

PLAN

- Les origines du Prolog.
- Deux exemples de programmes.
- Syntaxe
 - Constantes, variables, termes, prédicats.
 - Assertions, Règles, Buts.
- Sémantique
 - Unification
 - Arbres de dérivation
- 2ème partie
 - L'arithmétique en Prolog.

PROGRAMMER EN LOGIQUE

Les origines (vers 1970):

- A. Colmerauer et al., Univ. d'Aix-Marseille.

Un système de communication homme-machine en français.

- R. A. Kowalski, Imperial College, London.

Predicate logic as a programming languages.

Depuis, un grand nombre d'implementations. Voir par exemple:

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

La version que nous allons utiliser:

YAP (Yet Another Prolog)

LIACC, Université de Porto.

<http://www.ncc.up.pt/~vsc/Yap/>

PLAN

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

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.
```

On interroge le programme: buts

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,_,_,_,_).
```

Interaction au top level (1)

```
/* 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.  
  
# yap  
  
?- consult(bio).  
  
yes  
  
?- bio(louis14,-,X,-,-,-).  
  
X = 1638 ?;  
  
no
```


Interaction au top level (2)

```
?- 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
```


Interaction au top level (3)

```
?- bio(X,h,-,-,-,-).
```

```
X = louis13 ?;
```

```
X = louis14 ?;
```

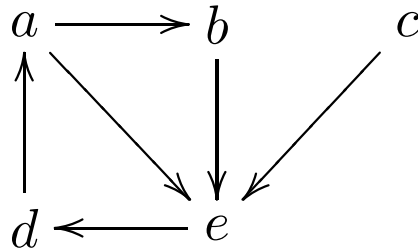
```
no
```

```
?- 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).
```

Interaction au top level

```
?- consult(graph).
```

```
yes
```

```
?- connexe(a,b).
```

```
yes
```

```
?- arc(a,X).
```

```
X = b ? ;
```

```
X = e ? ;
```

```
no
```

```
?- connexe(a,c).
```

```
boucle!
```

PLAN

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

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).

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 punctuations	<code>:- , . () ; ...</code>

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)

PLAN

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

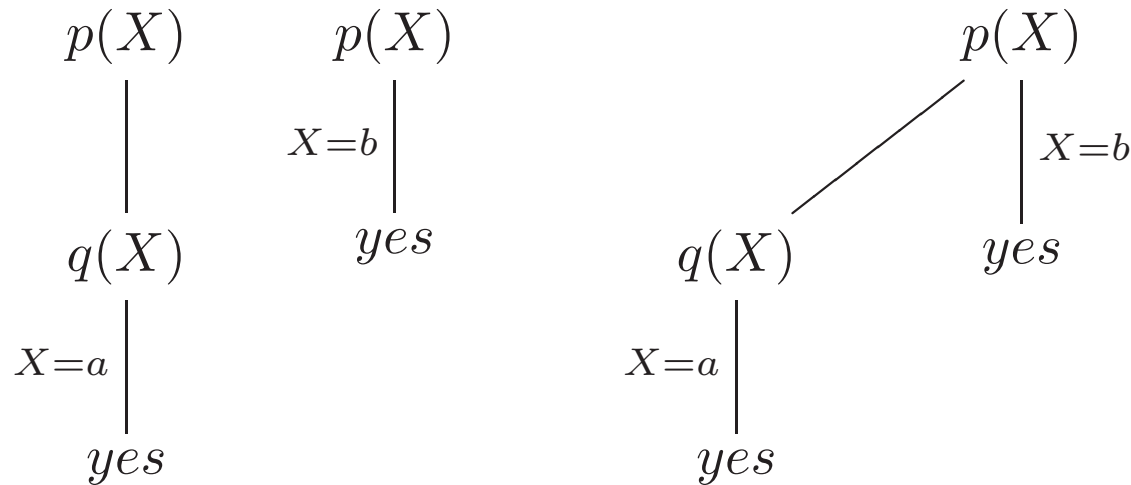
Sémantique opérationnelle: un exemple

$q(a)$.

$p(X) : \neg q(X)$.

$p(b)$.

but: $p(X)$



Arbre de dérivation de $p(X)$

Sémantique opérationnelle: l'unification (1)

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

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

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

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

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

Sémantique opérationnelle: l'unification (2)

Le prédicat binaire infix = de Prolog calcule les unificateurs (mgu)

?- $f(X) = f(g(a, Y))$.

$$X = g(a, Y)$$

?- $f(X,X)=f(g(a,Y),g(Z,b))$.

$$X = g(a, b),$$
$$Y = b,$$
$$Z = a$$

?- $X=f(X)$.

$$X =$$

```
f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f... Segmentation fault
```

Pas de *occur check*!

Sémantique opérationnelle: arbres de dérivation

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

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

(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 P dont la tête τ_s s'unifie avec h_1 , avec mgu σ .

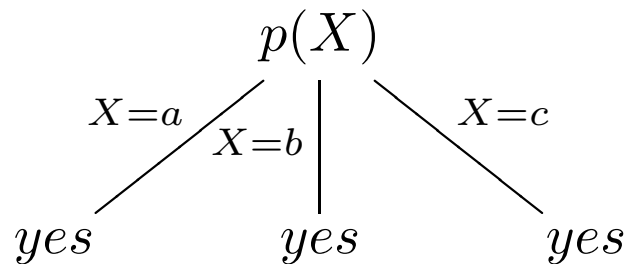
Alors on crée le noeud, de hauteur $n + 1$, $H' = b_s^1\sigma, \dots, b_s^{l_s}\sigma, h_2\sigma, \dots, h_l\sigma$ et on étiquette l'arc de H à H' par σ .

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

Sémantique opérationnelle: exemples d'arbres de dérivation

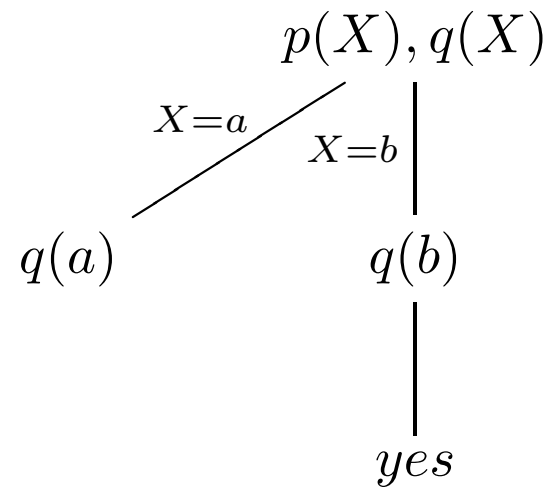
$P = p(a) . \quad p(b) . \quad p(c) .$

$G = p(X)$



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

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



Sémantique opérationnelle: arbres de dérivation

L'exécution d'un goal G pour un programme P a comme résultat l'ensemble de feuilles succes 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 de manière préfixe (parcours en profondeur d'abord).

Donc l'ordre des clause est important. Par exemple, pour $p(X) :- p(X) .$ $p(a)$ le but $p(X)$ ne donne aucun résultat (boucle), mais pour $p(a) . \quad p(X) :- p(X)$ le résultat $X=a$ est atteint.

2ème partie

Introduction a YAP : les entiers

Quelques symboles de fonctions sur les entiers

Plusieurs opérateurs sont prédéfinis pour des opérations arithmétiques simples.

- `+` l'addition
- `-` la soustraction
- `*` la multiplication
- `/` la division
- `**` la puissance
- `//` la division entière
- `mod` le reste de la division entière

L'expression `3 + 4` est à considérer en Prolog comme synonyme de `+(3,4)`, un terme d'arité 2 avec symbole fonctionnel `+` et arguments 3 et 4.

MCours.com

Quelques prédicats sur les entiers

A la requête `?- X = 3 + 4` Prolog répond `X = 3 + 4`

Pour évaluer `3 + 4` il faut utiliser le prédicat (extra-logique) `is` .

Exemples:

```
?- Y is 4 + 3.
```

```
Y = 7
```

```
Yes
```

```
?- X is Y + 1.
```

```
ERROR: Arguments are not sufficiently instantiated
```

```
?- Y = 2, X is Y + Y.
```

```
Y = 2
```

```
X = 4
```

```
Yes
```


Quelques prédicats sur les entiers (2)

Comparaisons

- $X > Y$
- $X < Y$
- $X \geq Y$
- $X \leq Y$
- $X ::= Y$ Les valeurs de X et Y sont égaux
- $X \neq Y$ Les valeurs de X et Y sont différents

Dans tous ces prédicats, les deux arguments doivent être clos.

L'opérateur $\neq$ est différent de $::=$

?- $1 + 3 ::= 3 + 1.$

Yes

?- 1 + 3 = 3 + 1.

No

?- 1 + A =:= B + 2.

ERROR: Arguments are not sufficiently instantiated

?- 1 + A = B + 2.

A = 2

B = 1

Yes

?- X = 3 + 2, X < 7.

X = 3+2

Yes

?- Y = 3, X is 3+Y, X + Y < 9.

No

MCours.com