UNIVERSITÉ TOULOUSE III PAUL SABATIER

Rédigé avec LaTeX
Version du 10 mars 2014

Programmation logique

Prolog

L3 Informatique
Semestre 6

Cours donné par Christine Maurel – Jean-Paul Arcangeli
Rédigé par Antoine de Roquemaurel

2014

Avant-propos

Même organisation que pour langage et automates. Cinq cours avec Christine Maurel et 5 cours avec Jean-Paul Arcangeli.

8 séances de TP, utilisation de Swi-Prolog[1].

1. http://www.swi-prolog.org

Table des matières

1 Introduction

Nouvelle façon de penser la programmation, nouveau paradigme venant la communauté de l'Intelligence Artificielle avec la formalisation d'un raisonnement logique : des hypothèses vers la conclusion, principe de résolution Robinson(1965). Implémentation du principe de Robinson en 1972.

Prolog par du principe qu'il faut **décrire** le problème plutôt que la solution

Le Prolog est un langage **normalisé** résultant du principe de résolution **non typé**.

Nous allons travailler avec de la logique donc des **prédicats** et des **relations**, résoudre et **unification**.

2 Prolog par l'exemple

Un programme Prolog est un ensemble ordonné d'énoncés (**clauses**) qui décrivent les
- Connaissances
- Propriétés
- Relations entre les objets du monde décrit.

On décrit ce monde avec
- Des assertions qui mettent en relation des informations : affirmation de faits
- Des règles qui mettent en relation des objets

2.1 Premier programme Prolog

```
1 /* biographie(enfant, sexe, ne_en, dcd_en, pere, mere). */
```

Commentaires /* */

biographie Nom de la relation : c'est un nom de prédicat à 6 argments : le nom de l'enfant, son sexe, son année de naissance, son année de décès, le nom de son père et celui de sa mère.

Arguments Les arguments sont entre parenthèses et séparés par des virgules.

La clause qui exprime un fat se termine par un point

```
1 biographie(louis13,m,1601,1643,henri4, truc)
```

2.2 Nouvelles relations : règles

```
1 /* pere(homme,enfant) */
```

homme est le père de enfant SI dans l'ensemble des clauses qui décrit la biographie il y a une relation biographie qui met en relation homme et enfant.

```
1 pere(P,E) :- biographie(E,_,_,_,P,_).
```

- On a des variables qui sont en majuscule : E,
- Des variables anonymes `_` (on se moque de leur nom dans ce cas)
- Exprimer la règle « a si b » en Prolog par la clause `a :- b`.
- a est la **tête** de clause (la conclusion) et b la **queue** de clause (hypothèse)

```
1 /* dureeDeVie(personne,age) */
```

age sera la durée de vie de personne SI dans l'ensemble de clauses qui décrit la biographie, il y a une relation biograhie qui met en relation son année de naissance et on année de décès, et si on fait leur différence entre ces 2 nombres.

```
dureeDeVie(P,A) :- biographie(P,_,N,D,_,_),A is D-N
```

– Exprimer la règle « a si b et c » en Prolog par la clause `a :- b, c`.
– a est la tête de clause et b,c la queue de clause.

Pour faire une opération arithmétique on utilise 'is' qui permet d'associer la valeur de `D-N` à A ; le A étant le nom de variable qui symbolise la durée de vie que l'on cherche.

On associe par unification, ce n'est pas une affectation, cela n'existe pas en Prolog.

2.3 Utilisation du programme

On va poser des questions (requêtes), interroger le programme, ce sera la conclusion d'un raisonnement logique décrit par les hypothèses que sont les faits et les règles contenus dans le programme.

Le système Prolog va résoudre cette requête : il va chercher à prouver, satisfaire la requête.

```
/* Est ce que Louis XIV est mort en 1715 */
?- biographie(loui14,_,_,1715,_,_).
true

/*Quel est la date de naissance de louis XIV*/
?- biographie(louis14,_,X,_,_,_).
X = 1638

/* Les personnes masculines */
?- biographie(E,m,_,_,_,_).
E = louis13 ;
E = louis14

/* Combien d'année louis XIV a-t-il survécu à son père */
?- biographie(louis14,_,_,X,P,_),biographie(P,_,_,Y,_,_).
R is X-Y
| X = 1715
P = louis13,
Y = 1643,
R = 72

/* */
biographie(X,_,_,Y,_,_à,Y>1650.
X = louis14
Y = 1715

/* durée de vie > 40 ans */
?- dureeDeVie(P,A),A>40
P = louis13,
A = 42;
P = elisabeth_france
A = 77
```

> R Prolog attend un ; afin de donner la suite des solutions.
> Il donnera toutes les solutions possibles.

2.3.1 Conditions

```
  a :- b,c. /* a si b et c */
a:- d. /* ou a si d */
a :-e,f,g. /* ou a si e et f et g */
```

2.4 Syntaxe

- Constante : commence par une minuscule
- Variable : commence par une majuscule
- Nombre
- Prédicat : `predicat(a1,...,an)` « Met en relation a1, . . . ,an »

> R Un programme Prolog est un ensemble ordonné de clauses : Faits (affirmation) ou Règles.

2.5 Autres exemples

2.5.1 Le graphe

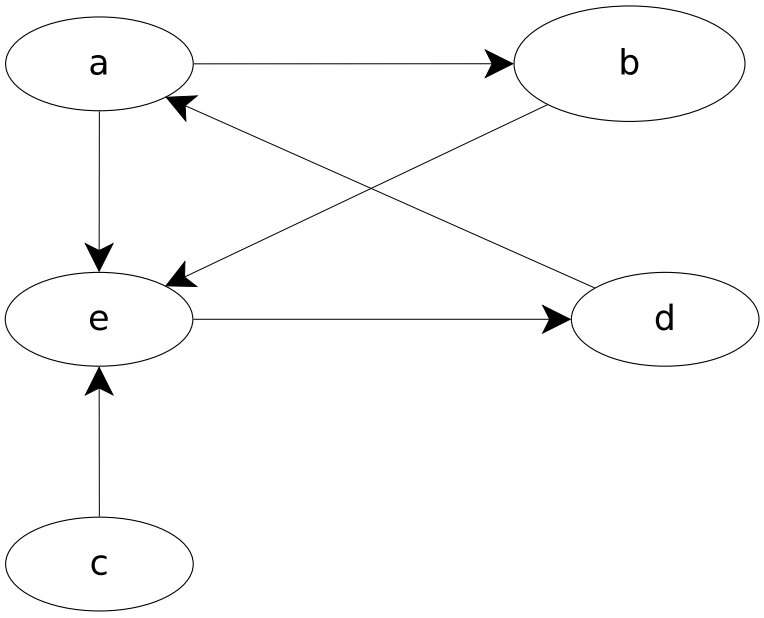

FIGURE 2.1 – Exemple de graphe

```
/* arc(source,destination) */
arc(a,b).
arc(a,e).
arc(b,e).
arc(c,e).
arc(e,d).
arc(d,a).

/* chemin(source,destination) */
chemin(S,D) :- arc(S,D).
chemin(S,D) :- arc(S,I),chemin(I,D).

/* y a-t-il un arc de b a e
arc(b,e).

/* Y a-t-il un arc qui arrive à d
arc(_,d).

/* Y a-t-il un arc qui arrrive à e */
arc(X,e).

/* Quels sont les sommets atteignables directement de d? */
arc(d,X)

/* Quels sont les sommets atteignables depuis d */
?- chemin(d,X)
X = a;
X = b;
X = e;
X = e;
X = d
X = a;
X = b;
X = e;
X = e;
X = d.  /* boucle infinie */
```

Notre définition du graphe implique qu'en cas de circuit, Prolog restera dans une boucle infinie.

2.5.2 Le crypthogramme

$S, E, N, D, M, O, R, Y \in \{0, \cdots, 9\}$ toutes différentes.

$$\begin{aligned} O + D + E &= R_1 \times 10 + Y \\ R_1 + N + R &= R_2 \times 10 + E \\ R_2 + E + 0 &= R_3 \times 10 + N \\ R_3 + S + M &= R_4 \times 10 + 0 \\ R_4 &= M \end{aligned}$$

$$\begin{array}{cccccc} & & \mathrm{S} & \mathrm{E} & \mathrm{N} & \mathrm{D} \\ + & & \mathrm{M} & \mathrm{O} & \mathrm{R} & \mathrm{E} \\ \hline \mathrm{M} & \mathrm{O} & \mathrm{N} & \mathrm{E} & \mathrm{Y} \end{array}$$

```
chiffre(0).
chiffre(1).
chiffre(2).
chiffre(3).
chiffre(4).
chiffre(5).
```

```
chiffre(6).
chiffre(7).
chiffre(8).
chiffre(9).

addpartielle(R,C1,C2,D,U) ↩
:- X is R+C1+C2, D is X/10, U i X mod 10.
```

3 Sémantique

.

3.1 Formaliser les fais et règles en logique et en prolog

- H1 : Jean est heureux si un de ses employés travaille
- H2 : Ceux qui s'amusent sont heureux
- H3 : Pierre et Paul sont des employés de Jean
- H4 : Pierre Travaille et Paul s'amuse

Logique	Prolog
3 constantes : Jean, Pierre, Paul	3 constantes jean, pierre, pa
Prédicats : – heureux(x), x est heureux – travaille(x) : x travaille – employé(x,y) x l'emloyé de y – amuse(x) : x s'amuse	– heureux(X) – employé(X,Y) – travaille(X) – amuse(X)
$H1 : (\exists X(employe(jean, X) \wedge traaille(X)) \rightarrow heureux(jean))$	`heureux(jean) :- employe(X,jean),t`
$H2 : (\exists X(amuse(jean, X) \rightarrow heureux(X)))$	`employe(paul,jean).employe(pier`
$H3 : employe(pierre, jean) \wedge employe(paul, jean)$	`amuse(paul).travaille(pier`

3.2 Résoudre une question : Arbre de recherche

4 Gestion du contrôle

4.1 Introduction par l'exemple

Écrire le prédicat eliminer tel que eliminer(X,L,R) est vrai si R est la liste L ou toutes les occurences de x ont été supprimées.

```
eliminer(X,[], []). %1
eliminer(X, [X|L], R) :- eliminer(X, L, R). %2
eliminer(X, [Y|L], [Y|R]) :- X \== Y, eliminer(X, L, R). %3
% 1 Exclusivité avec 2 et 3
% 2 et 3 pas non exclusives -> Si 2 ne pas faire 3 -> X \== Y
```

Écrire le prédicat maximum `maximum(L, M)` est vrai si M est le plus grand élément de la liste L.

```
maximum([M], M). %1
maximum([X|L], X) :- maximum(L, P), X > P. %2 maximum calculé deux
maximum([X|L], P) :- maximum(L, P), X < P. %3 fois

%%%
maximum([M], M). %1
maximum([X|L], M) :- maximum(L, P), sup(X,P,M). %2
sup(X, P, X) :- X >= P. %3
sup(X, P, P) :- X < P. %4
% 3 et 4 exclusives grace aux conditions
```

Écrire le prédicat intersection `intersection(E1, E2, I)` est vrai si I est l'intersection des deux ensembles E1 et E2. Un ensemble est représenté par une liste où chaque élément n'apparait qu'une seule fois.

```
intersection([], E2, []). %1
intersection([X|E1], E2, [X|I]) :- member(X, E2), intersection(E1, E2, I). %2
intersection([X|E1], E2, I) :- not(member(X,E2)), intersection(E1, E2, I). %3

% 1 exclusives avec 2 et 3
% 2 et 3 non exclusives, si X appartient E2
% rajouter not(member) dans clause 3...
```

4.2 Définition, utilisation du CUT

Prédicat « coupe choix » dit *cut* noté !.

Definition 4.1 La résolution d'un *cut* supprime tous les points de choix encore possible entre le littéral qui a provoqué son apparition dans la résolvante et le *cut* lui même. Tous les points de choix restent possible avant L et après *cut*.

```
eliminer(X,[], []). %1
eliminer(X, [X|L], R) :- !, eliminer(X, L, R). %2
eliminer(X, [Y|L], [Y|R]) :- eliminer(X, L, R). %3
```

Listing 4.1 – eliminer avec *Cut*

```
maximum([M], M) :- !. % Cut defficacité : non conseillé
maximum([X|L], M) :- maximum(L, P), sup(X,P,M). %2
sup(X, P, X) :- X >= P, !. % Si X >= P alors clause 3 mais pas 4
sup(X, P, P).
```

Listing 4.2 – maximum avec *Cut*

```
intersection([], E2, []). %1
intersection([X|E1], E2, [X|I]) :- member(X, E2), !, intersection(E1, E2, I). %2
intersection([X|E1], E2, I) :- intersection(E1, E2, I). %3
```

Listing 4.3 – intersection avec *Cut*

A Liste des codes sources