Intelligence Artificielle-Programmation Logique avec Prolog

Licence en Mathématiques (semestre 6)

Abdoulaye SERE
Maître assistant en Informatique,
UFR-ST/ Université Polytechnique de Bobo-Dioulasso

(cours de 2016-2017)

Plan

- **Introduction**
- **Aperçu sur Prolog (vue sur SWI-Prolog)**
- **Logique du 1er ordre (logique des propositions, calcul des predicats, calcul des relations)**
- **Application en PROLOG(Listes, fonctions, mise en oeuvre du raisonnement...)**

Introduction

1972 : Marseille, A. Colmerauer

Prolog est un langage de développement en Intelligence Artificielle.

Introduction : histoire de prolog

- 1971 : création de Prolog par A. Colmerauer et P. Rousse suite à des travaux sur la détection automatique
- 1972, PROLOG I
- 1975, fin premier manuel PROLOG I
- 1977, premier compilateur PROLOG (Warren)
- 1978, multiplication des compilateurs

Introduction : histoire de prolog

• 1983, • machine abstraite (WAM) PROLOG (Warren) =>
– Quintus-Prolog, SB-Prolog, SWI-Prolog, GNU-Prolog, ...
• PROLOG II -> Version commerciale
• 1989, PROLOG III -> solveur permettant de résoudre des co
linéaires
• 1996, PROLOG IV -> solveur, basé sur l'arithmétique des int
• travaux sur compilation, implantations parallèle, algorithmes
d'unification par contrainte, ...

Introduction : exemples domaines d'utilisation de Prolog

- Analyse du langage naturel ;
- Modelisation des raisonnements (les Systèmes experts) ;
- Bases de données ;
- D'autres domaines (vus en exposés)

→ **un langage de programmation déclaratif ;**
→ **un langage de requêtes.**

Introdution

- Langage d'expression des connaissances, fondé s
  la logique des prédicats du 1er ordre

- L'utilisateur définit une base de connaissances (de
  faits, des règles)

- Il n' y a pas d'instructions, pas d'affectation, pas d
  boucles explicites comme dans certains langages
  programmation (c, c++, java, python)

Introdution

- Uniquement des instructions exprimant des faits, d
  règles, des questions

- L'interpréteur PROLOG explore cette base de
  connaissances pour répondre à des interrogations
  utilisateurs

Introduction

Il existe de nombreux interpréteurs Prolog
C-Prolog, Sixtus Prolog
Nous allons utiliser **SWI-Prolog (actuellement version 7**

Programmation logique avec PROLOG (trois niveaux dans un programme PROLOG)

- **La base de faits** : base de connaissance
- **La base de règles :** logique de raisonnement;
- **L'interrogation (les questions)** : les questions posées par l'utilisateur
- En swi-prolog, la base de faits et la base de règl sont dans le même fichier .pl

Aperçu sur Prolog : la base de faits

- pere(seydou, jeanne).
- pere(seydou, michel).
- pere(michel,salif).
- mere(sylvie, salif).
- mere(sylvie, luc).
- pere(georges, sylvie).
- sexe(seydou, m).
- sexe-sylvie, f).

Aperçu sur Prolog : la base de règles

/* les règles grandpere et grandmere */
grandpere(X, Y):-pere(X, Z), pere(Z, Y).
grandpere(X, Y):-pere(X, Z), mere(Z, Y).
grandmere(X, Y):-mere(X, Z), mere(Z, Y).
grandmere(X, Y):-mere(X, Z), pere(Z, Y).

- / * frere(X, Y) */
  frere(X, Y):-mere(Z, X), mere(Z, Y), sexe(X, m), sexe(Y, m).
  frere(X, Y):-pere(Z, X), pere(Z, Y), sexe(X, m), sexe(Y, m).

- /*enrichir la base des faits*/

cousin(X, Y):-pere(Z, X), pere(W, Y), frere(Z, W), sexe(X, m), sexe(Y, m).
cousin(X, Y):-mere(Z, X), mere(W, Y), soeur(Z, W), sexe(X, m), sexe(Y, m).

Aperçu sur Prolog : la base de faits

- pere(seydou, jeanne).
- pere(seydou, michel).
- pere(michel,salif).
- mere(sylvie, salif).
- mere(sylvie, luc).
- pere(georges, sylvie).
- sexe(seydou, m).
- sexe-sylvie, f).

Aperçu sur Prolog : la base de règles

```
cousine(X, Y):-pere(Z, X), pere(W, Y), frere(Z, W), sexe(X, f),
sexe(Y, f).
cousine(X, Y):-mere(Z, X), mere(W, Y), soeur(Z, W), sexe(X, f)
sexe(Y, f).
```

Aperçu sur Prolog : les questions

:- pere(seydou, jeanne).
:- pere(seydou, salif).
:- pere(seydou, X).
:- mere(X, salif).
:- mere(X, michel).
:- pere(alain, X), pere(X, salif).

Aperçu sur Prolog : les commentaires sont :

- Pour une ligne jusqu'à la fin de la ligne : % commentaires
- Pour plusieurs lignes comme en c :/* les commenatires

Aperçu sur Prolog

En PROLOG, la règle p:-q1, q2, …, qn (clause HORN- à voir ultérieurement)

Signifie si **q1 et q2 et…et qn alors p**

Autrement dit q1 et q2 et…et qn=> p

- **Ou encore non (q1) ou non (q2) ou…non (qn) p**

Aperçu sur Prolog : Un programme est un ensemble de clause positives

Un programme Prolog est un ensemble de clauses non négatives=> et les clauses négatives (utilisation du prédicat prédéfini **fail**)?

Aperçu sur Prolog : fonctionnement de l'interprete
Exemples : l'ordre d'exécution des clauses différen

```
ancetre(X, Y):-parent(X,  Y).
ancetre(X, Y):-parent(X, Z), ancetre(Z, Y).

ancetre1(X, Y):-parent(X, Z), ancetre1(Z, Y).
ancetre1(X, Y):-parent(X,  Y).
```

Logique du premier ordre : définition de la logique du premier ordre

- Logique propositionnelle : réprésentée par la description faits
- Logique du 1er ordre : comme dans le langage naturel, le monde contient des objets : personnes, maisons, nombres couleurs, match de foot, guerres...

Logique du premier ordre : définition de la logique du premier ordre

- Relations :

→ relations unaires ou propriétés : rouge(a), arrondi(a), premier(a)...
→ relations n-aires : pere(alain, jeanne)

- Fonctions : une seule “valeur” pour une “entrée” donnée : f(a)...

Logique du premier ordre : éléments de formalisation en prolog

- **Constantes :**

  Une constante est une chaîne de caractère commençant par une minuscule, un nombre. Exemple : 8, seydou, paul, chaîne de cara entre “Bobo-dioulasso”.

- **Variables :** Elles sont en mahjuscule ou commençant par _ : X, A, _a, Une variable annonyme est désignée par _

Logique du premier ordre : éléments de formalisation en prolog

- **Symboles fonctionnels :** elles sont descrites comme des relations pren
  des listes de termes. Par exemple : f(x, y), f(a)

- **Un terme** : { constante, variable, symbole fonctionnel }

- **Un litteral : est un symbole de prédicats (une relation) :** Exemple -p
  Y)
  -mere(X, Y)

- **Une clause PROLOG est un ensemble de litteraux composés ou s**
  Exemples :

```
pere(alain, paul).  /* littéral simple*/
                    /*  littéraux composés */
grandpere(X, Y):-pere(X, Z),  pere(Z, Y).
grandpere(X,  Y):-pere(X, Z),  mere(Z, Y).
```

Logique du premier ordre (calcul des predicats
calcul des relations)

- Les expressions composées sont construites à partir des expressions atomiques et des connecteurs

- Exemples : ¬ E, E 1 ∧ E 2 , E 1 ∨ E 2 , E 1 ⇒ E 2 , E 1 2

- Exemple de prédicats : frere(john, richard) ∧ frere(richard john)
  En prolog, on aura : frere(john, richard), frere(richar john)

Logique du premier ordre (clauses de Horn)

- **Une clause de Horn : est une disjonction de littéraux do**
  maximum est positif
  Exemple : $(\neg A \vee \neg B \vee C)$ est une clause de Horn
  $(\neg C \vee A \vee B)$ n'est pas une clause de Horn

- Toute clause de Horn peut s'écrire sous la forme d'une implica
  prémisse = conjonction de littéraux positifs
  conclusion = littéral positif unique.
  Exemple $(\neg P(x) \vee \neg B \vee C) = ((P(x) \wedge B) \Rightarrow C)$

Logique du premier ordre (clauses de Horn)

- Clauses définies : clauses de Horn ayant exactement un littéral positif, **le littéral positif = tête**
- **Les littéraux négatifs** = corps de la clause
- **Fait** = clause sans littéraux négatifs

Logique du premier ordre (clauses de Horn-suite)

Programme Prolog : ensemble de faits et de règles qui décrivent un problème

Faits

P (...) . avec P un prédicat

Clauses de Horn, réduites à un littéral positif

pere(jean,paul).

mere(jean,marie).

Règles

Clauses de Horn complètes

P : − Q 1 , . . . , Q n .

Correspond en logique à Q 1 $\wedge$ . . . $\wedge$ Q n $\rightarrow$ P

P : tête de clause, Q 1 , . . . , Qn queue de clauses

Grandpere(X,Y) :- pere(X,Z), pere(Z,Y).

Logique du premier ordre et application en Prolog (clauses de Horn-suite)

Questions (ou liste de buts Qi, conclusion) ou demande de résolution :
Clauses de Horn sans littéral positif (clauses négatives)
Q1 , . . . , Qn .
:- pere(X,alain), Pere(alain,Y).

→ Exécuter un programme prolog= résoudre une liste de buts (une question) ou faire une preuve

Logique du premier ordre et application en Prolog (clauses de Horn-suite)

Un paquet de clauses est un ensemble de clauses ayant le même litteral positif de tête.

Exemple :

Paquet 1 :

- grandpere(X, Y):-pere(X, Z), pere(Z, Y).
- grandpere(X, Y):-pere(X, Z), mere(Z, Y).

Paquet 2 :

- grandmere(X, Y):-mere(X, Z), mere(Z, Y).
- grandmere(X, Y):-mere(X, Z), pere(Z, Y)?

Logique du premier ordre et application en Prolog (utilisation des prédicats)

Jean et paul habitent ensemble : habite(jean, paul)

habite(X, X):-true. //refexivité
habite(X, Y):-habite(X, Z), habite(Z, X). // transitivité
habite(X, Y):-habite(Y, X).//symétrie

X est une femme, si X possède un enfant Y.

femme(X):-possede(X,Y), enfant(Y).

Logique du premier ordre et application en Prolog (utilisation des prédicats prédéfinis write et read, tab)

- Le prédicat **write(X)** permet d'écrire le contenu de la variable X à l'écran.

Exemple : A la question

:- grandpere(X, Y),**write**(X), **write**(“ est le grand-père de “),**write**(Y), **nl**.

- Le mot clé nl, permet de retourner à la ligne suivante

Logique du premier ordre et application en Prolog (utilisation des prédicats write et read tab)

- **write(X)** : écrit sur le terminal, le contenu de la variable X.
- Exemple: write(a), or write("How are you?")

- **write_ln(X)** : écrit sur le terminal, le contenu de la variable X, suivie d'une nouvelle ligne.

- **tab**(X) : écrit sur le terminal, un X nombre de fois d'espace

Logique du premier ordre et application en Prolog (utilisation des prédicats prédéfinis write et read, tab)

- **read(X)** : lire sur le terminal des données pour la variable X.
- Exemple: afficher(X) :-write('quel est votre nom ?'), read(X), write('Hello le word'), tab(1), write(X).

Logique du premier ordre et application en Prolog (utilisation des prédicats prédéfinis assert)

- **assert(X)** : ajoute une clause ou un fait à la base de données.
- **asserta**(X) : ajoute au début de la liste des faits
- **assertz**(X): ajoute à la fin de la liste des faits
- Exemple: :-pere(mamadou, salif).
  assert(pere(alain, jean)).
  assert(pere(paul, issa)).
  assert((p:-q,r))
- ? listing(pere).

Logique du premier ordre et application en Prolog (utilisation des prédicats prédéfinis assert)

- Dans la base de règles faire :
- p(X, Y) :- write_ln (“entrez un nom”), read(X), write_ln(“entrez une couleur”), read(Y), assert(sac(X, Y)).
- ?sac(X, Y).

Logique du premier ordre et application en Prolog (utilisation des prédicats prédéfinis retract ou retractall)

- **retract**(X) : supprime une clause ou un fait de la base de donnés.
- **Retractall**(X): supprime des paquets de clauses X.
- Exemples : retract(pere(alain, robert)).
  retractall(pere(_, _)).

Logique du premier ordre et application en Prolog (utilisation des prédicats)

Traduction d'une fonction en escalier en Prolog

f(x)=0 si x<=2
f(x)=4 si x<=10
f(x)=5 sinon

On introduit un prédicat p(X, Y) <=> f(x)=y par exemple.

```
p(X, 0):-X<=2.
p(X, 4):-X>2, X<=10.
p(X, 5):-X>10.
```

Logique du premier ordre (prédicats prédefinis) : utilation de la coupe(cut) notéé “!” pour les paquets de clauses

La coupe permet d'arrêter la vérification des autres clauses : il provoque la suppression des choix possibles restants pour les littéraux allant de la tête de la clause courante incluse jusqu'à ce prédicat

```
p(X, 0):-X<=2, !.
p(X, 4):-X>2, X<=10, !.
p(X, 5):-X>10, !.
```

Logique du premier ordre (les prédicats prédéfinis) : la coupe !

max(A,B,B) :- A < B.

max(A,B,A).

?- max(3,4,M).

M = 4 ;

M = 3

max(A,B,B) :- A < B, !.

max(A,B,A).

?- max(3,4,M).

M = 4 ;

Logique du premier ordre (les prédicats prédéfinis) : la coupe !

```
if_then_else(P,Q,R) :-P, !, Q.
if_then_else(P,Q,R) :-R.
```

Logique du premier ordre (les prédicats prédéfinis) : la coupe !

```
factoriel (0, 1):-!.
factoriel (1, 1):-!.
factoriel(N,  R):-N>1, k is N-1, factoriel(K, Y),  R is  (Y*N).
```

Logique du premier ordre (les prédicats prédéfinis) : la coupe !

```
paire (X):-X>1, Y is mod(X, 2),  Y=:=0, !.
paire (0):-!.
Paire(1):-fail.
```

Logique du premier ordre (les prédicats prédéfinis) : la coupe ! : paire récursive

```
paire (X):-Y  is  X-2, paire(Y), !.
paire (0):-!.
```

Logique du premier ordre (les prédicats prédéfinis) : la négation avec fail.

Le mots clé fail aboutir une clause à false et changer de branche

nonp(X, Y) :- p(X, Y), !, fail.
nonp(X, Y).

Logique du premier ordre (prédicats prédéfin) : is

Le prédicat is permet d'évaluer les opérations arithmétiques :
Exemple :
X is 4+5. donne X=9
X is 4+5, Y is 5+4, X=Y.

is(X, +(*(3,7),8)) est similaire à X is 3*7+8

X is 6, Y is 8, Z is X+Y.

Logique du premier ordre (prédicats prédéfin) : is

Le prédicat is permet d'évaluer les opérations arithmétiques :

4 + 1 = 5
5 − 2 = 3
2 − 5 = − 3
4 / 2 = 2
2 ∗ 6 = 12
5 modulo 2 = 1

5 is 4 + 1.
3 is 5 - 2.
-3 is 2 - 5.
2 is 4 / 2.
12 is 2 * 6.
1 is mod(5,2).

Logique du premier ordre (prédicats prédéfin) : is

Le prédicat is permet d'évaluer les opérations arithmétiques :

somme (X, Y, Z):- Z is X+Y.

produit (X, Y, Z):- Z is X*Y.

Logique du premier ordre (prédicats prédéfinis) : comparaison

X< Y.
X=< Y.
X=:= Y.
X=\= Y.
X>= Y.
X> Y.

Applications à Prolog (les listes -récursivité- graphes)

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

chemin(X,Y):- arc(X,Z),!.

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

Applications à Prolog (les listes -récursivité- graphes)

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

chemin(X,Y):- arc(X,Y),!.

chemin(X,Y):- arc(X,Z), write( " arc de “),  write (X),  write(“ A “), write_ln (Y), chemin(Z,Y).
```

Applications à Prolog (les listes -récursivité- graphes)

```
villes(a,b, 12).
villes(b,c, 40).
villes(c,e, 49).
villes(c,d, 10).
villes(a,e,30).

route(X,Y, D):- villes(X,Y, D),!.

route(X,Y, D):-villes(X,Z, K), write( " villes de
“), write (X), write(“ A “), write (Y), write(“
distance “), write (K), routes(Z,Y, S), D is K+S.
```

Applications à Prolog (les listes -récursivité- graphes)

```
arc(a,b).
arc(b,c).
arc(c,e).
arc(c,d).
arc(a,e).
chemin(X,Y):- arc(X,Y),!.
chemin(X,Y):- arc(X,Z), chemin(Z,Y).
/*  longueur  d'un chemin */
chemin(X,Y, L):- arc(X,Y), L is 1, !.
chemin(X,Y,L):- arc(X,Z), chemin(Z,Y,L1),
L is L1 +1.
```

Applications à Prolog (les listes -récursivité-LISTE)

Tester les instructions suivantes :

?-[U,V] = [1,2,3]
no
?- [U,V] = [1,2]
– U = 1
– V= 2
?- [U|V] = [1,4,5]
– U= 1
– V= [4,5]
?- [U|V] = [6]
– U = 6
– V = []

Applications à Prolog (les listes -récursivité-LISTE)

Tester les instructions suivantes :

?-[U,V] = [7,8,9].
false
?- [U|V] = [a, “Bobo”,2].
– U = a
– V= [“Bobo”,2]
?- [U|V] = [1,[[7,9], 5].
– U= 1
– V= [7,9], 5]
?- [U|V] = [1, [5, 6]].
– U = 1
– V = [[5, 6]]

Applications à Prolog (les listes -récursivité-LISTE)

[T|Q] : T est la tête de la liste, Q est la queue.

la liste vide : []
[1,2,3,4] = [1| [2,3,4]]=[1| [2| [3,4]] = ...
= [1| [2| [3| [4| []]]]]

[E1| L1] =[E2 | L2] ssi E1 = E2 et L1 = L2

Applications à Prolog (les listes -récursivité-LISTE)

```
concater([],L,L).
concater([X|L1],L2,[X|L3]) :-concater(L1,L2,L3)
```

Applications à Prolog (les listes -récursivité-LISTE)

```
appartient(E,[E|_]).
appartient(E,[_|L]):-appartient(E,L).

longueur([ ],0).
longueur([_|L],N):-longueur(L,NB), N is
NB+1.
```

Applications à Prolog (les listes -récursivité-LISTE-exercices)

- **prédicat qui calcule le Max d'une liste.**

```
maxi([X|Y], X ):-maxi(Y,R), X>=R,!.
 maxi([X|Y], R ):-maxi(Y,R), X=<R,!.
maxi([X|[]],X ):-!.
```

- **Insertion tête**
  inserthead(L, X, [X|L]).

- **Insertion queue**
  insertqueue(L, X, [L|X]).

Applications à Prolog (les listes -récursivité- LISTE-exercices)

- **Suppressiontete**

```
suppressiontete([X|Y], Y).
```

- **Suppressionqueue**

```
suppressionqueue([X|[]], []):-!.
 suppressionqueue([X|Y],[X|K] ):- suppressionqueue(Y, K).
```

- **Sommeterme**

```
sommeterme([X|[]], X):-!.
sommeterme([X|Y],  S):-sommeterme(Y, K), S is K+X.
```

- **Produitterme**

```
produitterme([X|[]], X):-!.
produitterme([X|Y],  S):-produitterme(Y, K), S is K*X.
```

Applications à Prolog (les listes -récursivité-LISTE-exercices)

Somme de deux vecteurs

```
sommevecteur([], [], []):
sommevecteur([X|Y], [Q|W], [M|P] ):=M is  X+Q, sommevecteur( Y,[W], [P]).
```

Produit scalaire de deux vecteurs

```
produitvecteur([X|[]], [Y|[]], [K]): K is X * Y,!.
produitvecteur([X|Y], [Q|W], [M|P] ):=M is  X*Q, produitvecteur( Y,[W], [P]).
```

Produit par un scalaire d'un vecteur

```
produitscalairevecteur([X|[]], C, [K]):-K  is   X*C, !.
produitscalaire([X|Y], C, [M|P] ):=M is  X*C, produitscalaire( Y,C, P).
```

➔ **Il existe plusieur prédicats sur les listes (à analyser).**

Applications à Prolog(les autres prédicats prédéfinis)

P:-Q;W. (; désigne le ou)

P->F. (désigne implique)

Conclusion

- Logique des prédicats
- Clauses de Horn
- Fonctions prédéfinies
- Liste

Bibliographie

- L’art de Prolog, L. Sterling, E. Shapiro, Masson
- Programmer en Prolog, Clocksin, Mellosh, Eyrolles
- Clocksin, W.F., and Mellish C.S. Programming in Prolog: 4th edition. New York: Springer-Verlag. 1994.