PROGRAMMATION LOGIQUE

Prolog (PROgrammation LOGique) est un langage extraordinaire qui a servi de base aux programmes de recherche japonais sur les ordinateurs de 5ème génération. En Prolog, il nous suffit de décrire ce que l'on sait sur le domaine étudié, en vrac, dans l'ordre où ça nous vient (en Intelligence Artificielle, on appelle cela une base de connaissances), ensuite on décrit notre problème, Prolog va nous le résoudre, sans qu'on n'ait à lui dire comment faire ! Programmer en Prolog est la meilleure manière d'aborder la notion de la programmation déclarative. Donc, programmer en Prolog consistera à décrire les connaissances nécessaires pour résoudre un problème, laissant le soin de la solution au langage lui même. Le programmeur fournit les faits connus et le problème à résoudre, son centre d'intérêt devient le savoir et non plus les algorithmes l'exploitant. Du point de vue de l'utilisateur Prolog est un langage clair et concis particulièrement adopté aux systèmes experts, syntaxes du langage naturel, etc.

1. INTRODUCTION

1.1. Historique

Prolog est issu d'une idée abstraite sur la programmation logique évoquée par Bob Kowalski à Edimbourg au courant des années 70. Prolog en tant que langage de programmation a été premièrement développé par Alain Colmerauer et son équipe à Marseille. Depuis, plusieurs implémentations de Prolog ont été complétées, notons les plus connues de Roussel (1975) à l'université de Marseille, de Roberts (1977) à l'Université de Waterloo, de Pereira and Warren (1977, 1978) et d'autres (Clark et McCabe, Yokota).

1.2. Définition

Prolog est un langage de programmation basé sur le principe de résolution de la logique du premier ordre. L'idée de cette logique est que la déduction peut être vue comme une forme de calcul et qu'un énoncé déclaratif de la forme *P* si *Q* et *R* et *S* peut également s'interpréter procéduralement comme pour résoudre *P* il suffit de résoudre *Q* et *R* et *S*.

Un ensemble fini de règles qui définissent des relations entre objets constitue un programme logique. Dans un système de résolution ce qu'on connaît est représenté par un *ensemble de clauses* ou *Clauses de Horn* qui sont des énoncés de la forme suivante :

A est une *assertion*,

A si B_1, B_2, ..., B_n, cet énoncé est appelé une *règle* où *A* est la *tête* de la règle et les conditions B_1, B_2, ..., B_n sont le *corps*.

Du point de vue utilisateur, Prolog est flexible, modulable et concis. Habituellement un programme écrit en Prolog est beaucoup plus petit que le même écrit en un langage procédural.

2. LA PROGRAMMATION LOGIQUE

2.1. Un mode de programmation à part

Il est important avant d'expliquer ce qu'est PROLOG et d'étudier son utilisation de bien voir ce qu'il n'est pas ; c'est à dire de comprendre sa 'philosophie'. Pour cela, il faut décrire ce qu'est la programmation logique et la comparer aux autres modes de programmation.

2. Les autres modes de programmation :

La programmation impérative

Cette programmation s'inscrit dans une démarche algorithmique qui décrit la façon de traiter les données pour atteindre un résultat par une série d'actions. Celles-ci sont toujours de trois types : le test, l'ordre (chaque instruction par laquelle le programme passe est impérative) et l'itération (obtenue grâce aux boucles).
Le déroulement du programme est parfaitement déterministe.
Exemple de langages : Fortran, Pascal, C …

La programmation fonctionnelle

Le résultat est ici comme la composition de fonctions. Pratiquement, elle est proche de la programmation impérative ; cependant ses fondements (λ-calcul) ainsi que l'absence de boucles et d'affectation (du moins dans sa forme théorique) en fait un mode de programmation à part.
Remarque: Dans la programmation fonctionnelle, on distingue deux grandes familles : - les langages fortement typés : ML (ex : Caml) - les langages non typés : LISP (ex : Scheme)

La programmation logique

Les modes de programmation décrits juste au-dessus sont dits procéduraux car ils cherchent à obtenir le résultat grâce à une procédure (qui peut être une suite d'actions ou une composition de fonctions). A cela on oppose la programmation logique qui est dite déclarative. En effet, ici on ne s'occupe pas de la manière d'obtenir le résultat ; par contre, le programmeur doit faire la description du problème à résoudre en listant les objets concernés, les propriétés et les relations qu'ils vérifient.
Ensuite, le mécanisme de résolution (pris entièrement en charge par le langage) est général et universel. Il parcourt toutes les possibilités du problème et peut donc retourner plusieurs solutions.

La programmation orientée objet

Ce mode de programmation a été mis à part car il regroupe en fait tous les modes pécédemment vus en utilisant à la fois des techniques déclaratives et d'autres procédurales.
Exemple de langages : C++, Java …

2.3. Conseils pratiques

Vous cherchez un Prolog ? je vous conseille SWI Prolog (sous Linux ou Windows), disponible gratuitement (licence GPL) au département informatique de l'Université d'Amsterdam (http://www.swi-prolog.org/ ou http://www.swi.psy.uva.nl/projects/SWI-Prolog/). Il en existe d'autres, en particulier GNU-Prolog (par l'INRIA). D'autres liens sur Prolog ? testez la WWW Virtual Library (en anglais).

On peut tester un programme en l'interprétant interactivement, soit en le compilant au préalable.

Solution interactive : appeler pl, puis donner comme but : `consult(nomfichier).` Ne pas oublier le point, il est inutile d'ajouter le .pl qui est ajouté par défaut au nom du fichier.

Solution compilée : je crée mon programme (xxx.pl) et je le compile par *pl -o xxx -c xxx.pl* (-c en dernier !). Puis j'appelle xxx, il me propose de donner des buts (je dois les terminer par .), et

pour l'aide (doc en ligne) :

- help. : toute l'aide
- help(1). : intro générale (voir www.swi.psy.uva.nl/projects/xcpe)
- help(2). : options de ligne de commande, historique, mode trace/spy (2-6), compilation et options (2-7) ,caractères spéciaux (2-12)
- help(3). : tous les prédicats prédéfinis, les types (3-4), comparaisons (3-5) fail et ! (3-6) assert (3-10) fichiers (3-13) E/S sur écran (3-15) string (3-19) opérateurs (3-20) arithmétique (3-21) listes (3-24) write (3-30) shell commands (3-32) fichiers (3-33), commandes break, trace, debug (3-34) appels DDE windows (3-40)
- help(4). : les modules
- help(5). : interface C
- help(6). : runtime
- help(7). : spécial hackers (pas pour nous)
- help(8). : index de TOUS les mots définis avec quelques mots d'explication

Pour déboguer, on utilise le mode `trace` : au goal je donne : `trace ,ma_question (MesParametres).` A chaque pas on appuie sur entrée (h pour l'aide des autres options). voir `help(2-6). listing.` réécrit les règles à l'écran, ou `listing(predicat).` pour toutes celles limitées à un prédicat.

Je quitte par `halt.` (le point compris !) ou CTRL D.

Vous installez l'interprète Prolog à votre système, ensuite vous utilisez votre éditeur de texte pour écrire vos programmes (par exemple Emacs). Une fois enregistré votre fichier (vous pouvez utilisez l'extension .pl pour vos fichiers sources prolog), vous appelez votre interprète Prolog en tapant pl à l'invite de votre prompt. Vous obtenez le point d'interrogation, c'est l'invite de l'interprète Prolog où vous pouvez « poser vos questions » suivant les déclarations que vous avez effectuées dans votre programme. Si par exemple je demande de l'aide sur un prédicat (ici « consult ») en tapant :

```
?-help(consult).
```

Voilà la réponse :

```
consult(+File)

     Read File as a Prolog source file. File may be a list of files,
in which case all members are consulted in turn. File may start with
the Unix shell special sequences ~, <user> and $<var>. File may also
be library(Name), in which case the libraries are searched for a
file with the specified name. See also library_directory/1 and
file_search_path/2. consult/1 may be abbreviated by just typing a
number of file names in a list. Examples:

        ?- consult(load).       % consult load or load.pl

        ?- [library(quintus)].  % load Quintus compatibility library

        ?- [user].

The predicate consult/1 is equivalent to load_files(Files, []),
except for handling the special file user, which reads clauses from
the terminal. See also the stream(Input) option of load_files/2.
```

2.4. Quelques définitions

Pour bien comprendre la syntaxe en Prolog, il est utile de rappeler qu'il s'intéresse comme la logique aux relations d'implications entre propositions. Voyons un exemple : la première proposition *" Qui aime la logique est content."* et la deuxième *" Platon aime la logique.",* nous permet de déduire la proposition *" Platon est content."*. Dans les propositions ci-dessus on distingue les **objets** (Platon, la logique), **des propriétés attachées à un objet** (content), et **des relations entre objets** (aime). On appelera **prédicats** des relations ou des propriétés entre objets et **arguments** les objets sur lesquels portent les prédicats.

3. RÈGLES SUR CONSTANTES ET VARIABLES SYMBOLIQUES

Le code source (programme) se compose d'un ensemble de règles (sous la forme : *but SI conditions*) et de faits (en général des affirmations : on précise ce qui est vrai, tout ce qui n'est pas précisé est faux ou inconnu).

Le langage gère des variables simples (entiers, réels,...) et des listes (mais pas les tableaux).

Les variables simples sont les entiers, les réels, les chaînes de caractères (entre " ") et les constantes symboliques (chaînes sans espace ni caractères spéciaux, commençant par une minuscule, mais ne nécessitant pas de "). Les noms de variables commencent obligatoirement par une majuscule (contrairement aux constantes symboliques), puis comme dans les autres langages est suivi de lettres, chiffres, et _ . Une variable "contenant" une valeur est dite "liée" ou "instanciée", une variable de valeur inconnue est dite "libre".

Le premier programme simple :

```
/*************** famille.pl *** *************/

%présentation des faits :

feminin(claude).        %se lit « claude est feminin »

feminin(sophie).

feminin(alice).

feminin(mathilde).

feminin(morgane).

feminin(lucile).

feminin(lisa).

feminin(anne).


masculin(sam).          %se lit « sam est masculin »

masculin(olivier).

masculin(leo).

masculin(marc).

masculin(gilles).

masculin(mathieu).

masculin(pierre).

masculin(pascal).

masculin(francis).

masculin(thomas).


mere(claude, sophie).              %se lit « claude est la mère de sophie »

mere(claude, gilles).

mere(claude, mathieu).

mere(claude, anne).
```

```
mere(lucile, lisa).

mere(lucile, alice).

mere(sophie, mathilde).

mere(sophie, leo).

mere(morgane, marc).

mere(anne, francis).


pere(sam, sophie).                  %se lit « sam est le père de sophie »

pere(sam, gilles).

pere(sam, mathieu).

pere(sam, anne).

pere(olivier, mathilde).

pere(olivier, leo).

pere(gilles, marc).

pere(pascal, francis).

pere(mathieu, lisa).

pere(mathieu, alice).


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

%se lit « X est_parent de Y si est seulement si « X est_mère de Y » OU
logique « X est_père de Y »

grandparent(X, Y):- parent(X, T), parent(T, Y).

%se lit « X est_grandparent de Y » si est seulement si « X est_parent de T»
ET logique « T est_grandparent de Y »

frere(X, Y):- masculin(X), parent(Z, X), parent(Z, Y).

%se lit « X est_masculin » et «Z est_parent de X », et «Z est_parent de
Y ».

soeur(X, Y):- feminin(X), parent(Z, X), parent(Z, Y).

fille(X, Y):- feminin(X), parent(Y, X).
```

```
fils(X, Y):- masculin(X), parent(Y, X).
```

`masculin/1` : est un "prédicat" et 1 est le nombre d'arguments pour le prédicat 'masculin'. Il sera soit vrai, soit faux, soit inconnu. Dans le "programme", toutes les règles concernant le même prédicat doivent être regroupées ensemble.

Pour exécuter ce programme sous Prolog, il suffit de l'enregistrer dans un fichier texte (famille.pl), puis d'appeler pl, charger le fichier par "`consult('famille.pl').`" (n'oubliez pas le '**.**' « point »).

On peut alors "exécuter" le programme, en fait on fait une requête :

```
?- masculin(marc).
Yes                              %ou true
?- masculin(anne).
No                               %ou fail
?- masculin(luc).
No
```

Prolog cherche à prouver que le but (goal) demandé est vrai. Pour cela, il analyse les règles, et considère comme faux tout ce qu'il n'a pas pu prouver. Quand le but contient une variable libre, Prolog la liera à toutes les possibilités :

```
?- masculin(X).
X=sam ;          %(entrez ; pour demander la solution suivante)
X=olivier
?- pere(Pere,mathieu).
Pere=sam ;…… etc ….
```

On peut combiner les buts à l'aide des opérations logiques et (,) , ou (;) et non (not). Il y a parfois plusieurs façons de définir le même prédicat.

On ne trouvera ici qu'une solution, pour véritablement tester ceci il faut augmenter notre base de connaissances. Le comportement de Prolog est donc différent suivant que les variables soient liées (il cherche alors à prouver la véracité) ou libres (il essaie alors de les lier à des valeurs plausibles). Ceci démontre une différence capitale entre un langage déclaratif (on dit simplement ce que l'on sait) et les langages procéduraux classiques (on doit dire comment l'ordinateur doit faire, en prévoyant tous les cas).

Exercice 1 : définir les prédicats :

oncle/2,

tante/2,

cousin/2,

cousine/2,

belle_mere/2,

beau_pere/2.

Lorsque vous interrogez l'interprète `?- frere(X, Y).` Que vous répond-il ? Vous trouvez ça 'bizarre' ?

Essayez d'introduire dans le programme à la suite des faits le prédicat suivant, ensuite changez également les prédicats `frere/2` et `sœur/2,` exécutez de nouveau, qu'est-ce qui change ?

```
different(X, Y):- not(X = Y).            %se lit « X est différent de Y »
```

```
frere(X, Y):- masculin(X), parent(Z, X), parent(Z, Y), different(X, Y).

%se lit « X est_masculin » et «Z est_parent de X », et «Z est_parent de Y »
et « X est_différent de Y »
```