PROLOG
- Concepts de base -

Michel RUEHER

PLAN DU COURS

I - Introduction :

1 - Documents en ligne

2 - Concepts de base : un langage de haut niveau, un langage déclaratif

II – Éléments syntaxique du langage PROLOG

1 - Les termes, les atomes logiques, les clauses, les Listes (syntaxe et manipulations élémentaires)

2 - Portée et quantification des variables

III - Sémantique d'un programme Prolog (Présentation informelle) :

1 - Signification logique : dénotation

2 - Signification constructive : arbre de preuve; signification opérationnelle : arbre de recherche

3 - Vision procédurale de Prolog

IV - Contrôle et la Négation

1) Le Contrôle
    - Définition de la Coupure — Exemples
    - Applications de la Coupure
2) La Négation : définition de la *négation par échec* — Exemples

UN EXEMPLE

```
append([],L,L).
append([H|T],L,[H|R]) :- append(T,L,R).
```

UN LANGAGE DE HAUT NIVEAU

- **Programmation Procédurale**

  Instruction = *Ordre*

  → *Ordre pour la machine*

  → *Ordre dans l'énoncé*

  ⇒ Spécification d'une solution en terme de *comportement de la machine*

- **Programmation Fonctionnelle**

  Instruction = *Fonction* ⇒ Spécification d'une solution en terme de *valeurs calculées*

- **Programmation en Logique**

  Instruction = *Relation*

  ⇒ Spécification d'une solution en terme de *relations* entre entités (ou classes d'entités)

  ⇒ **Programme en logique** ≅ **spécification exécutable**

UN LANGAGE DECLARATIF

- **Programmer en logique = Décrire l'univers** du problème

- **Programme Prolog = Ensemble de propriétés et relations entre les objets de l'univers**

  Un programme Prolog *ne décrit pas une solution* : c'est une suite *d'affirmations*

- **Problème = Ensemble de questions** concernant certains objets

- **Exécution = Déduction de nouvelles relations** à partir des affirmations du programme

CONSTITUANTS ELEMENTAIRES ET TERMES

Constituants élémentaires :

- **Variable :** *objet inconnu de l'univers du problème*
  - chaîne commençant par une majuscule ou par _ (variable anonyme : _)
  - exemples : *X*, *Y1*, *_ObjetInconnu*, ...
- **Constante :** *objet particulier de l'univers du problème*
  - nombre : *12*, *1.056*, ...
  - chaine commençant par une minuscule : *toto*, *a*, *jean_paul_II*, ...
  - chaine entre "" : "*Constante chaine*", "*123*",

*Un **terme** est soit une constante, soit une variable, soit un terme fonctionnel*

Un ***terme fonctionnel*** est de la forme ***f(t1,…,tn)*** avec:

- *f* un symbole fonctionnel,
- *t1*,…,*tn* : suite de termes

Exemples : *succ(zero)*, *f(X,12)*, *adresse(2,"rue des mimosas", valbonne)*, ...
les constantes sont des fonctions d'arité nulle)

LES LISTES : SYNTAXE ET MANIPULATION

La liste est un terme fonctionnel

- **Définition**
  - Foncteur : . - Arité : 2
  - Arguments :
    - premier argument : terme,
    - deuxième argument : liste

- **Notation :**
  - syntaxe : .(*terme*,*liste*) `ou` [*terme* | *liste*]
  - Notation simplifiée:
    .(*terme1*,.(*terme2*,.(*terme*$_n$,.(...,*liste*),...) `peut s'écrire :` [*terme1*,*terme2*,...,*terme*$_n$ | *liste*]
  - la liste vide est notée : []
  - **Exemples :**

'.'(*a*, '.'(*b*, '.'(*c*, []))) notée *[a,b,c]*

'.'(*Tête*, *Queue*) notée *[Tête | Queue]*

'.'(1, '.'(2, X)) notée [1, 2 | X]

LES ATOMES

- **atome logique :** *propriété, relation entre termes*

  Syntaxe : **symbole_de_prédicat(term$_{e_1}$,…,term$_{e_n}$)**
  n : arité du prédicat

  Exemples:
  est_pere_de(pierre,paul), temps(ensoleillé)
  est_mere_de(X,paul), atome_sans_termes

- **atome clos :** *atome sans variables*

  Exemples:
  est_pere_de(pierre,paul), temps(ensoleillé)

LES CLAUSES

- **clause :** *relation (certaine ou conditionelle)*

T :- Q1,…,Qn.

T : littéral *positif*, appelé *Tête de Clause*
Q1,...,Qn : suite de littéraux *négatifs* appelée *Corps de clause*.

Si { Q1,...,Qn } ≠ ∅ et T ≠ ∅, la clause est une
affirmation conditionnelle *(règle)*
Exemple *même_pere(X,Y) :-*
pere_de(P,X), pere_de(P,Y).

Si { Q1,...,Qn } = ∅, la clause est une **affirmation inconditionnelle** *(fait)*
exemple : *homme(pierre).*

Si T = ∅, la clause est une question ***(dénégation)***
exemple : *?-homme(pierre).*

- **Sémantique informelle**

  Si tous les atomes du corps sont vrais, alors l'atome de tête est vrai

  *':-' : **Implication l**ogique*
  *',' : **ET logique***

QUANTIFICATION DES VARIABLES

- **Portée des variables :** les variables sont locales aux clauses.
- **Quantification des variables**

  Soit une clause ***A :- B*** et une variable x

  Si x ∈ A et x ∈ B, alors x est quantifié universellement dans A et B

  ∀ x (Vrai B ⇒ Vrai A)

  Si x ∈ A et x ∉ B, ou si B = ∅, alors x est quantifié universellement dans A

  Vrai B ⇒(∀ x Vrai A)

  Si x ∈ B et x ∉ A,ou si A = ∅, alors x est quantifié existentiellement dans B

  (∃ x | Vrai B)⇒ Vrai A

- **Exemple :**

```
même_pere(X,Y):-pere(P,X),pere(P,Y).

∀X∀Y.(∃P.(pere(P,X) et pere(P,Y))

                    => meme_pere(X,Y))
```

PROGRAMME ET PAQUETS

- **Un programme Prolog :** suite de clauses regroupées en paquets
- **Paquet = ensemble de clauses qui ont :**
  - **le même *symbole de prédicat* en tête de clause**
  - **la même *arité*.**

Deux clauses d'un même paquet sont liées par un *ou* logique.

```
parent(X,Y) :- est_pere_de(X,Y).
parent(X,Y) :- est_mere_de(X,Y).
```

- **Remarque :**

Un prédicat est défini par une conjonction de clauses.

Soit le prédicat p défini par le programme :

```
p  :-  a1,  a2,  a3.
p  :-  b1, b2.
```

On a: **p ⇐ (a1 & a2 & a3) ∨ (b1 & b2)**

D'ou **(p ∨ ¬ a1 ∨ ¬ a2 ∨ ¬ a3) & (p ∨ ¬ b1 ∨ ¬ b2)**

LA SYNTAXE PROLOG : RECAPITULATIF

Programme = ensemble de *Paquets*

Paquet = ensemble de *Clauses* qui ont le même prédicat (i.e., même symbole de prédicat et même arité de prédicat) comme tête de clause

Clause =

Atome_logique '.'
| Atome_logique ':-' Atome_logique ',' ... ',' Atome_logique '.'

Atome_logique =

Symbole_de_prédicat
| Symbole_de_prédicat '(' Terme ',' ... ',' Terme ')'

Terme =

Constante
| Variable
| Symbole_de_fonction '(' Terme ',' ... ',' Terme ')'

Constante = Entier | Réel | '"' Caractère* '"' |
Minuscule (Car_alphanum | '_')*

Variable = Majuscule (Car_alphanum | '_')* | '_'

Symbole_de_prédicat =
Minuscule (Car_alphanum | '_')*

Symbole_de_fonction =
Minuscule (Car_alphanum | '_')*

L'EXEMPLE DE LA FAMILLE

Programme :

```
/* fils(Pere,Mere,Fils*/                                   ou
fils(claude, nicole, françois).        fils(parents(claude, nicole),françois).
fils(daniel, marie, nicolas).          ...

/* fille(Pere,Mere,Fille*/                ou
fille(claude, nicole, claire).         fille(parents(claude, nicole), claire).
fille(daniel, marie, virginie).        ...

/* pere(Pere,Enfant) */                 ou
pere(P,E):- fils(P,_,E)                pere(P,E):- fils(parents(P,_),E)
pere(P,E):- fille(P,_,E).        ...

/* mere(Mere,Enfant) */                 ou
mere(M,E) :- fils(_,M,E).              mere(M,E):- fils(parents(_,M),E)
mere(M,E) :- fille(_,M,E).             ...

/* parent(Parent,Enfant) */             ou
parent(P,E) :- fils(_,P,E).            parent(P,E):- fils(parents(P,_),E)
parent(P,E) :- fils(P,_,E).            parent(M,E):- fils(parents(_,M),E)
```

/* parents(Père,Mère,Enfant) */

parents(P,M,E) :- pere(P,E), mere(M,E).

Exemples de questions :

?-parent(P, françois)

P = claude

P = nicole

True

?-parent(P,théodore)

false

?-parents(P, M, claire)

P = claude

P = nicole

True

SIGNIFICATION LOGIQUE

- **Signification logique d'un programme P : Dénotation de P**

DEN(P) = Ensemble des atomes qui sont des conséquences logiques de P
(ensemble souvent infini)

Exemple 1 :

P= { p(a).
p(b).
q(c).
q(X) :- p(X).
f(X,Y) :- q(X), p(Y).}

DEN(P) = {p(a),p(b),
q(c),q(a),q(b),
f(a,a),f(a,b),
f(b,b),f(b,a),
f(c,b),f(c,a)}

Exemple :

```
P={  plus(zero,X,X).
     plus(suc(X),Y,suc(Z)) :- plus(X,Y,Z).  }
```

$DEN(P) = \{ plus(zero,X,X), plus(suc(zero),Y,suc(Y)),$
$plus(suc(suc(zero)),Y,suc(suc(Y))),$
$plus(suc(suc(suc(zero))),Y,suc(suc(suc(Y)))),...\}$
$= \{plus(suc^n(zero),A,suc^n(A)),\ \forall\ n \geq 0, \forall A \in T\}$
avec T : ensemble des termes de P

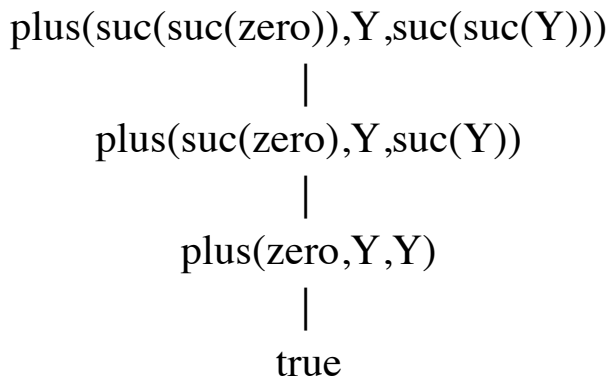
- Pour un programme P, la réponse Prolog à une question A est l'ensemble S des instances de A appartenant à la dénotation de P

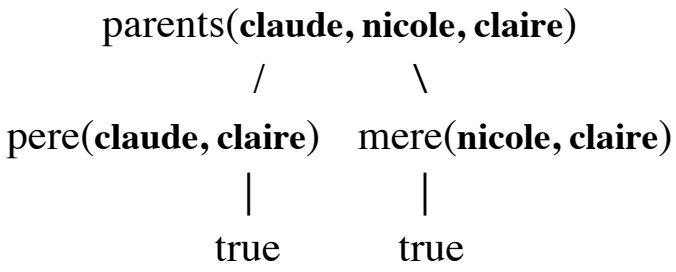
  S ={s(A) / s(A) ∈ DEN(P)}

SIGNIFICATION CONSTRUCTIVE

- **Objectif :** Démontrer qu'un atome est conséquence logique d'un programme
- **Approche:** Utilisation des arbres de preuve (arbres finis orientés)
- **Arbre de preuve:**
  - *feuilles* : *vrai*
  - à chaque *nœud non terminal* est associé une instance *i* d'une clause tel que :
    - nœud = tête de *i*,
    - fils du nœud = atomes du corps de *i* (si le corps de *i* est vide, alors le nœud a pour unique fils *vrai*)
- **Propriétés des arbres de preuve :**
  - tout sous arbre d'un arbre de preuve est un arbre de preuve,
  - un arbre de preuve peut comporter des variables,
  - toute instance d'un arbre de preuve est un arbre de preuve
- **Théorèmes :**
  - A est conséquence logique de P si et seulement si A est racine d'un arbre de preuve.
  - DEN(P) = {A | A = racine d'un arbre de preuve}
- La construction d'un arbre de preuve est **non déterministe**
  - choix de clauses
  - choix de feuilles

SIGNIFICATION CONSTRUCTIVE (EXEMPLES D'ARBRE DE PREUVE)





SIGNIFICATION OPERATIONNELLE

Signification Opérationnelle = Méthode *déterministe* de construction d'un arbre de recherche

⇒ Calcul de toutes les instances d'un but appartenant à la dénotation du programme

Principe:

Parcours en *profondeur d'abord* et de *gauche à droite* de l' arbre de recherche

→ choix des clauses à partir de la première du paquet
→ choix des feuilles à partir de la gauche

⇒ **Stratégie de recherche est correcte :** DEN(P) ⊇ Ensemble des réponses

⇒ **Stratégie de recherche n'est pas complète :** l'arbre de recherche peut être infini

- l'ordre des clauses dans un paquet est significatif
- l'ordre des atomes dans une clause est significatif

VISION PROCEDURALE DE PROLOG

Question ≈ Appel de procédure
Unification ≈ Transmission de paramètres
Paquet ≈ Procédure
Clauses d'un paquet ≈ Définition de la procédure

Exemple :

```
add(zero,X,X).
add(suc(X),Y,suc(Z)) :- add(X,Y,Z).
        ≈
procédure add(arg1, arg2, arg3) :
  if arg1 = zero then unify(arg2,arg3)
  elseif unify(arg1,suc(X)) and   unify(arg3,suc(Z))
  then add(X, arg2, Z)
  endif
end add
```

LE CONTROLE

Problèmes :

- **Coût élevé du parcours** de l'ensemble de l'arbre de recherche,

- Expression de la **connaissance négative**

LA COUPURE

- La *coupure* est un atome, noté **!**
- La *coupure* est **sans signification logique**
- L'appel de la coupure réussit toujours
- **L'appel de la coupure a pour effet de bord de modifier l'arbre de recherche**
- L'appel de la coupure supprime toutes les branches en attente dans l'arbre depuis l'appel de la clause qui la contient

LA COUPURE — EXEMPLES —

Exemple

```
r(b,b1).        q(a).
r(c,c1).        q(b).
r(a,a1).        q(c).
r(a,a2).        r(a,a3).

p(X,Y) :- q(X), r(X,Y).
p(d,d1).

p1(X,Y) :- q(X), r(X,Y), !.
p1(d,d1).

p2(X,Y) :- q(X), !, r(X,Y).
p2(d,d1).

p3(X,Y) :- !, q(X),  r(X,Y).
p3(d,d1).
```

APPLICATION DE LA COUPURE

- **Recherche déterministe de la première solution**

  Exemple :

  ?- grand_pere(X,Y), !.

- **Optimisation :** évite des recherches inutiles, voire infinies ...

  Exemple :

  plus(zero,X,X).
  plus(suc(X),Y,suc(Z)) :- plus(X,Y,Z).
  ?- plus(X,zero,Z), plus(suc(X),Z,suc(zero)), !.

- **Masquage d'une définition incomplète :** *mauvaise* utilisation

  Exemple :

  fact(0,1):- ***!.*** `au lieu de` *fact(0,1).*
  fact(X,Y) :- *fact(X,Y) :-*
  X1 is X-1, ***X \== 0,***
  fact(X1,Y1), *X1 is X-1,*
  *Y is X*Y1.* *fact(X1,Y1),*
  *Y is X*Y1.*

LA COUPURE (1)

→ **EXPRIMER LE DETERMINISME**

→ **OPTIMISER L'ESPACE DE RECHERCHE**

Exemples:

```
minimum(X,Y,X) :- Y ≥ X, !.
minimum(X,Y,Y) :- X > Y, !.
```

LA COUPURE (2)

```
fusion([X|Xs],[Y|Ys],[X|Zs]) :-
   X < Y, !,fusion(Xs,[Y|Ys],Zs).

fusion([X|Xs],[Y|Ys],[X,Y|Zs]) :-
   X = Y, !,fusion(Xs,Ys,Zs).

fusion([X|Xs],[Y|Ys],[Y|Zs]) :-
   X > Y, !,fusion([X|Xs],Ys,Zs).

fusion(Xs,[],Xs):- !.
fusion([],Ys,Ys) :-  !.
```

Tous les cas sont mutuellement exclusifs

LA COUPURE (3)

→ **OMISSION DE *CONDITIONS* EXPLICITES**

→ **MODIFICATION DE LA *SEMANTIQUE* DU PROGRAMME**

Exemples:

```
minimum(X,Y,X) :- Y ≥ X, !.
minimum(X,Y,Y).
```

Et donc

```
?-minimum(X,Y,Y).
```

yes !!

```
member(X,[X|Xs]) :- !.
member(X,[Y|Ys]) :- member(X,Ys).
```

LA NEGATION

- **Absence de négation logique**

  Le principe de résolution "confisque" la négation logique disponible dans les clauses de Horn

  ⇒ *On ne peut exprimer en Prolog que le vrai*

  non(A) ∉ DEN(P)

- **La négation par l'échec**

  A n'est pas une conséquence logique de P

  → non (A est une conséquence logique de P)

  Définition en Prolog :

```
non(X) :- X, !, échec.      % Où échec est un
non(X).                     % prédicat faux.
```

  Prédicat prédéfini en Prolog : \+

LIMITES DE LA NEGATION PAR L'ECHEC

- **Basée sur l'hypothèse du *monde clos* :**

  " Tout ce qui n'est pas démontrable est FAUX "

- **Pas de sémantique précise :**

  Exemple :

  homme(pierre).
  homme(jacques).
  riche(pierre).

?- homme(X), \+(riche(X)).

> X = jacques

?- \+(riche(X)), homme(X).

> no