Spécifications formelles – Programmation logique et Prolog

cours 1

Lionel Blatter lionel.blatter@cea.fr
avec les slides de Allan Blanchard allan.blanchard@cea.fr
et de Guillaume Petiot guillaume.petiot@cea.fr

CEA, LIST, LSL

2016-2017

Programmation Logique

> *A logic program is a set of axioms, or rules, defining relations between objects. A computation of a logic program is a deduction of consequences of the program. A program defines a set of consequences, which is its meaning. The art of logic programming is constructing concise and elegant programs that have the desired meaning.*

(The Art of Prolog)

Logique propositionnelle

- propositions atomiques : symboles propositionnels (P, Q, ...) qui peuvent être vrais ou faux
- propositions complexes : compositions de propositions atomiques par des connecteurs logiques ($\neg$, $\wedge$, $\vee$, $\implies$, $\iff$)

Table de vérité

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\bot$
$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$

- $\neg P$ est vrai ssi P est faux
- $P \wedge Q$ est vrai ssi P est vrai et Q est vrai
- $P \vee Q$ est vrai ssi P est vrai ou Q est vrai
- $P \implies Q$ est vrai ssi P est faux ou Q est vrai
  $P \implies Q$ est faux ssi P est vrai et Q est faux
- $P \iff Q$ est vrai ssi $P \implies Q$ est vrai et $Q \implies P$ est vrai

Logique des prédicats du 1er ordre

- variables
- constantes
- prédicats :
  propriété portant sur des objets, qui peut être vraie ou fausse
- connecteurs logiques ($\neg$, $\wedge$, $\vee$, $\implies$, $\iff$)
- égalité ($=$)
- quantificateurs ($\forall$, $\exists$)

Quantificateurs

Quantificateur universel $\forall$

- $\forall x.P(x)$: vrai si $P(x)$ est vrai pour tous les x
- $\forall x.P(x) \implies Q(x)$: pour tout x, si $P(x)$ est vrai alors $Q(x)$ est vrai

Quantificateur existentiel $\exists$

- $\exists x.P(x)$: vrai si $P(x)$ est vrai pour (au moins) un x
- $\exists x.P(x) \wedge Q(x)$: il existe un x tel que $P(x)$ est vrai et $Q(x)$ est vrai

Introduction à la programmation logique

Programmation impérative (C, C++, etc.)

- définition d'instructions
- comment résoudre un problème

Programmation logique (Prolog, etc.)

- définition d'une base de connaissance (faits, règles) et de buts
- style déclaratif
- on décrit le problème mais pas comment le résoudre
- l'interpréteur Prolog utilise la base de connaissances pour résoudre les buts

Prolog

Histoire

- Premier interpréteur Prolog réalisé par Alain Colmerauer et Philippe Roussel (1972)

Application

- intelligence artificielle
- traitement linguistique: interface homme/machine de l'ISS (NASA)
- solver de contraint
- Application web
- ...

Implémentation

- SWI Prolog (existe aussi pour browser : http://swish.swi-prolog.org)
- GNU Prolog
- ECLIPSe
- ...

Éléments du langage : les termes

- termes de base (ou termes atomiques)
  - variables : objet dont le nom commence par une majuscule ou _
  - constantes :
    - nombres : entier ou flottants,
    - atomes : objet dont le nom commence par une minuscule,
    - chaînes de caractères
- termes composés

$$\underbrace{h}_{\text{symbole de foncteur}}\underbrace{(\overbrace{18}^{\text{constante}},\overbrace{f(g,\text{'}Twenty\text{'},X)}^{\text{terme composé}},\overbrace{X}^{\text{variable}})}_{\text{arguments}}$$

À propos des variables

- n'importe quelle variable peut représenter un nombre, une chaîne, une liste, un arbre, etc.
- pendant une exécution, les variables sont **contraintes**, plutôt qu'affectées et mises à jour

Structure d'un programme Prolog

- **Programme** : suite de procédures
  - **Procédure** : suite de **clauses** du même prédicat
    - **Fait** :

$$\overbrace{\underbrace{p}_{\text{nom de prédicat}} \; (\underbrace{a, b}_{\text{arguments}})}^{\text{but atomique}} .$$

    - **Règle** :

$$\overbrace{\underbrace{p(b, Y)}_{\text{but atomique}}}^{\text{tête}} \; :- \; \overbrace{\underbrace{q(Y)}_{\text{but atomique}} \; , r(Y, c)}^{\text{corps}} .$$

toutes les occurrences de Y représentent le même objet

Interprétation d'un **Fait**

$$\underbrace{p(\overbrace{\ 2\ }^{2}, \overbrace{\ z\ }^{z})}_{p(2,z)}. \qquad \underbrace{p(\overbrace{\ a\ }^{a}, \overbrace{f(b)}^{f(b)}, Y)}_{\forall Y.p(a,f(b),Y)}.$$

```
pere(jean, jacques).
homme(jean).
age(jean, 30).
pere(X, odin).
```

Interprétation d'une **Règle**

$$\underbrace{p \ :- \ q}_{q \Longrightarrow p}. \qquad \underbrace{p(a,b) \ :- \ q(a,f(c)),r(d)}_{q(a,f(c)) \wedge r(d) \Longrightarrow p(a,b)}.$$

$$\underbrace{p(X) \ :- \ q}_{\forall X.(q \Longrightarrow p(X))} \ . \qquad \underbrace{p(X) \ :- \ q(X)}_{\forall X.(q(X) \Longrightarrow p(X))} \ . \qquad \underbrace{p(a) \ :- \ q(X)}_{(\exists X.q(X)) \Longrightarrow p(a)} \ .$$

De manière générale :

$$\underbrace{H \ :- \ B_1, \ldots, B_n}_{\forall V_1,\ldots,V_k.(B_1' \wedge \ldots \wedge B_n' \Longrightarrow H')}$$

Parallèle avec les clauses de Horn

$$
\begin{aligned}
B'_1 \land \ldots \land B'_n \implies H' &\equiv \neg(B'_1 \land \ldots \land B'_n) \lor H' \\
&\equiv \underbrace{\underbrace{\neg B'_1 \lor \ldots \lor \neg B'_n}_{\text{littéraux négatifs}} \lor \underbrace{H'}_{\text{littéral positif}}}_{\text{clause de Horn}}
\end{aligned}
$$

Exemples de règles I

```
a :- b,c,d.
b.
c.
d :- e.
e.
```

Exemples de règles II

```
parent(X, Y) :- pere(X, Y).
parent(X, Y) :- mere(X, Y).
```

$$\forall X, Y.(pere(X, Y) \implies parent(X, Y))$$

$$\forall X, Y.(mere(X, Y) \implies parent(X, Y))$$

```
parent(X,Y) :- pere(X,Y) ; mere(X,Y).
```

$$\forall X, Y.((pere(X, Y) \vee mere(X, Y)) \implies parent(X, Y))$$

Exemples de règles III

```
grandpere(X, Y) :- parent(X, P), pere(P, Y).
```

$$\forall X, Y.((\exists P, \mathit{parent}(X, P) \wedge \mathit{pere}(P, Y)) \implies \mathit{grandpere}(X, Y))$$

Exécution d'un programme Prolog

- **yes** ou **no** si la question est une conséquence logique ou pas.
- L'ensemble des valeurs des variables pour laquelle la question est une conséquence logique.

```
?- pere(marie, paul).
  %% Paul est-il le pere de Marie ?
?- pere(jean,X).
  %% Quel est le pere de Jean ?
?- pere(X,tom).
  %% Quels sont les enfants de Tom ?
?- pere(X,Y).
  %% Qui est le pere de qui ?
```

Algorithme pour but simple

- **Input** : A ground goal G and a program P.
- **Output** : *yes* if G is a logical consequence of P, *no* otherwise.
- **Algorithme** :

begin
  Initialize the resolvent to G.
  while *the resolvent is not empty* **do**
    Choose a goal A from the resolvent.
    Choose a ground instance of clause $A' :- B_1, ..., B_n$ from P such that A and A' are identical.
    if *no such goal and clause exist* **then**
      Exit the while loop
    end
    Replace A by $B_1, ..., B_n$ in the resolvent
  end
  if *the resolvent is empty* **then**
    output *yes*
  end
  else
    output *no*
  end
end

Exemple de trace

Programme P :

```
homme(jean).
homme(jacques).
pere(jean,jacques).
fils(X,Y) :- pere(Y,X),homme(X).
```

Exemple de trace

begin

Input : *fils(jacques, jean)* ? and program *P*

Resolvent = *fils(jacques, jean)*

Resolvent is not *empty*

begin

choose *fils(jacques,jean)* (the only choice)

choose *fils(jacques,jean) :- pere(jean,jacques), homme(jacques)*

replace *fils(jacques,jean)* by *pere(jean,jacques), homme(jacques)*

end

new resolvent is *pere(jean,jacques), homme(jacques)*

Resolvent is not empty

begin

choose *pere(jean,jacques)*

choose *pere(jean,jacques)*

replace *pere(jean,jacques)* by *empty*

end

new resolvent is *homme(jacques)*

Resolvent is not empty

begin

choose *homme(jacques)*

choose *homme(jacques)*

replace *pere(jean,jacques)* by *empty*

end

new resolvent is *empty*

Output: *yes*

end

Instanciation et Unification

- L'instanciation des variables est basée sur le processus d'unification.
- Si Prolog essaie de vérifier la requête: - a (X). et rencontre la règle a (a). dans la base de données, il unifiera la variable X avec la constante a.
- Puisque a (a) est vrai, Prolog retournera X = a comme solution.

```
a(a)
?- a(X)
unification: X = a
```

Unification – exemple 1

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(a), Y, Z))).$$

Unification – exemple 1

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(a), Y, Z))).$$

$$\{Z = h(X), \underline{f(X, b, Z) = f(g(a), Y, Z)}\}$$

Unification – exemple 1

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(a), Y, Z))).$$

$$\{Z = h(X), \underline{f(X, b, Z) = f(g(a), Y, Z)}\}$$

$$\{Z = h(X), X = g(a), \underline{b = Y}, Z = Z\}$$

Unification – exemple 1

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(a), Y, Z))).$$

$$\{Z = h(X), \underline{f(X, b, Z) = f(g(a), Y, Z)}\}$$

$$\{Z = h(X), X = g(a), \underline{b = Y}, Z = Z\}$$

$$\{Z = h(X), X = g(a), Y = b, \underline{Z = Z}\}$$

Unification – exemple 1

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(a), Y, Z))).$$

$$\{Z = h(X), \underline{f(X, b, Z) = f(g(a), Y, Z)}\}$$

$$\{Z = h(X), X = g(a), \underline{b = Y}, Z = Z\}$$

$$\{Z = h(X), X = g(a), Y = b, \underline{Z = Z}\}$$

$$\{Z = h(X), \underline{X = g(a)}, Y = b\}$$

Unification – exemple 1

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(a), Y, Z))).$$

$$\{Z = h(X), \underline{f(X, b, Z) = f(g(a), Y, Z)}\}$$

$$\{Z = h(X), X = g(a), \underline{b = Y}, Z = Z\}$$

$$\{Z = h(X), X = g(a), Y = b, \underline{Z = Z}\}$$

$$\{Z = h(X), \underline{X = g(a)}, Y = b\}$$

$$\underbrace{\{Z = h(g(a)), X = g(a), Y = b\}}_{\text{unificateur}}$$

Unification – exemple 2

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(Z), Y, Z))).$$

Unification – exemple 2

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(Z), Y, Z))).$$

$$\{Z = h(X), \underline{f(X, b, Z) = f(g(Z), Y, Z)}\}$$

Unification – exemple 2

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(Z), Y, Z))).$$

$$\{Z = h(X), \underline{f(X, b, Z) = f(g(Z), Y, Z)}\}$$

$$\{Z = h(X), X = g(Z), \underline{b = Y}, Z = Z\}$$

Unification – exemple 2

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(Z), Y, Z))).$$

$$\{Z = h(X), \underline{f(X, b, Z) = f(g(Z), Y, Z)}\}$$

$$\{Z = h(X), X = g(Z), \underline{b = Y}, Z = Z\}$$

$$\{Z = h(X), X = g(Z), Y = b, \underline{Z = Z}\}$$

Unification – exemple 2

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(Z), Y, Z))).$$

$$\{Z = h(X), \underline{f(X, b, Z) = f(g(Z), Y, Z)}\}$$

$$\{Z = h(X), X = g(Z), \underline{b = Y}, Z = Z\}$$

$$\{Z = h(X), X = g(Z), Y = b, \underline{Z = Z}\}$$

$$\{Z = h(X), \underline{X = g(Z)}, Y = b\}$$

Unification – exemple 2

$$p(k(Z, f(X, b, Z))) = p(k(h(X), f(g(Z), Y, Z))).$$

$$\{Z = h(X), \underline{f(X, b, Z) = f(g(Z), Y, Z)}\}$$

$$\{Z = h(X), X = g(Z), \underline{b = Y}, Z = Z\}$$

$$\{Z = h(X), X = g(Z), Y = b, \underline{Z = Z}\}$$

$$\{Z = h(X), \underline{X = g(Z)}, Y = b\}$$

$$\underbrace{\{Z = h(g(Z)), X = g(Z), Y = b\}}_{\text{pas un unificateur, les 2 termes ne sont pas unifiables}}$$

Unification – exemple 3 I

```
nul(K) :- egal(K,0).
egal(X,X).

?- egal(C,4).
C = 4.
```

- { egal(C,4)= egal(X_0,X_0) }
- { C = X_0, X_0 = 4 }
- { C = 4 }
- la requête a une solution

Unification – exemple 3 II

```
nul(K) :- egal(K,0).
egal(X,X).

?- nul(4).
false.
```

- { `nul(4)= nul(K_0)` }
- { `nul(4)= nul(K_0)`, `egal(K_0,0)= egal(X_0,X_0)` }
- { `K_0 = 4`, `K_0 = X_0`, `X_0 = 0` }
- la requête n'a pas de solution

Backtracking I

```
parent(X,Y) :- pere(X,Y).
parent(X,Y) :- mere(X,Y).
mere(anna, sylvie).

?- parent(_,sylvie).
true.
```

- les exécutions comportent des **points de choix**, permettant de prendre des chemins d'exécution alternatifs
- si les contraintes sont **insatisfiables**, le programme **"backtrack"** jusqu'au dernier point de choix et explore un autre chemin
- le backtracking permet aussi d'obtenir des solutions supplémentaires

Backtracking II

```
parent(X,Y) :- pere(X,Y).
parent(X,Y) :- mere(X,Y).
mere(anna, sylvie).

?- parent(_,sylvie).
true.
```

`parent(_,sylvie)= parent(X0,Y0)`

{ `X0 = _`, `Y0 = sylvie` } (contraintes mémorisées)

- 1er choix : `pere(X0,Y0)`
  - { `X0 = _`, `Y0 = sylvie`, `pere(X0,Y0)= ??` }
  - pas de solution, on backtrack
- 2nd choix : `mere(X0,Y0)`
  - { `X0 = _`, `Y0 = sylvie`, `mere(anna,sylvie)= mere(X0,Y0)` }
  - { `X0 = _`, `Y0 = sylvie`, `X0 = anna`, `Y0 = sylvie` }
  - { `Y0 = sylvie` } une solution

Listes

- entre crochets
- éléments séparés par des virgules
- peut contenir tous types de termes

```
[], [x1,x2,x3], [x1,[toto,"abab"]],
[Head|Tail],[A,B,C|R], [A,B|[C,D]]
```

Prédicats utiles :

```
append(L1,L2,Res). %% concatenation
member(Elt,L). %% appartenance
```

Exercices sur les listes – member

Recodons `member(Elt,L)`

Exercices sur les listes – member

Recodons `member(Elt,L)`

$$\forall X, member(X, [X|\_])$$

$$\forall X, T.member(X, T) \implies member(X, [\_|T])$$

Exercices sur les listes – member

Recodons `member(Elt,L)`

$$\forall X, member(X, [X|\_])$$

$$\forall X, T.member(X, T) \implies member(X, [\_|T])$$

```
my_member(X,[X|_]).
my_member(X,[_|T]):-my_member(X,T).
```

Exercices sur les listes – append

Recodons `append(L1,L2,Res)`

Exercices sur les listes – append

Recodons `append(L1,L2,Res)`

$$\forall X. append([], X, X)$$
$$\forall X, H, T, R.\ append(T, X, R) \implies append([H|T], X, [H|R])$$

Exercices sur les listes – append

Recodons `append(L1,L2,Res)`

$$\forall X. append([], X, X)$$

$$\forall X, H, T, R.\ append(T, X, R) \implies append([H|T], X, [H|R])$$

```
my_append([],X,X).
my_append([H|T],X,[H|R]):-my_append(T,X,R).
```

D'autres exemples avec listes

- compter les occurrences
- tri par sélection
- tri par insertion
- tri fusion
- ...

Définition d'un prédicat : questions à se poser

- Comment vais-je l'utiliser ?
- Quelles sont les données ?
- Quels sont les résultats ?
- Peut-il y avoir plusieurs solutions ?
  - si on veut une seule solution, il faut faire des cas exclusifs

Bibliographie

- K.R. Apt and M. G. Wallace, 2007, **Constraint Logic Programming using ECLiPSe**, Cambridge University Press
- J. Robinson, 1965, **A machine-oriented logic based on the resolution principle**, J. ACM, 12, pp. 23-41.
- A. Martelli and U. Montanari, 1982, **An efficient unnification algorithm**, ACM Transactions on Programming Languages and Systems, 4, pp. 258-282.
- R. Kowalski, 1974, **Predicate logic as a programming language**, IFIP'74, pp. 569-574.
- L. Sterling and E. Shapiro, 1994, **The Art of Prolog**, The MIT Press, Cambridge, Massachusetts, 2nd edn.