

Spécifications formelles – Programmation logique et Prolog

cours 1

Guillaume Petiot

`guillaume.petiot@cea.fr`

CEA, LIST, LSL

2014-2015

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	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
\perp	\perp	\top	\perp	\perp	\top	\top
\perp	\top	\top	\perp	\top	\top	\perp
\top	\perp	\perp	\perp	\top	\perp	\perp
\top	\top	\perp	\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
- ▶ $P \Rightarrow Q$ est vrai ssi P est faux ou Q est vrai
 $P \Rightarrow Q$ est faux ssi P est vrai et Q est faux
- ▶ $P \Leftrightarrow Q$ est vrai ssi $P \Rightarrow Q$ est vrai et $Q \Rightarrow P$ 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.)

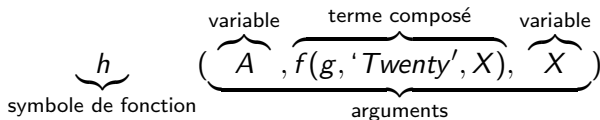
- ▶ définition d'instructions
- ▶ comment résoudre un problème

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

Éléments du langage : les termes

- ▶ termes de base
 - ▶ variables : objet dont le nom commence par une majuscule ou _
 - ▶ constantes :
 - ▶ nombres
 - ▶ atomes : objet dont le nom commence par une minuscule, chaînes littérales entre ''
- ▶ termes composés



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** :

$$\overbrace{p(a, b)}^{\text{but atomique}}.$$

nom de prédicat arguments

- ▶ **Règle** :

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

but atomique but atomique

toutes les occurrences de Y représentent le même objet

Interprétation d'un fait

$$\underbrace{p(\overbrace{2}^2, \overbrace{z}^z})}_{p(2,z)}.$$

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

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

Interprétation d'une règle

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

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

De manière générale :

$$\underbrace{H :- B_1, \dots, B_n}_{\forall V_1, \dots, V_k.(B'_1 \wedge \dots \wedge B'_n \Rightarrow H')}$$

Parallèle avec les clauses de Horn

$$\begin{aligned} B'_1 \wedge \dots \wedge B'_n \implies H' &\equiv \neg(B'_1 \wedge \dots \wedge B'_n) \vee H' \\ &\equiv \underbrace{\neg B'_1 \vee \dots \vee \neg B'_n}_{\text{littéraux négatifs}} \vee \underbrace{H'}_{\text{littéral positif}} \\ &\underbrace{\hspace{10em}}_{\text{clause de Horn}} \end{aligned}$$

D'autres exemples de règles I

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

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

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

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

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

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

D'autres exemples de règles II

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

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

Requêtes

```
?- 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 ?
```

Tentative d'**unification** du but avec les faits et les règles.

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

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

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

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

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

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

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

unificateur

Unification – exemple 2

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

Unification – exemple 2

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

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

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

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

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

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

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

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

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

pas un unificateur, les 2 termes ne sont pas unifiables

Requêtes 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

Requêtes 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(El, L). %% appartenance
```


Exercices sur les listes – member

Recodons `member` (`Elt`, `L`)

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

```
my_member (X, [X|_]) .
```

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

```
    my_member (X, T) .
```

Exercices sur les listes – append I

Recodons `append(L1, L2, Res)`

Exercices sur les listes – append II

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

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 unification 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.