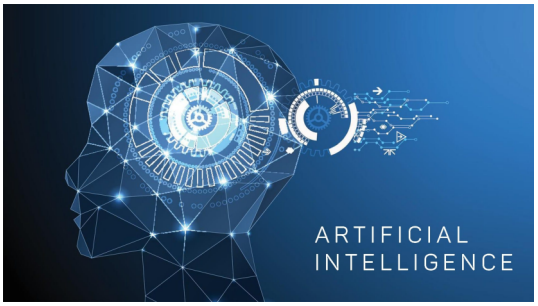
Chapitre 4 : Prolog



Pr. Mustapha BOURAHLA, Département d'Informatique, Université de M'Sila, Contact : mustapha.bourahla@univ-msila.dz

Janvier 2020
v 1.0

Table des matières

Introduction

Ce quatrième chapitre présente le langage Prolog comme langage de l'IA.

Le langage PROLOG

I

- Langage d'expression des connaissances fondé sur le langage des prédicats du premier ordre
- Programmation déclarative :
    - L'utilisateur définit une base de connaissances
    - L'interpréteur Prolog utilise cette base de connaissances pour répondre à des questions

Constantes et variables

- Constantes
    - Nombres : 12, 3.5
    - Atomes
        - Chaînes de caractères commençant par une minuscule
        - Chaînes de caractères entre " "
        - Liste vide []
- Variables
    - Chaînes de caractères commençant par une majuscule
    - Chaînes de caractères commençant par _
    - La variable « indéterminée » : _

TROIS SORTES DE CONNAISSANCES : FAITS , RÈGLES , QUESTIONS

- Faits : P(...). avec P un prédicat
    - pere(jean, paul).
    - pere(albert, jean).
    - Clause de Horn réduite à un littéral positif
- Règles : P(...) :- Q(...), ..., R(...).
    - papy(X,Y) :- pere(X,Z), pere(Z,Y).
    - Clause de Horn complète
- Questions : S(...), ..., T(...).
    - pere(jean,X), mere(annie,X).
    - Clause de Horn sans littéral positif

Graphe de résolution

II

programme.pl

```
pere(charlie, david).
pere(henri, charlie).
papy(X,Y) :- pere(X,Z), pere(Z,Y).
```

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

```
X=henri, Y=david
```

```
$swiprolog
Welcome to SWI-Prolog (Version 3.3.0)
Copyright (c) 1993-1999 University of Amsterdam.
All rights reserved.
For help, use ?- help(Topic). or ?-apropos(Word).
?- [programme].
% programme compiled 0.00 sec, 824 bytes
true.
?- listing.
pere(charlie, david).
pere(henri, charlie).
papy(X, Y) :-
     pere(X, Z),
     pere(Z, Y).
true.
```

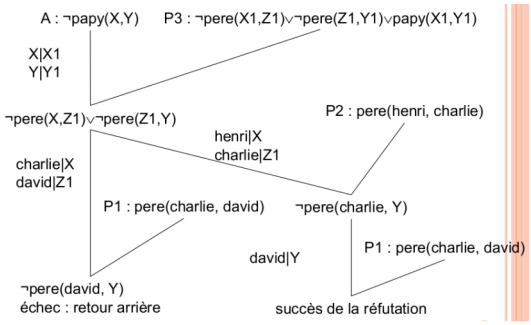
```
?- pere(charlie,david).
true.
?- pere(charlie,henri).
false.
?- pere(X,Y).
X = charlie
Y = david
true.
?- pere(X,Y).
X = charlie
Y = david ;
X = henri
Y = charlie
?- papy(x,y).
false.
?- papy(X,Y).
X = henri
Y = david
?- papy(henri,X).
X = david
true.
```

```
22 ?- halt.
23 $
```

Remarque : Ordre des réponses

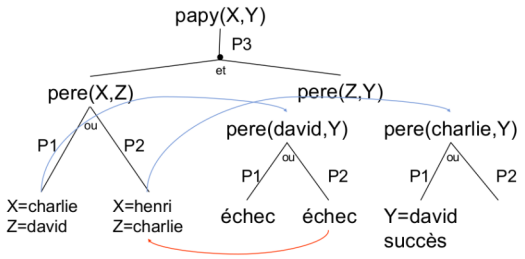
Prolog parcourt le paquet de clauses de haut en bas, chaque clause étant parcourue de gauche à droite

Graphe de résolution



Résolution par réfutation

Graphe de résolution



Graphe de résolution ET/OU

Les arithmétiques en Prolog

III

Symboles fonctionnels

Dans le fait age(homme(ahmed), 25), le symbole homme(ahmed) est un fonctionnel, ce n'est pas un prédicat.

Programme récursif

- Un programme récursif est un programme qui s'appelle lui-même
- Pour écrire un programme récursif, Il faut :

    1. Choisir sur quoi faire l'appel récursif
    2. Choisir comment passer du résultat de l'appel récursif au résultat que l'on cherche
    3. Choisir le(s) cas d'arrêt

```
fact(N,R) :- fact(N,1,R).
fact(1,R,R).
fact(N,I,R) :- N > 1,
    Nm1 is N-1,
    NewI is N*I,
    fact(Nm1,NewI,R).
```

```
?- trace, fact(3,N).
Call: (7) fact(3, _G234) ? creep
Call: (8) fact(3, 1, _G234) ? creep
Call: (9) 3>1 ? creep
Exit: (9) 3>1 ? creep
^ Call: (9) _G305 is 1*3 ? creep
^ Exit: (9) 3 is 1*3 ? creep
^ Call: (9) _G308 is 3-1 ? creep
^ Exit: (9) 2 is 3-1 ? creep
Call: (9) fact(2, 3, _G234) ? creep
Call: (10) 2>1 ? creep
Exit: (10) 2>1 ? creep
^ Call: (10) _G311 is 3*2 ? creep
^ Exit: (10) 6 is 3*2 ? creep
^ Call: (10) _G314 is 2-1 ? creep
^ Exit: (10) 1 is 2-1 ? creep
Call: (10) fact(1, 6, _G234) ? creep
Call: (11) 1>1 ? creep
Fail: (11) 1>1 ? creep
Redo: (10) fact(1, 6, _G234) ? creep
Exit: (10) fact(1, 6, 6) ? creep
N = 6
```

Bouclage : Cas 1

```
maries(jean, sophie).
maries(philippe, stephanie).
maries(A, B) :- maries(B, A).
```

```
?- maries(jean,sophie).
true.
?- maries(sophie,jean).
true.
?- maries(X,Y).
X = jean
Y = sophie ;
X = philippe
Y = stephanie ;
X = sophie
Y = jean ;
X = stephanie
Y = philippe ;
X = jean
Y = sophie ;
...
```

Bouclage : Cas 2

```
maries(jean, sophie).
maries(philippe, stephanie).
sont_maries(A, B) :- maries(A, B).
sont_maries(A, B) :- maries(B, A).
```

```
?- sont_maries(X,Y).
X = jean
Y = sophie ;
X = philippe
Y = stephanie ;
X = sophie
Y = jean ;
X = stephanie
Y = philippe ;
false.
?-
```

Les arithmétiques

- Comparaisons:
    - =:=, =\=, >, <, >=, =<
- Affectaction : is
    - ?- X is 3+2.
    - X=5
- Fonctions prédéfinies :
    - -, +, *, /, ^, mod, abs, min, max, sign, random, sqrt, sin, cos, tan, log, exp, ...

Comparaison et unification de termes

- Vérifications de type :
    - var, nonvar, integer, float, number, atom, string, ...
- Comparer deux termes :
    - T1==T2 réussit si T1 est identique à T2
    - T1\==T2 réussit si T1 n'est pas identique à T2
    - T1=T2 unifie T1 avec T2
    - T1\=T2 réussit si T1 n'est pas unifiable à T2

Les listes en Prolog

IV

- Liste vide : [ ]
- Cas général : [Tete|Queue]
    - [a,b,c] ≡ [a|[b|[c|[ ]]]]

Exemple

- [X|L] = [a,b,c] ⟶ X = a, L = [b,c]
- [X|L] = [a] ⟶ X = a, L = []
- [X|L] = [] ⟶ échec
- [X,Y] = [a,b,c] ⟶ échec
- [X,Y|L] = [a,b,c] ⟶ X = a, Y = b, L = [c]
- [X|L] = [X,Y|L2] ⟶ L = [Y|L2]

Somme des éléments

```
SOMME DES ÉLÉMENTS D'UNE LISTE DE NOMBRES
/* somme(L, S) L liste de nb donnée, S nb résultat */
somme([],0).
somme([X|L],N) :- somme(L,R), N is R+X.

?- somme([1,2,3,5],N).
N = 11
```

Variable indéterminée

```
/* ieme(L,I,X) L liste donnée, I entier donné, X elt res */
ieme([X|_],1,X).
ieme([_|L],I,R) :- I>1, Im1 is I-1, ieme(L,Im1,R).

?- ieme([a,b,c,d],2,N).
N = b ;
false.
```

Test ou génération

```
/* appart(X,L) X elt donné, L liste donnée */
appart(X,[X|_]).
appart(X,[_|L]) :- appart(X,L).

?- appart(a,[b,a,c]).
true.
```

```
?- appart(d,[b,a,c]).
false.
?- appart(X,[b,a,c]).
X = b ;
X = a ;
X = c ;
false.
?- trace, appart(X,[b,a,c]).
Call: (7) appart(_G284, [b, a, c]) ? creep
Exit: (7) appart(b, [b, a, c]) ? creep
X = b ;
Redo: (7) appart(_G284, [b, a, c]) ? creep
Call: (8) appart(_G284, [a,
Exit: (8) appart(a, [a, c]) ? creep
X = a ;
Redo: (8) appart(_G284, [a, c]) ? creep
Call: (9) appart(_G284, [c]) ? creep
Exit: (9) appart(c, [c]) ? creep
X = c ;
Redo: (9) appart(_G284, [c]) ? creep
Call: (10) appart(_G284, []) ? creep
Fail: (10) appart(_G284, []) ? creep
false.
```

LE PRÉDICAT MEMBER

- Le prédicat appart est prédéfini en Prolog
- Il est très utile

```
?- member(c,[a,z,e,c,r,t]).
true
?- member(X,[a,z,e,r,t]).
X = a ; X = z ; X = e ; X = r ; X = t.
?- member([3,V],[[4,a],[2,n],[3,f],[7,g]]).
V = f .
```

Utilisation du prédicat append

```
Append est le prédicat prédéfini pour la concaténation de listes
?- append([a,b,c],[d,e],L).
L = [a, b, c, d, e]
Il est complètement symétrique et peut donc être utilisé pour
Trouver le dernier élément d'une liste :
?- append(_,[X],[a,b,c,d]).
X = d
Couper une liste en sous-listes :
?- append(L1,[a|L2],[b,c,d,a,e,t]).
L1 = [b, c, d],
L2 = [e, t]
```

Exercices sur Prolog

V

Exercice : Programmation avec Prolog *[solution* n°1 p.14*]*

Écrire un programme Prolog pour le problème de remplissage des seaux.

Écrire un programme Prolog pour le problème de jeu de taquin à 9 tuiles.

Conclusion

Un aperçu sur le langage Prolog est présenté dans ce chapitre.

Solutions des exercices

> Solution n°1

Exercice p. 12

Écrire un programme Prolog pour le problème de remplissage des seaux.

Écrire un programme Prolog pour le problème de jeu de taquin à 9 tuiles.

Les programmes Prolog