

```
bio(louis13,h,1601,1643,henri4,marie_medicis).
```

```
bio(elisabeth france,f,1603,1644,henri4,marie_medicis).
```

```
bio(louis14,h,1638,1715,louis13,anne_autriche).
```

```
pere(X,Y):-bio(Y,_,_,_,X,_).
```

```
age(X,Y):-bio(X,_,T,Z,_,_), Y is Z-T.
```

```
# sicstus
```

```
?- consult(bio).
```

```
yes
```

```
?-bio(louis14,_,X,_,_,_).
```

```
X = 1638 ?;
```

```
no
```

# 1er exemple : « biographie »

?- pere(X,louis13).

X = henry4 ?;

no

?-bio(louis13,\_,\_,\_,X,\_).

X=henry4? ;

no

?- bio(louis14,\_,\_,Y,Z,\_),bio(Z,\_,\_,T,\_,\_),R is Y-T.

R = 72, T = 1643, Y = 1715, Z = louis13 ?;

no

# 1er exemple : « biographie »

?- bio(X,h,\_,\_,\_,\_,\_).

X = louis13 ?;

X = louis14 ?;

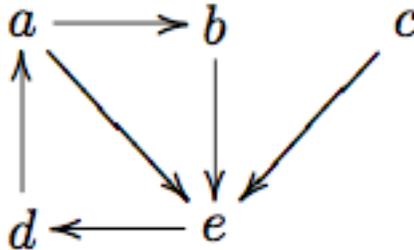
no

?- halt.

[ Prolog execution halted ]

#

## 2ème exemple : modélisation d'un graphe orienté



```
/* arc(source,destination) */
```

```
arc(a,b). arc(a,e). arc(b,e). arc(c,e). arc(e,d). arc(d,a).
```

```
/* connexe(source,destination) */
```

```
connexe(X,Y) :- arc(X,Y).
```

```
connexe(X,Y):- arc(X,Z),connexe(Z,Y).
```

## 2ème exemple : modélisation d'un graphe orienté

?- `consult(graph).`

yes

?- `connexe(a,b).`

yes

?- `arc(a,X).`

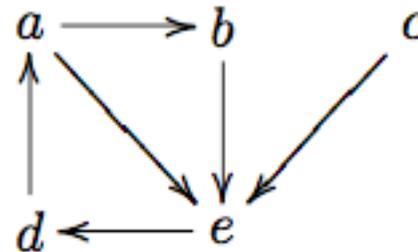
X=b? ;

X=e? ;

no

?- `connexe(a,c).`

boucle!



# Prolog : syntaxe

Les éléments de base d'un programme Prolog sont les prédicats et les termes.

Dans

***connexe(a,X).***

***connexe*** est un (symbole de) prédicat.

***a*** est un terme (une constante).

***X*** est un terme (une variable).

Il n'y a pas de différence syntaxique entre prédicats et termes: les termes aussi peuvent avoir des arguments:

***membre(X,node(X,\_,\_)).***

ici ***node(X,\_,\_)*** est un terme à trois arguments (un arbre binaire étiqueté, par exemple).

# Prolog : syntaxe

le lexique (simplifié)

**Constantes** : chaînes de caractères dont le premier est minuscule

**variables** : chaînes de caractères dont le premier est majuscule ou \_

**nombres** : en notation décimale

**les ponctuations** : :- , . ( ) ; ...

# Prolog : syntaxe

la grammaire (simplifiée)

programme-prolog ::= clause { clause }

clause ::= assertion | regle

assertion ::= predicat.

predicat ::= constante [ ( liste-termes ) ]

regle ::= predicat :- corps.

corps ::= predicat { , predicat }

liste-termes ::= terme { , terme }

terme ::= terme-simple | terme-complexe

terme-simple ::= constante | variable | nombre

terme-complexe ::= constante ( liste-termes )

# Prolog : sémantique

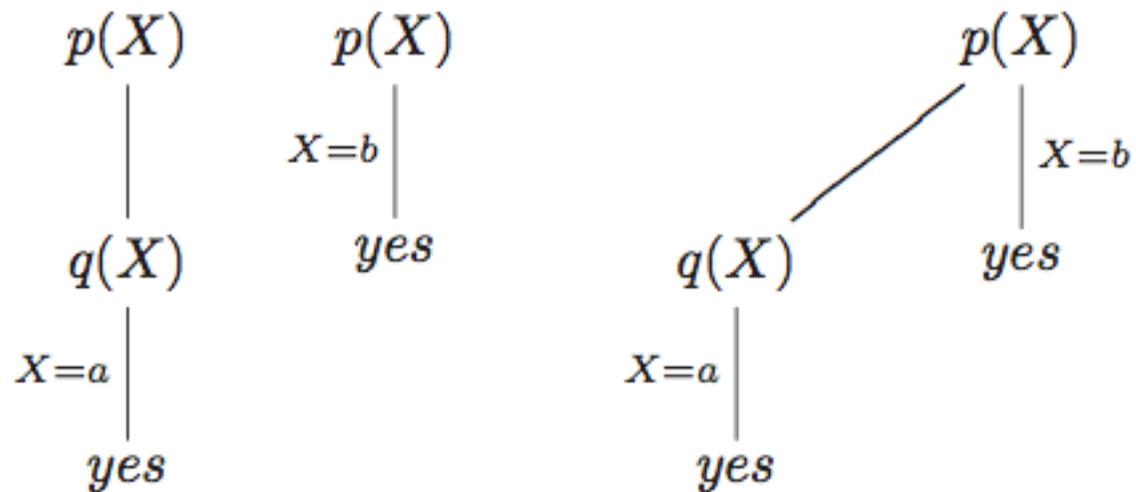
Sémantique opérationnelle: un exemple

$q(a)$ .

$p(X):-q(X)$ .

$p(b)$ .

but:  $p(X)$



Arbre de dérivation de  $p(X)$

# Prolog : sémantique

Sémantique opérationnelle: l'unification

Une **substitution** est une fonction partielle qui associe des termes à des variables (qu'on peut noter comme un ensemble de couples (Var, terme). Par exemple  $\sigma_0 = \{ (X, \text{zero}), (Y, \text{succ}(T)) \}$  est une substitution).

**L'application d'une substitution  $\sigma$  à un terme  $t$** , notée  $t\sigma$ , est le terme  $t$  dans lequel les variables de  $\sigma$  sont remplacées par les termes correspondants.

Par exemple  $\text{add}(X, Y)\sigma_0 = \text{add}(\text{zero}, \text{succ}(T))$

Deux termes  $t$  et  $s$  sont **unifiés** par la substitution  $\sigma$  si  $t\sigma = s\sigma$  (c.à.d. si les deux termes résultants de la substitution sont identiques).

La substitution  $\sigma$  est **l'unificateur le plus général (mgu)** de  $t$  et  $s$  S'il unifie  $t$  et  $s$  et si tout autre unificateur de  $t$  et  $s$  s'obtient par instantiation de  $\sigma$ .



# Prolog : sémantique

Sémantique opérationnelle: Arbre de dérivation

On considère un programme  $P=c_1, \dots, c_k$  et un but  $G=g_1, \dots, g_l$ .

Soit  $c_i = t_i :- b_i^1, b_i^2, \dots, b_i^{l_i}$ .

(si  $l_i = 0$  alors  $c_i$  est une assertion, sinon c'est une règle).

On définit noeuds et arcs de l'arbre de dérivation de  $G$  pour  $P$  par induction (sur la hauteur) :

- la racine est  $G$ .
- Soit  $H=h_1, \dots, h_l$  un noeud de hauteur  $n$ , et soit  $c_s$  une clause de  $F$  dont la tête  $t_s$  s'unifie avec  $h_1$ , avec mgu  $\sigma$ .

Alors on crée le noeud, de hauteur  $n + 1$ ,  $H' = b_s^1 \sigma, \dots, b_s^{l_s} \sigma, h_2 \sigma, \dots, h_l \sigma$  et on étiquette l'arc de  $H$  à  $H'$  par  $\sigma$ .

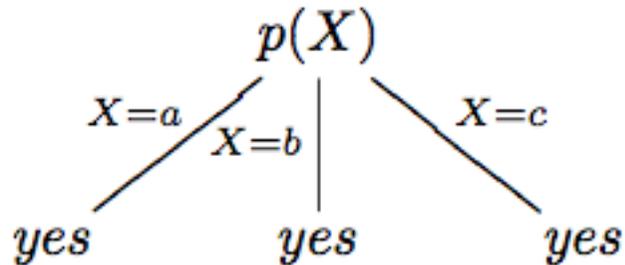
Une feuille est soit un but vide (succes), soit un but dont le premier prédicat ne s'unifie avec aucune tête de clause (echec).

# Prolog : sémantique

Sémantique opérationnelle: exemples d'arbres de dérivation

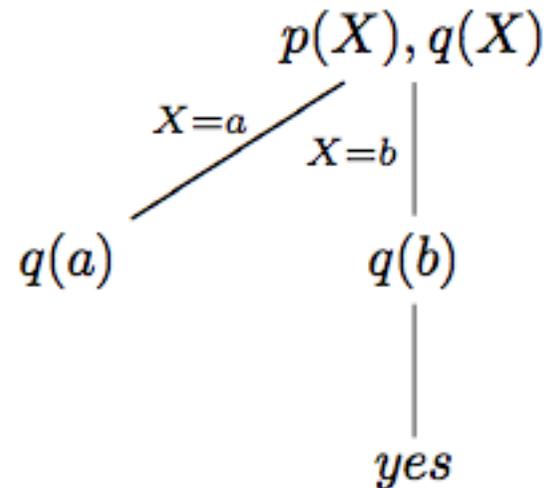
$P = p(a). \quad p(b). \quad p(c).$

$G = p(X)$



$P = p(a). \quad p(b). \quad q(b).$

$G = p(X), q(X)$



# Prolog : sémantique

Sémantique opérationnelle: arbres de dérivation

L'exécution d'un goal  $G$  pour un programme  $P$  a comme résultat l'ensemble de feuilles succès de l'arbre de dérivation correspondant.

Plus précisément, pour chacune de ces feuilles, le résultat est l'ensemble des instantiations des variables de  $G$  qui se trouvent sur le chemin qui mène de la racine à la feuille en question.

L'arbre de dérivation est produit par un parcours en profondeur d'abord. Donc l'ordre des clauses est important.

Par exemple, pour  $p(X):-p(X). p(a)$  le but  $p(X)$  ne donne aucun résultat (boucle), mais pour  $p(a). p(X):-p(X)$  le résultat  $X=a$  est atteint.

# Le langage : termes et prédicats

## Comment modéliser un problème

Les entités, les objets dont on parle: **les termes**, construits à l'aide des **symboles de fonction**.

Les relations entre termes: les **formules**, construites à l'aide des **symboles de prédicats**.

### **exemple:**

Adam aime les pommes. Clara aime les carottes.

Olivier aime les oranges. Les pommes sont des fruits.

Les oranges sont des fruits. Les carottes sont des légumes. Ceux qui aiment les fruits sont en bonne santé.

Les objets dont on parle (symboles de fonction):

**adam/0, pommes/0, clara/0, carottes/0, olivier/0, oranges/0**

Les relations entre objets (symboles de prédicat):

**fruit/1, legume/1, aime/2, bonne sante/1**

# Le langage : termes et prédicats

## Comment modéliser un problème

Le programme correspondant à ce choix de termes et prédicats:

```
aime(adam,pommes).
```

```
aime(clara,carottes).
```

```
aime(olivier,oranges).
```

```
fruit(pommes).
```

```
fruit(oranges).
```

```
legumes(carottes).
```

```
bonne_sante(X):-aime(X,Y),fruit(Y).
```

# Le langage : termes et prédicats

## Un choix alternatif

Les objets dont on parle (symboles de fonction):

adam/0, pommes/0, clara/0, carottes/0, olivier/0,  
oranges/0, fruit/0, legume/0

Les relations entre objets (symboles de prédicat):

aime/2, bonne\_sante/1, est\_instance\_de/2

Le programme correspondant:

```
aime(adam,pommes).
```

```
aime(clara,carottes).
```

```
aime(olivier,oranges).
```

```
est_instance_de(pommes,fruit).
```

```
est_instance_de(oranges,fruit).
```

```
est_instance_de(carottes,legumes). bonne_sante(X):-
```

```
aime(X,Y),est_instance_de(Y,fruit).
```

# Le langage : termes et prédicats

## Comment modéliser un problème (2)

Un deuxième exemple, où les termes ne sont pas forcément des constantes.

Voici ceux qu'il faut modéliser:

**Un arbre binaire** est soit une feuille ,  
soit un noeud dont les deux fils sont des arbres binaires.

**La hauteur** d'une feuille est 0,  
et la hauteur d'un noeud est le max des hauteurs de ses fils plus 1.

**La taille** d'une feuille est 0, et la taille d'un noeud est la somme des tailles de ses fils plus 1.



# Le langage : termes et prédicats

## Comment modéliser un problème (4)

Les symboles de prédicat:

hauteur/2, taille/2

Assertions et règles:

hauteur(feuille,0).

hauteur(noeud(T1,T2), R):-

    hauteur(T1,H1),

    hauteur(T2,H2),

    max(H1,H2,S),

    R is S+1.

taille(feuille,0).

taille(noeud(T1,T2), R):-taille(T1,R1),taille(T2,R2),R is R1  
+ R2 + 1.

# Les listes

## Représentation des listes

- une liste est
  - soit la constante « liste vide » : le terme []
  - soit un terme à deux arguments: le premier élément (head) et le reste de la liste (tail).
- Le premier élément d'une liste est un terme quelquonque, le reste doit être une liste.
- Le symbole fonctionnel ./2 est le constructeur de listes : **.( Premier, Reste)**
- exemple: **.( paul, .( pierre, .( marc, .(marie, []))))**
- une autre manière d'écrire la même liste:  
**[paul,pierre,marc,marie]**

# Les listes

## Représentation des listes (2)

- On peut aussi utiliser la notation **[Premier | Reste]** au lieu de **.( Premier, Reste)**
- On peut énumérer plusieurs éléments avant le |.

Par exemple:

$$[a,b,c] = [a | [b, c]] =$$
$$[a, b | [c]] = [a, b, c | []]$$

# Les listes

## Opérations sur les listes

- On définit la relation d'appartenance **membre(X,L)**.
- Le but **membre(X,L)** doit réussir si X est un des éléments de la liste L.
- $\text{membre}(b,[a, b, c])$  et  $\text{membre}([b, c],[a, [b, c]])$  sont vrais.
- $\text{membre}(b,[a, [b,c]])$  est faux.
- On peut observer que X est un membre de L,
  - si est X est le premier élément de L (cas de base), ou
  - X est membre du reste de L (cas récursif)

# Les listes

En Prolog:

```
membre( X, [X | L]).
```

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

# Les listes

## Opérations sur les listes (2)

### Concaténation:

- On définit la relation:  $\text{concat}(L1, L2, L3)$  telle que  $\text{concat}(L1, L2, L3)$  est vrai si et seulement si la liste  $L3$  est la concatenation de  $L1$  et  $L2$ .
- Pour définir  $\text{concat}(L1, L2, L3)$  on peut considérer deux cas, en fonction de la forme du premier argument  $L1$ :
  - Si  $L1$  est vide, alors  $L3$  doit être égale à  $L2$  (cas de base).
  - Sinon,  $L1$  est de la forme  $[X | L]$ . Dans ce cas, le résultat est la liste  $[X | L3]$  où  $L3$  est la concaténation de  $L$  et  $L2$  (cas récursif ).
  - ça donne en PROLOG:  
 $\text{concat}([], L, L).$   
 $\text{concat}([X | L], L2, [X | L3]) :- \text{concat}(L, L2, L3).$

# Les listes

## Exemples d'utilisation de concat:

?- concat([a,b,c],[1,2,3],[a,b,c,1,2,3]).

yes

?- concat([a,b,c],[1,2,3],L).

L = [a, b, c, 1, 2, 3]

?- concat([a,[b,c],[[]]],[a,[]],L).

L = [a, [b, c], [[]], a, []]

# Les listes

## Exemples d'utilisation de concat:

?- concat(L1,L2,[3,4,b]).

L1 = []

L2 = [3, 4, b] ;

L1 = [3]

L2 = [4, b] ;

L1 = [3, 4]

L2 = [b] ;

L1 = [3, 4, b]

L2 = [] ;

no

# Les listes

## Appartenance en utilisant concat

On peut re-définir la relation membre en utilisant concat.

```
membre1( X, L ) :-  
    concat( L1, [X | L2], L ).
```

Ou

```
membre1( X, L ) :-  
    concat( _, [X | _], L ).
```

# Les listes

## Enlever un élément:

- Enlever l'élément X d'une liste L peut être réalisé par un prédicat **enlever(X,L,L1)** où L1 est la liste L privée de l'élément X
- Il y'a deux cas:
  - Si X est le premier de L, alors le résultat est le reste de L (cas de base).
  - Si X est dans le reste de L, alors il est enlevé du reste (cas récursif ).

- En Prolog:

enlever( X, [X | Q], Q).

enlever( X, [Y | Q], [Y | Q1]) :- enlever(X, Q, Q1).

# Les listes

- Si  $X$  apparaît plusieurs fois dans  $L$ , des retours en arrière successifs donnent toutes les possibilités.
- enlever échoue si  $X$  n'appartient pas à la liste.

# Les listes

Exemples d'utilisation de enlever:

?- enlever( a, [a, b, b, a, b], L).

L = [b, b, a, b] ;

L = [a, b, b, b] ;

no

?- enlever( X, [a, c, c], L).

X=a L = [c, c] ;

X=c L = [a, c] ;

X=c L = [a, c] ;

no

# Les listes

## Exemples d'utilisation de enlever:

?- enlever( a, L, [c, d, e]).

L = [a, c, d, e] ;

L = [c, a, d, e] ;

L = [c, d, a, e] ;

L = [c, d, e, a] ;

no

# Les listes

## Insérer un élément:

- Avec enlever, on peut définir insérer( $X,L,L1$ ) qui insère l'élément  $X$  dans la liste  $L$ , donnant la liste  $L1$  comme résultat.

insérer( $X,L,L1$ ) :-

enlever( $X,L1,L$ ).

- On peut aussi définir la relation membre en utilisant enlever. Un terme  $X$  est membre d'une liste si on peut l'enlever de cette liste.

membre2( $X, L$ ) :-

enlever( $X, L, \_$ ).