PROLOG: concepts de base

Michel RUEHER

PLAN DU COURS

I **Introduction:** un langage de haut niveau, un langage déclaratif

II **Éléments syntaxique du langage Prolog**

1 Les termes, les atomes logiques, les clauses, les Listes

2 Portée et quantification des variables

III **Sémantique d'un programme Prolog** :

1 - Sémantique logique, dénotation

2 - Signification opérationnelle: SLD, exploration en profondeur

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 langage déclaratif pour les problèmes combinatoires … et bien plus !

- Formulation en une dizaine de lignes de "petits" problèmes combinatoires comme **les n-reines, le sudoku, le carré magique**, …
- Facilité d'intégration de concepts comme les "contraintes" pour l'expression des **heuristiques** et une **résolution efficace des problèmes réels difficiles** (emploi du temps, ordonnancement, …)
- Outil remarquable pour le **parsing et l'évaluation** des expressions dans une grammaire formelle

Un exemple, ou plutôt deux !

```
lecture (lucy, baudelaire).
lecture (lea, vargas).
lecture (lucy, L):- lecture (lea, L).

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*
Programme en logique ≈ spécification exécutable

UN LANGAGE DÉCLARATIF *ET UNE SÉMANTIQUE LOGIQUE*

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*

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

- **Programme & questions : clauses de Horn**

PROLOG : ÉLÉMENTS SYNTAXIQUES

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 connu*** *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* **I** *liste*]
  - Notation simplifiée: .(*terme*$_1$,.(*terme*$_2$,.(*terme*$_n$,.(...,*liste*),...)
    --> [*terme*$_1$, *terme*$_2$, ... , *terme*$_n$ I *liste*]
  - la liste vide est notée : []
- **Exemples :**

  '.'(a, '.'(b, '.'(c, []))) --> *[a,b,c]*

  '.'(1, '.'(2, X)) --> [1, 2 | X]

LES ATOMES

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

  Syntaxe : **symbole_de_prédicat(terme1,…,termen)**
  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 conditionnelle*

T :- Q_1,…,Q_n. où T, Q_1, …, Q_n sont des atomes logiques

T : littéral *positif*, appelé *Tête de Clause*

Q_1,...,Q_n : suite de littéraux *négatifs* appelée *Corps de clause.*

Si { Q1,...,Qn } ≠ ∅ et T ≠ ∅ , la clause est une ***règle***

Exemple *même_pere(X,Y) :- pere_de(P,X), pere_de(P,Y).*

Si { Q1,...,Qn } = ∅ , la clause est un ***fait***

Exemple : *homme(pierre).*

Si T = ∅ , la clause est une ***question (dénégation)***

Exemple : *?-homme(pierre)*

Sémantique informelle et quantificateurs

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

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

- **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 (B ⇒ A)

  Si x ∈ A et x ∉ B alors x est quantifié universellement dans A : B ⇒ ∀x A

  Si x ∈ B et x ∉ A alors x est quantifié existentiellement dans B : (∃ x B) ⇒ A

  Exemple :

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

  ∀ X ∀Y (∃ P (pere(P,X) ∧ 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).

… mais un prédicat est défini par une ***conjonction*** de clauses !

(ensemble de clauses = axiomes)

Soit le prédicat p défini par : **p :- a_1, a_2, a_3.**
p :- b_1, b_2.

On a: $(p \Leftarrow (a_1 \wedge a_2 \wedge a_3)) \wedge (p \Leftarrow (b_1 \wedge b_2))$

$\equiv (p \vee \neg a_1 \vee \neg a_2 \vee \neg a_3) \wedge (p \vee \neg b_1 \vee \neg b_2)$

$\equiv p \vee \neg((a_1 \wedge a_2 \wedge a_3) \vee (b_1 \wedge b_2))$

$\equiv p \Leftarrow ((a_1 \wedge a_2 \wedge a_3) \vee (b_1 \wedge b_2))$

La syntaxe Prolog: Recapitulatif (1)

Programme : ensemble de *Paquets*

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

Clause :

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

Atome_logique :

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

La syntaxe Prolog: Recapitulatif (2)

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(claude, nicole, françois).

fille(claude, nicole, claire).

pere(P,E):- fils(P,_,E)

mere(M,E) :- fils(_,M,E).

parent(P,E) :- fils(_,P,E).

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

fils(daniel, marie, nicolas).

fille(daniel, marie, virginie).

pere(P,E):- fille(P,_,E).

mere(M,E) :- fille(_,M,E).

parent(P,E) :- fils(P,_,E).

Questions:

?-parent(P, françois)

P = claude P = nicole

?-parent(P,théodore)

false

?-parents(P, M, claire)

P = claude M = nicole

Sémantique d'un programme Prolog

- **Programme** : ***clauses de Horn*** (au plus un littéral positif)
- **Résultat** : ***« effet de bord » de la résolution*** (stratégie « input ordonnée »)
- Preuves limités aux littéraux positif → ***perte de la négation logique***
- Stratégie de recherche en profondeur d'abord est ***incomplète***

Dénotation d'un programme Prolog (1)

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

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)}

Dénotation d'un programme Prolog (2)

Exemple 2 :

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)), $\forall n \geq 0$, , $\forall A \in T$} où T = termes de P

Formalisation:

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)}

VISION PROCÉDURALE D'UN PROGRAMME PROLOG

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

Exemple :

```
add(0,X,X).
add(suc(X),Y,suc(Z)) :- add(X,Y,Z).
```

≈

```
procedure add(arg1, arg2, arg3) :
    if arg1 = 0 then arg2 = arg3
    | % Choix non-déterministe
    if arg1 = suc(X)) and   arg3 = suc(Z) then add(X, arg2, Z)
end add
```

Le contrôle en Prolog (1)

Problèmes :

- **Coût élevé du parcours** de l'ensemble de l'arbre de recherche,
- Besoin de « palliatif pour l'expression de la **connaissance négative**

Le contrôle en Prolog (2)

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

Le contrôle en Prolog: la Coupure (1)

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

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).
```

Le contrôle en Prolog: la Coupure (2)

?- p(X,Y).
X = a Y = a1 ; X = a Y = a2; X = a Y = a3 ;
X = b Y = b1 ; X = c Y = c1; X = d Y = d1 ; no
?- p(d,Y).
Y = d1

?- p1(X,Y).
X = a Y = a1 ; no
?- p1(d,Y). Y = d1

?- p2(X,Y).
X = a Y = a1; X = a Y = a2; X = a Y = a3; no
?- p2(d,Y).
Y = d1

?- p3(X,Y).
X = a Y = a1; X = a Y = a2; X = a Y = a3;
X = b Y = b1; X = c Y = c1; no
?- p3(d,Y). no

Le contrôle en Prolog – Applications (1)

- **Recherche déterministe de la première solution**

  ?- grand_pere(X,Y), !. % Option par défaut

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

```
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.
```

- **Optimiser l'espace de recherche**

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

Le contrôle en Prolog – Applications (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 négation en Prolog

- **Absence de négation logique**

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

  $\Rightarrow$ ***On ne peut exprimer en Prolog que le vrai***

  $\mathbf{non(A) \notin DEN(P)}$

- **La négation par l'échec**

  A n'est pas une conséquence logique de P

  $\rightarrow$ échec à montrer que A est une conséquence logique de P

  Définition en Prolog :

```
not(X) :- X, !, fail.        % Où fail est un prédicat faux.
not(X).
```

  Prédicat prédéfini en Prolog : \+

Limites de la négation par l'échec

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

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

- **Pas de sémantique précise / claire :**

  Exemple :

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

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

X = jacques

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

no