Programmation Logique et Intelligence Artificielle

IUP 3 Université Paris 7

Organisation

- Pages Web:

  `http://www.liafa.jussieu.fr/~klimann/`
  `http://www.liafa.jussieu.fr/~kesner/`

- Contrôle des connaissances:
  Contrôle continu = Projet
  Note finale première session = 2/3 Note examen + 1/3 Note contrôle continu
  Note finale deuxième session = max(2/3 examen deux + 1/3 CC, examen deux)

- Cours: Mercredi 14h30, Delia Kesner (kesner@pps.jussieu.fr)

- Semaine prochaine, TP: Mercredi 16h30, TD: Jeudi 8h30, Ines Klimann (klimann@liafa.jussieu.fr)

- Ensuite : TD: Mercredi 16h30, TP: Jeudi 8h30.

- Bibliographie:

  - I. Bratko, Programmation en Prolog pour l'intelligence artificielle, 1988.
  - J.M. Alliot et T.Schiex, Intelligence Artificielle et Informatique Théorique, Cépaduès Editions, 1993.
  - N. Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann, 1998.
  - S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 1994.

Plan du cours

- Introduction à la programmation en Prolog
  - Programmation logique
  - Fonctionnement de base de Prolog
  - Recursivité, Listes, Arithmétique
- Introduction à l'Intelligence Artificielle
  - Les agents intelligents (du très simple au plus complexe)
  - Les algorithmes de recherche
  - Les algorithmes de jeux
  - Les algorithmes d'apprentissage
    * Les algorithmes de classification
    * Les réseaux de neurones

Programmation en Prolog / Introduction

Programmation:

- Écrire un texte, qui peut être exécuté par une machine (virtuelle)
- Résoudre un problème: Pour une classe de problèmes développer une méthode générale pour les résoudre

Programmation en Logique:

- texte est écrit dans un formalisme logique
- calcul de la solution du problème en déduisant à partir d'une description déclarative du problème:
  - Quels sont les objets ?
  - Quels sont leurs propriétés ?
  - Quels sont les relations entre eux ?

Exemple

La famille Dupont consiste en Paul, Marie, Marc, Claudia et Toto.

- Fait: Paul, Marie, Marc et Claudia sont des personnes.
- Fait: Toto est un chat.
- Fait: Marie et Claudia sont féminines.
- Fait: Paul et Marc sont masculins.
- Fait: Toto est masculin.

En PROLOG:

```
personne(paul).
personne(marie).
personne(marc).
personne(claudia).
chat(toto).
feminine(marie).
feminine(claudia).
masculin(paul).
masculin(marc).
masculin(toto).
```

Règles et Questions

La propriété femme peut être déduit de personne et féminine.

Règle: Quelqu'un est femme, si

1. ce quelqu'un est une personne, et
2. ce quelqu'un est féminine

En PROLOG:

```
femme(Quelquun) :-
      personne(Quelquun),
      feminine(Quelquun).
```

Questions:

- Est-ce que toto est un chat ?
- Est-ce que bob est un chat ?
- Qui sont les femmes ?

Hypothèse du monde clos:
Prolog connaît seulement ce qu'on lui dit

En PROLOG:

```
?- chat(toto).
Yes

?- chat(bob).
No

?- femme(Qui).
Qui = marie ;
Qui = claudia ;
No
```

Programmer en PROLOG

- Construire une base de connaissance pour le problème
- Poser des questions

Construire une base de connaissance

Les faits (ou assertions) modélisent des propriétés.

- Le chat s'appelle toto (en PROLOG: `chat(toto).`)
  s'obtient en utilisant le schéma X est un chat. On appelle chat un prédicat.

- Les personnes sont Paul, Marie, Marc et Claudia.
  On utilise le schéma X est une personne.

```
personne(paul).
personne(marie).
personne(marc).
personne(claudia).
```

- Paul, Marc et Toto sont masculins.

```
masculin(paul).
masculin(marc).
masculin(toto).
```

- Marc et Claudia sont les enfants de Paul et Marie.

```
enfant(marc,paul).
enfant(claudia,paul).
enfant(marc,marie).
enfant(claudia,marie).
```

Le prédicat enfant définit la relation
X est enfant de Y.

Règles

On peut définir des nouveaux prédicats à partir d'autres prédicats. Des propriétés (comme femme) ou des relations.

- Exemple: X est père de Y
- Les pères sont masculins et ont un enfant.
- X est père de Y, si X est masculin et Y est enfant de X
- ```
  pere(Pere, Enfant) :-
      masculin(Pere),
      enfant(Enfant,Pere).
  ```

  ou
  ```

```
pere(X, Y) :-
   masculin(X),
   enfant(Y,X).
```

- Une règle a la forme: `Tête :- Corps.`
- Pere, Enfant, X et Y sont des variables.

Notation

Très important !!

- Les noms des variables commencent toujours avec une majuscule (`X`, `Y`, `Cours`, `Idiot`, `C_a_1`,etc.) ou commencent avec "_" (nouvelles variables crées par le système). La variable _ (variable anonyme) a un statut spécial.
- Les noms d'objets (paul,toto) et les noms de prédicats,relation (enfant) commencent toujours avec une minuscule ou sont entouré de `'`.
- Deux relations avec un nombre différent d'arguments peuvent avoir le même nom.
- Attention: N'utilisez pas d'accents dans les noms de variables et relations.

Poser des questions

- Après avoir chargé le programme PROLOG (la base de connaissance) on peut poser des questions au système.
- Les questions peuvent contenir des variables.
- Des questions simples (buts) peuvent être combinées avec le “et” logique.
- Exemple:

```
?- chat(Chat), feminine(Chat).
No
?- chat(Chat), masculin(Chat).
Chat = toto
Yes
```

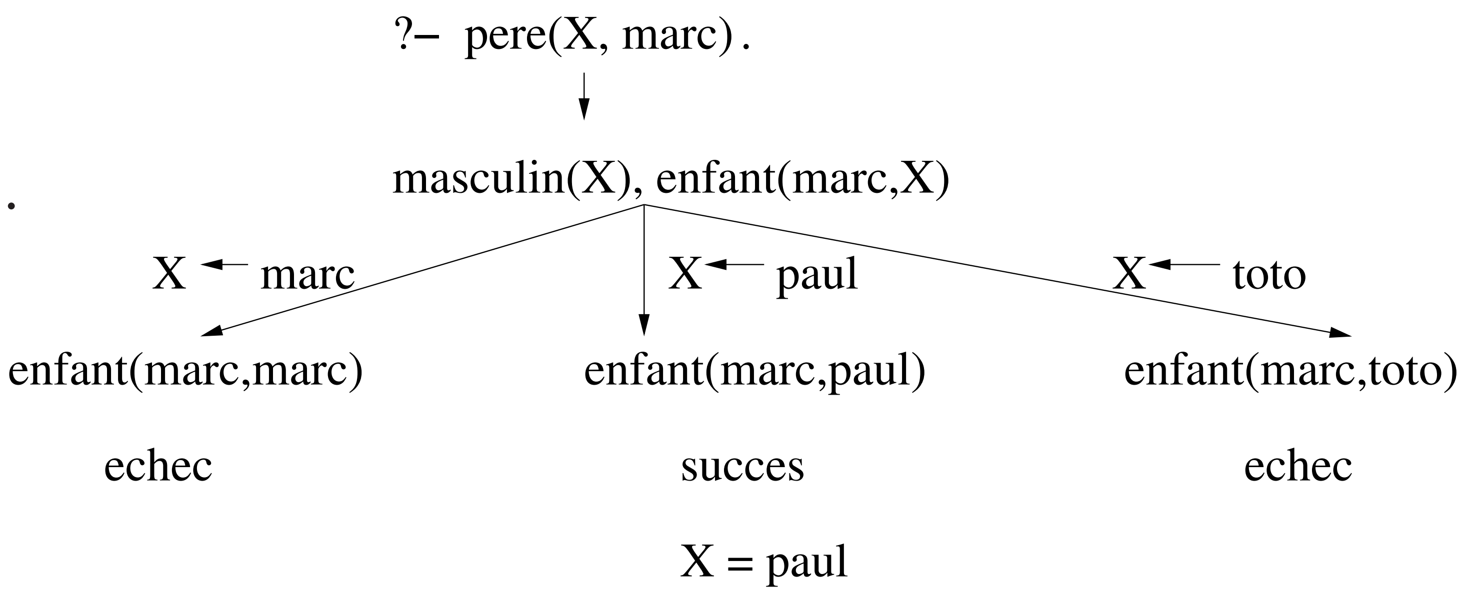
Comment répond PROLOG aux questions ?

Exemple:

```
enfant(marc,paul).
enfant(claudia,paul).
enfant(marc,marie).
enfant(claudia,marie).

masculin(marc).
masculin(paul).
masculin(toto).

pere(X, Y) :-
   masculin(X),
   enfant(Y,X).
```



Comment répond PROLOG aux questions ?

Ici: fonctionnement simplifié ! (plus de détails après)

- Si la question comporte plusieurs buts, ils sont résolus de gauche à droite.

- PROLOG cherche du haut vers le bas dans la base de connaissances si une assertion ou la tête d'une règle correspond au but traité.

- Si il y a une assertion qui “marche”, le but traité est “effacé” (résolu). PROLOG mémorise les valeurs éventuelles de variables substituées.

- Si il y a une règle dont la tête correspond au but traité, PROLOG remplace le but par le corps de règle et continue. Les valeurs éventuelles de variables substituées sont mémorisées.

- Si il n'y a pas d'assertion ou règle qui “marche” le but “échoue” et PROLOG revient en arrière pour chercher une autre solution.

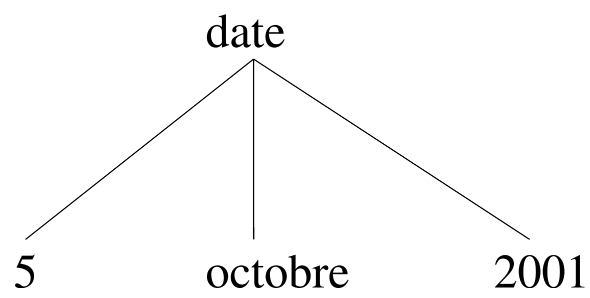
Syntaxe de Prolog

Un programme (et les données) est constitué de termes.
Un terme est

- une constante (terme simple)
  - un atome, un nombre
- une variable (terme simple)
  - cas special: variable anonyme _
    (Exemple: `a_un_enfant(X) :- enfant(_,X).`)
- une structure (terme composé)
  - symbole fonctionnel
  - des arguments séparés de ,
  - On écrit certains symboles fonctionnels comme des opérateurs (+,-,*, etc.).

– Exemple: Pour représenter une date, on peut écrire: `date(5,octobre,2001)` où `date` est un symbole fonctionnel d'arité 3.

- le terme `date(5,octobre,2001)` peut être vu comme l'arbre:



Comment répond PROLOG aux questions ?

Les ingrédients de base:

- manipulation de termes
  - Unification
  - Substitution
- Règles de preuves
- Stratégies de recherche
  - de gauche à droite
  - du haut vers le bas
- Retour en arrière (Backtracking)

Unification

essaie de faire égaux deux termes en instanciant (le moins détaillé possible) leurs variables.

- Exemple: `mange(toto,X)` et `mange(toto,poisson)` s'unifient avec instanciation `X` $\leftarrow$ `poisson`.

- Deux termes s'unifient si
  - ils sont identiques, ou
  - les variables dans les deux termes peuvent être instanciées par des objets de sorte qu'après substitution des variables par ces objets les termes deviennent identiques.

- Substitution:
  - Substituer la variable v par un terme s dans un terme t signifie remplacer toutes les occurrences de v dans t par s.
  - Notation : $t' = t\{v \leftarrow s\}$

- On généralise : substituer les variables $x_1, \ldots, x_n$ par les termes $s_1, \ldots, s_n$ dans un terme t signifie remplacer simultanément toutes les occurrences de $x_1, \ldots, x_n$ par $s_1, \ldots, s_n$ dans t.
- Example : $f(x_1, x_2)\{x_1 \leftarrow g(x_2), x_2 \leftarrow b\} = f(g(x_2), b)$
- t' est une instance de t

- Une substitution est noté σ et peut concerner plusieurs variables. On note $t\sigma$ le terme qu'on obtient en appliquant la substitution σ au terme t.

- t est une instance commune de t' et de t'', s'il existe une substitution σ telles que $t \equiv t'\sigma \equiv t''\sigma$. La substitution σ est appelée unificateur.

Unification - Exemples

En PROLOG l'unification entre deux termes est predéfinie par l'opérateur =.

- `mange(Y,poisson) = mange(toto,X)`
- `Z = mange(toto,X)`
- `mange(X,souris) = mange(toto,X)`
- `mange(X,X) = mange(toto,Z)`
- `mange(toto,X,Z) = mange(toto,Z)`

Unificateur le plus général

Parmi tous les unificateurs on s'intéresse à l'unificateur le plus général. C'est celui qui instancie le moins possible les variables.

- Exemple:

```
?- date(D, M, 2001) = date(D1, octobre, Y1).
D = _G354
M = octobre
D1 = _G354
Y1 = 2001
```

```
?- date(D, M, 2001) = date(D1, octobre, Y1),
   date(D, M, 2001) = date(15, M, Y).
D = 15
M = octobre
D1 = 15
Y1 = 2001
Y = 2001
```

Pendant l'exécution de buts consécutifs les variables sont instanciées typiquement de plus en plus.

Unification - Algorithme

```
Entrée:  deux termes t1 et t2
Sortie:  la substitution σ

Initialiser σ à vide
Empiler t1 = t2
Tant que pile non vide
    dépiler X = Y de la pile
    Suivant les cas:
```

- X est une variable sans occurrence dans Y: On substitue Y à X dans la pile et dans σ, on rajoute $\{X \leftarrow Y\}$ à σ.
- Y est une variable sans occurrence dans X: On substitue X à Y dans la pile et dans σ, on rajoute $\{Y \leftarrow X\}$ à σ.
- X et Y sont des constantes ou variables identiques: On continue
- Cas général:

$X = f(X_1, \ldots, X_n)$ et $Y = f(Y_1, \ldots, Y_n)$.
On empile $X_i = Y_i$ pour i de 1 à n.

Sinon échec.

Exemple

```
f(g(X,A,h(X,b)),Z) = f(g(A,b,Z),Y)

On empile les arguments de gauche à droite.
Après un tour la pile est:

g(X,A,h(X,b)) =  g(A,b,Z)  Substitution {}
     Z       =    Y

Après le deuxième tour:
     X       =    A   Substitution {}
     A       =    b
  h(X,b)     =    Z
     Z       =    Y
```

```
Ensuite:
     A       =    b   Substitution {X <- A}
  h(A,b)     =    Z
     Z       =    Y

Ensuite:
  h(b,b)     =    Z   Substitution {X <- b, A <- b}
     Z       =    Y
Ensuite:
     h(b,b) =     Y    Substitution
                     {X <- b, A <- b, Z <- h(b,b)}
Résultat:
     {X <- b, A <- b, Z <- h(b,b), Y <- h(b,b)}
```

Comment répond PROLOG aux questions ?

```
 PROGRAMME
enfant(marc,paul).
enfant(claudia,paul).
enfant(marc,marie).
enfant(claudia,marie).

masculin(marc).
masculin(paul).
masculin(toto).

pere(X, Y) :-
   masculin(X),
   enfant(Y,X).
```

```
pere(X,marc)
  Question
```

- PROLOG essaie d'unifier le premier but de la question avec un fait ou la tête d'une règle. On prend toujours la première du haut qui marche.
- On copie la clause, les variables sont renommées. Ici:
  `pere(X', Y') :- masculin(X'),enfant(Y',X').`
- On unifie `pere(X', Y')` et `pere(X,marc)` et on applique l'unificateur `{X <- X', Y' <- marc}` à la question et à la copie de la clause choisie:
  `pere(X', marc) :- masculin(X'),enfant(marc,X').`

```
masculin(X'),enfant(marc,X')
  Nouvelle Question
```

- On remplace le premier but avec le corps de la copie de la règle.
- On continue avec le nouveau but.
- Le premier fait qui marche: `masculin(marc)` avec unificateur `{X' <- marc}`
- Si on trouve un fait correspondant on efface le but et on applique l'unificateur au reste de la question: `enfant(marc,marc)`

```
enfant(marc,marc)
  Nouvelle Question
```

- On ne trouve pas de fait ou de tête de règle correspondant.
- On revient sur le dernier point de décision (retour en arrière, backtracking)
- Ici: On revient à la question
  `masculin(X'),enfant(marc,X').`
- On oublie toutes les substitutions faites après le point de décision.

```
masculin(X'),enfant(marc,X')
  Ancienne Question
```

- On continue à chercher un fait ou une tête de règle qui marche (toujours du haut vers le bas).
- On trouve `masculin(paul)` avec unificateur `{X' <- paul}`.
- On efface le but et on applique l'unificateur au reste de la question.
- On obtient `enfant(marc,paul)`.

```
enfant(marc,paul)
 Nouvelle Question
```

- On cherche dans le programme et on trouve le fait `enfant(marc,paul)`.
- On efface le but.
- La question est vide, donc résolu.
- On regarde du début vers la fin quelles sont les substitution qui ont été faites, on les appliques une par une et on les donnent. Ici: `{X <- paul}`.

```
masculin(X'),enfant(marc,X')
  Ancienne Question
```

- Si l'utilisateur le veut on revient en arrière pour chercher une autre solution.
- `masculin(X')` marche aussi avec `masculin(toto)` et l'unificateur `{X' <- toto}`.
- On obtient un nouveau but: `enfant(marc,toto)`.
- Ce but échoue et quand on revient en arrière il n'y a plus de choix.