Programmation Logique

Introduction

RAYAR Frédéric
frederic.rayar@univ-tours.fr

Année 2011-2012

Sommaire

Sommaire

Notions vues avec Mr. Ragot

Intelligence Artificielle

- Agents intelligents
- Systèmes Experts (*e.g.* MYCIN)
- Logique des propositions
- Systèmes à base de règles

Notions vues avec Mr. Ragot

Prédicats

- Logique des prédicats
- Substitution et Unification
- Arbre de recherche avec backtrack
- Definitions :
  - clause, clauses de Horn
  - mise sous forme normale conjonctive
  - inférence par résolution
  - notion de but
  - *etc ...*

Contenu du module

Introduction

- Paradigme de programmation
- Historique

Des rappels ...

- Logique des propositions
- Logique des prédicats

Contenu du module

Le language PROLOG

- Vocabulaire utilisé
- Syntaxe du langage
- Concepts PROLOG (contrôle,meta programmation, *etc.*)

Aller plus loin ...

- PROLOG avancé (XPCE, fichiers)
- Un cas d'utilisation

Organisation du module

Enseignement

- 8h de Cours/TD
- 8h de TD/TP
- Sur 3 semaines

Examen

- Assiduité
- Un examen terminal

Sommaire

Programmation impérative

Programmation impérative

- "Tout" est **instruction**
  $\Rightarrow$ *On dit à la machine ce qu'elle doit faire et quand elle doit le faire*
  $\Rightarrow$ Contrôle total du flot d'exécution
  $\Rightarrow$ Syntaxe souvent lourde
- Langages : Pascal, C, *etc.*

Exemple

Fibonacci (C)

```
int fibonacci(int iterations)
{
    int first = 0; int second = 1;
    for (int i = 0; i < iterations; ++i) {
        int sum = first + second;
        first = second;
        second = sum;
    }
    return first;
}
printf("Fibonacci(10) = %d\n", fibonacci(10));
```

Programmation fonctionnelle

Programmation fonctionnelle

- "Tout" est **fonction**
  $\Rightarrow$ Syntaxe dépouillée
  $\Rightarrow$ Forte base mathématique (λ-calcul)
  $\Rightarrow$ Utilisation intensive de la récursivité
- Langages : Lisp, Haskell, Caml, *etc.*

Exemple

Fibonacci (Haskell)

```
-- describe an infinite list based on the recurrence
-- relation for Fibonacci numbers
fibRecurrence first second = first : fibRecurrence second (first + second)

-- describe fibonacci list as fibRecurrence with initial values 0 and 1
fibonacci = fibRecurrence 0 1

-- describe action to print the 10th element of the fibonacci list
main = print (fibonacci !! 10)
```

Programmation orienté objet

Programmation orienté objet

- "Tout" est **objet**
  ⇒ Représente un concept, une idée ou toute entité du monde physique
  ⇒ Définition des structures internes, relations et communication entre ces objets
- Langages :
  - fonctionnel : CLOS (Common Lisp object System)
  - impératifs : C++, Java
  - les deux : python, OCaml

Exemple

Fibonacci (Java)

```
// Computes and prints the first N Fibonacci numbers.
public class Fibonacci {

    public static long fib(int n) {
        if (n <= 1) return n;
        else return fib(n-1) + fib(n-2);
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        for (int i = 1; i <= N; i++)
            System.out.println(i + ": " + fib(i));
    }

}
```

Programmation déclarative

Programmation déclarative

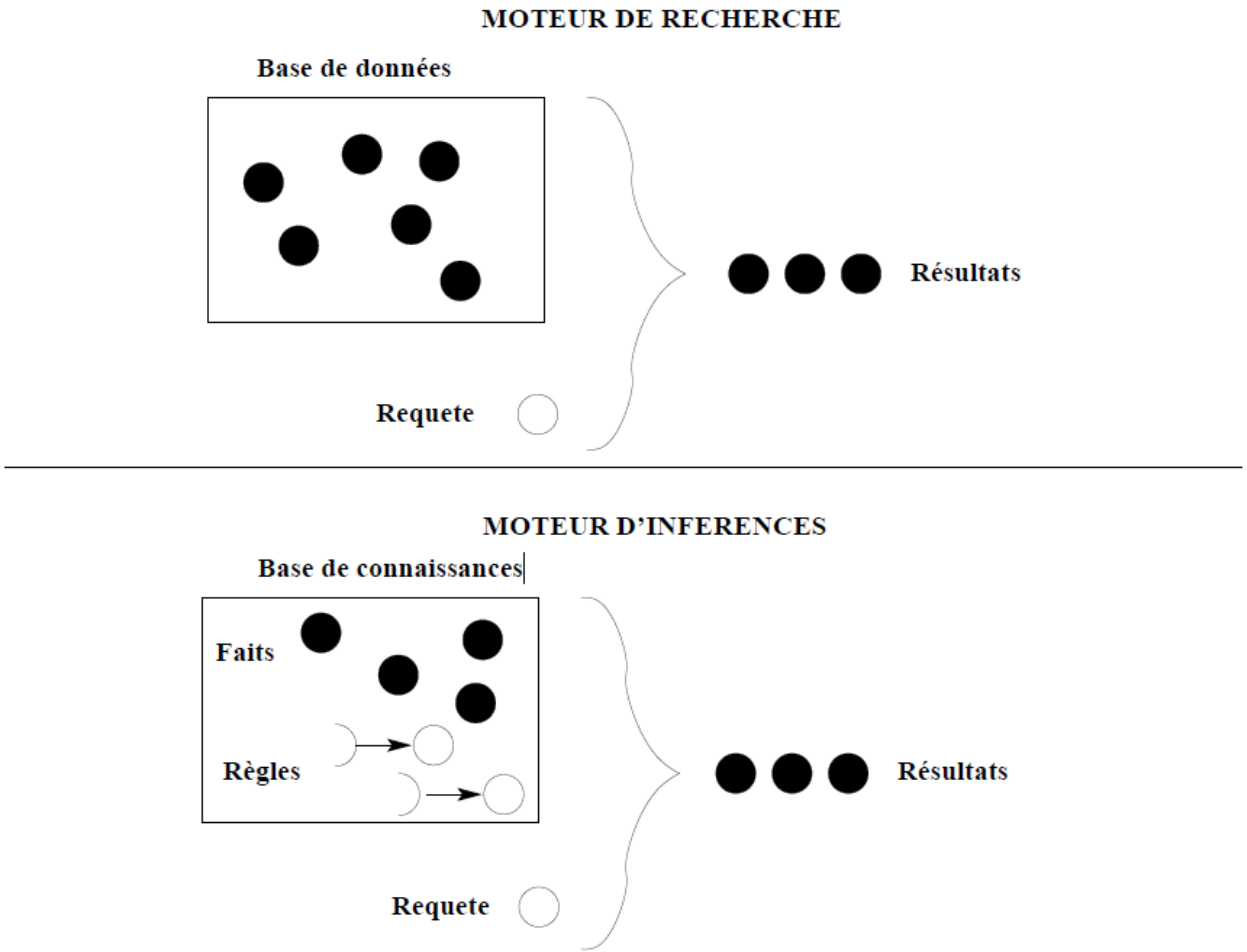
- "Tout" est **déclaration**
  $\Rightarrow$ Suite de déclaration qui constitue une **base de connaissances**
  $\Rightarrow$ On affirme **ce qui est**, sans forcément préciser **ce qu'il faut en faire**
- Exemples :
  "Tout homme est mortel"
  "Socrate est un homme"

Programmation déclarative

Extraction d'information

- Pour obtenir une information pouvant être déduite de cette base de connaissances, on a besoin d'un **moteur d'inférence** et d'une **requête**.
- Le moteur d'inférence met en oeuvre une procédure d'extraction de l'information à partir de la requête.

Programmation déclarative



Programmation (en) logique

Programmation logique

- Forme particulière de la programmation déclarative
- Déclaration = formule logique
- Exemples :
  homme(socrate)
  $\forall$ x, homme(x) $\Rightarrow$ mortel(x)
- $\Rightarrow$ Syntaxe simple et dépouillé
  $\Rightarrow$ Plus de contrôle donné à la machine
- Un langage phare : le **PROLOG**

Point de repères

- **1930** Calcul des prédicats (J. Herbrand)
- **1965** Principe de résolution (J. A. Robinson)
- **1970** Utiliser la logique comme *langage de programmation*
  $\Rightarrow$ clauses de Horn (R. Kowalski)
  $\Rightarrow$ Q-systèmes (A. Colmerauer)
- **1972** Premier interprète PROLOG
  A. Colmerauer et P. Roussel - Université d'Aix-Marseille
- **1977** Premier compilateur PROLOG
  D. H. D. Warren - Université d'Édimbourg
- **1980** Projet japonais des ordinateurs de 5e génération
- **1990** PROLOG évolue vers la *Programmation par Contraintes*

Sommaire

Aspects syntaxiques

Données

- Un ensemble **P** de variables propositionnelles
  $\mathbf{P} = \{ \mathbf{p}, \mathbf{q}, \mathbf{r}, \mathbf{...} \}$
- Un ensemble **C** de connecteurs
  $\mathbf{C} = \{ \neg, \wedge, \vee, \rightarrow, \leftrightarrow \}$

Formules

- **p** est une formule si $\mathbf{p} \in \mathbf{P}$
- $\neg$**(H)** est une formule si **H** est une formule
- **(H)** Δ **K** est une formule si **H** et **K** sont des formules et $\Delta \in \mathbf{C}$

Aspects sémantiques

Aspects sémantiques

- Logique booléenne : faux (0) ou vrai (1)
- Utilisation de table de vérité
- Formules particulières :
  - Tautologies : $p \vee \neg p$
  - Formules inconsistantes : $p \wedge \neg p$
- Formes normales
  - conjonctives : conjonction de disjonction $(\wedge\vee)$
  - disjonctives : disjonction de conjonction $(\vee\wedge)$

Aspects déductifs

Conséqence logique

Soit A = {F1, ..., Fn} un ensemble de n formules
$A \vdash G$ ssi $\vdash (F1 \wedge ... \wedge Fn) \rightarrow G$

Notion de réfutation

e.g. Démonstration par l'absurde
$A \vdash G$ ssi $\vdash (F1 \wedge ... \wedge Fn \wedge \neg G)$ est inconsistante

Déduction

Quelques règles classiques :

- modus ponens : $p, p \rightarrow q \vdash q$
- syllogisme : $p \rightarrow q, q \rightarrow r \vdash p \rightarrow r$
- principe de résolution (Robinson) : $l \vee C_1, \neg l \vee C_2 \vdash C_1 \vee C_2$

Sommaire

Limites du calcul propositionnel

Exemple

- Les chandelles sont faites pour éclairer
- **Quelques** chandelles éclairent très mal
- **Quelques** objets qui sont faits pour éclairer **le** font très mal

$\Rightarrow$ Modélisation impossible

Logique des prédicats

Exemple

- Les chandelles sont faites pour éclairer
  $\forall$ X, chandelle(X) $\rightarrow$ éclaire(X)
- **Quelques** chandelles éclairent très mal
  $\exists$X, chandelle(X) $\wedge$ éclaireMal(X)
- **Quelques** objets qui sont faits pour éclairer **le** font très mal
  $\exists$X, éclaire(X) $\wedge$ éclaireMal(X)

$\Rightarrow$ ~~Modélisation impossible~~

Logique des prédicats

Syntaxe

- Des **connecteurs** ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$)
- Des **variables** (X, Y, ...)
- Des **quantificateurs** ($\forall, \exists$)
- Des relations, appelés **prédicats** (éclaire, éclaireMal, ...)
- Des symboles de **fonctions** et **constantes**.

Vocabulaire

Définition

Un **terme** peut être :

- une **variable**, *e.g.* X, Nom, ...
- une **constante**, *e.g.* toto, 12, ...
- une **fonction**, *e.g.* homme(socrate), moins(2,5)
  (l'**arité** est le nombre d'opérandes associées) $\Rightarrow$ désigne un objet, résultat d'une opération sur une ou plusieurs autre objets,
  i.e. pas de valeur de vérité.

Vocabulaire

Définition

Une **formule atomique** (ou **littéral positif**) est définie à partir des **prédicats** (symbole de relation), et de termes.
Exemples :

- est_en_France(Paris)
- est_en(Paris, France)
- est_en(X, Y)

$\Rightarrow$ possède une interprétation , *i.e.* une valeur de vérité, V ou F

Vocabulaire

Définition

Une **formule** est définie à partir des formules atomiques, des connecteurs et des quantificateurs.
Si **F** et **G** sont des formules, alors les expressions suivantes sont aussi des formules :

- $\neg$ (**F**)
- **(F)** $\wedge$ **(G)** et **(F)** $\vee$ **(G)**
- **(F)** $\rightarrow$ **(G)** et **(F)** $\leftrightarrow$ **(G)**
- $\forall$x (**F**) et $\exists$x (**F**)

Vocabulaire

Définition

Une **occurence** d'une variable x dans une formule F est un endroit où x apparait dans sans être immédiatement précédée d'un quantificateur $\forall$ ou $\exists$

Définition

Une occurence de variable est dite **liées** quand elle est dans le champs de portée d'un quantificateur, sinon elle est **libre**.

Exemple

$\exists x(P(x,y)) \wedge \forall z(Q(z,x))$
Determiner les occurences de variables, puis si elle sont libre ou liée.

Vocabulaire

Définition

Une variable est libre dans le une formule F si elle a au moins une occurence libre dans F.

Définition

Une formule n'ayant pas de variable libre est dite **close**

Aspects sémantiques

Preuves et Démonstrations

- Des représentations adaptées des formules
  - forme normale **prénexe**
  - forme standard de **Skolem**
  - $\Rightarrow$ **Forme clausale**
- Un théorème bien utile !
  - le théorème de **Herbrand** et son corollaire
- Des démarches adaptées
  - exploitation de la négation sous forme clausale
  - exploitation du raisonnement par l'absurde
  - partir de la conclusion et arriver à une contradiction
  - utiliser le principe de résolution

Sommaire

Un exemple

Données

- $\forall x \forall y \forall z$ (pere(x,y)$\wedge$parent(y,z)) $\rightarrow$ grand-pere(x,z)
- $\forall x \forall y$ ((mere(x,y) $\vee$ (pere(x,y)) $\rightarrow$ parent(x,y)
- mere(a,b) pere(c,d) pere(d,a) pere(e,c)

But

On veut prouver $\exists$x, grand-pere(x,b)

Un exemple

Mise sous forme clausale

- $\neg pere(x_1, y_1) \vee \neg parent(y_1, z_1) \vee$ grand-pere(x_1, z_1)
- $\neg mere(x_2, y_2) \vee parent(x_2, y_2)$
- $\neg pere(x_3, y_3) \vee parent(x_3, y_3)$
- mere(a,b) pere(c,d) pere(d,a) pere(e,c)

But

Négation sous forme clausale : ¬grand-pere(x,b)
On part de la conclusion pour atteindre $\square$

Résultat

On obtient au moins une solution **pere(e,c)**

Un exemple

Bilan

- On a réussi à prouver $\exists$x, grand-pere(x,b)
- On a réussi à **calculer** un x
- Les unifications utilisées lors de la résolution, peuvent être assimilées à des calculs
  $\Rightarrow$ on donne des valeurs aux variables

Calcul = programmation

- on va pouvoir programmer avec la logique
- on automatise complètement le processus de preuves/démonstration !