Les éléments de base du langage Prolog (cours1)

* Opérationnel sur Ordinateur
* Logiciel utilisé : ECLiPSE

Domaine d’application : IA, plannification, prototypage (une idée > rapide à mettre en œuvre), simulation, optimisation …

C’est ce que l’on appelle une ***Programmation par contrainte***

Notre point de vue (par rapport au cours de logique) :

* *Prolog* peut être vu comme un démonstrateur de théorème (résolution sur les clauses de Horn)

La syntaxe à l’origine était très simple et dépouillée :

+ Facilité d’apprentissage

– Il fallait tout réinventer : entiers naturels, listes,…

Nous verrons la syntaxe originelle et la sémantique associée puis nous introduirons quelques notions syntaxiques.

Les termes et les relations (⬄prédicat) (CF logique : Prédicat) sont les **deux SEULES constructions de base** du langage.

* Les termes servent à noter (directement ou non), les objets qui interviennent dans les programmes (ils désignent des éléments syntaxiques d’un domaine D : Univers de Herbrand (c’est l’univers syntaxique des programmes)

**<terme> := <variable> || <termeatomique> || <termeconstruit>**

**<termeatomique> := <atome> || <nombre> || <chaine>**

**<termeconstruit> := <identfonct>(<terme>{,<terme>}\*)**

Vocabulaire des termes :

* Une <variable> est un identificateur dont la première lettre est une majuscule
* Une <variable> désigne un terme inconnu (comme en logique)
* Un <atome> est un identificateur dont la première lettre est une minuscule
* Un <atome> est un objet de l’univers de Herbrand
* Un <nombre> est noté classiquement en représentation décimal
* Un <identfonct> est un identificateur dont la première lettre est une minuscule (rem : une fonction d’arité=0 est un atome)
* Un <termeconstruit> est un objet de l’univers de Herbrand (**désigné indirectement**!)

Quelques exemples :

* Des Termes atomiques

Nombre entier (147,-74,…)

Nombre décimal

Chaine (‘coucou’,…)

Atome (hj12, insa,…)

* Des Variables (**H**j12, **I**nsa,…)
* Des termes construits (hg4(147,hj12,truc(4,’4info’,Insa)), a(B),…)

rmq : les **variables apparaissent toujours aux feuilles**

**Les relations** servent à noter les liens qui existent entre les objets notés par les termes

**<relation> := <identpredicat> (<terme> {, <terme> }\* ) || <identdepredicat>**

Un identdepredicat est (en général un verbe) un identificateur **dont la première lettre est une minuscule**

Ex : **h**g4(147,hj12,truc(4,’4info’,Insa))

**a**(B)

rmq : De même que pour les termes construits (subtilité utilisé par Prolog), on **peut quantifier les prédicats**

/!\ ATTENTION à la syntaxe : Nous remarquons qu’il n’y a **aucune différence syntaxique entre un terme construit et une relation** (identfonct et identpred)

* En Prolog pas de différence entre le programme (composé de relation) et les données de ce programme (notées grâce aux termes, arguments des prédicats)
* En Prolog aucune trace d’instruction (ni itération, ni conditionnelle,…)

**Les faits et règles d’un programme Prolog sont construits à partir des relations**, de la même façon qu’une clause du langage des prédicats sont construite à partir des formules atomiques

**<programm>  := <clause> {.<clause>}\***

**<clause>  := <fait>||<règle>**

**<fait> := <relation>**

**<règle> := <relation> :- <relation>{,<relation>}\***

Exemple de programme :

desx (fr(5,a)). // Clause <fait>,desx est un <idpred> et fr <idfonct>

desx(X) :- att(X,Y),desx(Y),xcd(Y,’ah bon’). // Clause <règle>, desx est un <idpred>

att(5,7).

att(12,8) :- desx(11), desx(fr(2,5)).

Le programme comporte *2 faits et 2 règles*

A partir de leur position syntaxique (à l’extérieur de tout parenthèsage) Prolog déduit que desx, xcd et att sont des identificateurs de prédicat. Fr est quant à lui un identificateur de fonction.

* **Tête de clause** : la relation qui est **avant :-** (la relation dans les faits)
* **Corps de clause** : **après :-** (vide dans les faits)
* **Paquet de clauses** : ensemble de clauses qui ont même ident de prédicat en tête de clause

Rmq : **Pour des raisons d’efficacité Prolog exige que les clauses d’un même paquet soient contigües**

Exemple :

desx(X) :- att(X,Y) , desx(Y) , xcd(Y,’ah bon’).

att(5,7). // INCORRECT car alternance desx att desx att …..

desx(fr(5,A)).

att…

Les commentaires Prolog **/\* ………\*/ ou %**

Il existe une troisième forme de clause : **les requêtes** (qui ne sont pas des clauses que l’on trouve dans les programmes, **mais dans la fenêtre de l’interpréteur**)

**<requete> := <relation> {,<relation>}\*.**

(*Dans un premier temps nous nous limiterons à des requêtes réduites à une seule relation*)

Rmq : les corps de règle et les requêtes ont exactement la même syntaxe. Pourtant il n’est pas possible de les confondre car elles n’ont pas le même contexte !

On a donc 3 formes de clauses (faits, règles, requêtes). Concrètement Prolog est un interpréteur :

* Le programmeur saisit **son programme (fais + règles)** avec un éditeur, et le sauvegarde dans un fichier texte
* Dans un 2nd temps il appelle l’interpréteur et lui adresse des requêtes ! (la première requête consiste en la consultation du fichier qui contient le programme)

|  |  |  |
| --- | --- | --- |
|  | Tête | Corps |
| Fait | r1  r1 :- | r1,r3,r4. |
| Regle |
| requete |  | r2,r3. |

**La sémantique de Prolog :**

Il faut commencer par se débarrasser de ses aprioris concernant la programmation traditionnelle :

* Classiquement => un algorithme écrit dans un langage adapté à l’ordinateur (un algorithme étant une séquence d’instruction qu’il faut exécuter pour atteindre un résultat)
* En Prolog, rien de tout ça ! **un programme Prolog = une description du monde**. Le programmeur énonce ce qu’il sait concernant un certain domaine : PAS la moindre algorithme
* Ex : 2>3 => il ne sait pas si on ne lui a pas défini … (comme un enfant prodige à qui il faut tt apprendre !)

*Programme Prolog = ENSEMBLE D’AFFIRMATION DU PROGRAMMEUR*

Le programmeur énonce, grâce à des faits et des règles, tout ce qu’il sait concernant un domaine :

* **:- signifie « à condition que »**
* La virgule (**,**) signifie presque « et » (signifie plutôt PUIS)

Exemple en français : Je sais qu’Eddy a 10 ans et que Vincent a 120 ans. Je sais aussi qu’on est jeune quand on a moins de 25 ans. (2 faits 1 règle)

jeune(X) :- age(X,Y), inf(Y,25).

age(eddy=10). /\*eddy et non Eddy car c’est un atome et non une variable !\*/

age(vincent=120).

Rem : Implicite

Les variables en tête de clause sont universellement quantifiées

Les variables qui n’apparaissent que dans le corps de clause sont existentiellement quantifiées

Traduction en Logique : **{pour tout x (il existe y (age(x,y) … }**

*Programme Prolog = ENSEMBLE DE CLAUSES DE HORN (vue du logicien)*

* **Clause de Horn est une clause comportant exactement un littéral positif (un littéral POSITIF = sans négation)**

Reprenons l’exemple en langage des clauses

{ !age(x,y) **v** !inf(y,25) **v** jeune(x) …..}

En effet toute clause Prolog que l’on veut exprimer sous forme de clause du langage des prédicats ne peut donner lieu **qu’à 1 seule clause de Horn**

* Evident pour les faits qui sont des clauses de Horn
* Les autres sont de la forme :
  + t :-c1,…,cn soit
  + c1**^**c2**^**…**^**cn=t soit
  + **!(**c1**^**…**^**cn**)v** t //mettre la négation
  + **!**c1**v!**c2…**v**t

**Le programmeur ne peut donc exprimer que des clauses de Horn.** Cela a un inconvénient (évident car le langage prologue n’est pas logiquement équivalent à L) et deux avantages (aucun moyen d’exprimer un ensemble contradictoire car **la résolvante de 2 clauses de Horn et une clause de Horn** et il est donc impossible d’inférer la clause vide ! + La résolution est relativement efficace sur un ensemble de Horn)

*Programme Prolog = DEFINITION D’UN ENSEMBLE DE LITTERAUX POSITIF (vue du logicien)*

On peut voir un programme Prolog (P) comme une façon de noter l’ensemble (généralement fini) des littéraux positifs qui sont conséquences logiques de P

E = {L|P |= L}

Exemple :

jeune(X) :- age(X,Y), inf(Y,25).

age(eddy=10).

age(vincent=120).

E= {age(eddy,10),age(vincent,120),jeune(eddy)}

En effet si on ajoute la négation d’un de ces faites à P on obtient un ensemble contradictoire !

Programme P

jeune(X) :- age(X,Y), inf(Y,25).

age(eddy=10).

age(pierre=17).

age(vincent=120).

Appel Prolog (Linux) :

*>eclipsep*

*…..*

*[1] % prompt de prolog=> consultation du Programme*

[2] jeune(eddy).

Yes

[3] jeune(U). % quelle est la condition de U pour que U soit jeune ?

U=eddy(more ?) **; % “;” rajouter par le programmeur**

U = pierre(more ?)**;**

No.

[4]age(X,Y).

X=eddy.

Y=10(more ?)**;**

X=pierre.

Y=17(more ?)**;**

X=Vincent.

Y=120.

..

[6]age (vincent,Y). %quel âge a vincent ?

Y=120.

--------------------------------------------

(TP)

P1 : le monde de la famille

P2 : le monde de l’espérance

esthom().

…

estfem().

--------------------------------------------