Semaine 1

Thomas RIBAUT

Table des matières

1 L'histoire du Prolog

1.1 Les caractéristiques

Le langage Prolog, inventé par deux français : Alain Colmerauer et Philippe Roussel en 1972 est un langage de programmation prévu pour être utilisé dans les bases de données relationnelles, mais est aujourd'hui principalement utilisé dans les intelligences artificielles.

Ce langage, signifiant **"PROgrammation en LOGique"** a été pensé dans le but d'être utilisé par des personnes n'ayant pas de connaissances en informatique, ainsi, son principe de base repose sur la logique compréhensible par l'humain et non pas par la machine.

Le programmeur n'a pas pour rôle de définir tous les cas possibles de son programme, il devra seulement donner des **règles**, des **faits** et des **relations** pour ensuite interroger le programme. Les possibilités offertes par ce langage sont infinies, il sera alors possible de réaliser un programme à propos des relations entre des personnes, avec des animaux ou à propos de n'importe quel autre sujet abstrait.

1.2 Les principes fondamentaux

Le Prolog ne reposant que sur des prédicats, il est nécessaire de suivre une logique totalement différente des autres langages pour réaliser un programme en Prolog.

Il faudrat alors comprendre et différencier les trois points suivants :

a) Les faits

Les faits, composent la base de connaissances avec les règles, représentent les valeurs données au programme pour qu'il puisse construire une base pour son raisonnement. C'est ainsi en partant de ces faits qu'il pourra donner des réponses.

Exemples :

```
1 pere(paul,jean).
2 pere(paul, francois).
3
4 mere(marion,jean).
5 mere(francoise,marion).
6
7 animaux(chat).
8 animaux(poisson).
```

Ici les déclarations suivantes sont effectuées :

- Le père de Jean est Paul.
- Le père de Paul est François.
- La mère de Jean est Marion.
- La mère de Marion est Françoise.
- Un chat est un animal.
- Un poisson est un animal.

Il est recommandé d'enregistrer les faits ainsi que les règles dans un fichier.

b) Les règles

Les règles permettent d'établir des relations entre les faits, comme par exemple des conditions ou des liens logiques entre plusieurs prédicats.

Exemples :

```
grandpere(M,E) :-pere(M,X), -pere(X, E).
grandmere(M,E):-mere(M,X), -mere(X, E).

animalAvecJambes(X):-chat(X).
```

Ici les règles suivantes sont définies :

- M est le grand-père de E si M possède un père X et que ce même père X possède un père qui est M.
- M est la grand-mère de E si M possède une mère X et que cette même mère X possède une mère qui est M.
- Un animal possède des jambes si cet animal est un chat.

Il est recommandé d'enregistrer les faits ainsi que les règles dans un fichier.

c) Les requêtes

Les requêtes permettent d'exécuter le programme. Ainsi grâce aux requêtes, il est possible d'obtenir des réponses, de faire fonctionner une logique et de l'utiliser pour obtenir toutes les solutions possibles des requêtes demandées.

Exemples :

```
pere(X,jean).
X = paul.

pere(paul, X).
X = jean.

mere(X,X).
false.

mere(_,_).
true;
true.

mere(X,marion).
X = francoise.

animaux(_).
true;
```

```
19 true.
20
21 animaux(poisson).
22 true.
23
24 grandpere(francois, paul).
25 true.
```

Les requêtes et les réponses obtenues sont donc les suivantes :

- Qui est la père de Jean ? -> Paul est le père de Jean.
- Qui sont les fils de Paul ? -> Jean est le fils de Paul.
- Qui est sa propre mère ? -> Personne.
- Quelles sont les combinaisons possibles de mères et filles connues par le programme ? -> Deux combinaisons (Marion et Jean ainsi que Françoise et Marion).
- Qui est la mère de Marion ? -> Françoise.
- Quels sont les animaux connus par la base de connaissances du programme ? -> Deux animaux (Chat et poisson).
- Est-ce qu'un poisson est un animal ? -> Oui.
- Est-ce que François est le grand-père de Paul ? -> Oui.

1.3 Les différences par rapport aux autres langages de programmation

En Prolog, il n'y a pas de variables au même titre que les autres langages, comme dit précédemment, il n'y a que des prédicats. Les seules variables utilisées sont celles au sein des prédicats, notamment lors des règles.

Les variables sont écrites en majuscules comme dit précédemment, et elles se réfèrent souvent à "X", "Y", voir l'exemple ci-dessous :

```
1 pere(paul,jean).
2
3 pere(X,jean).
4 X = paul.
```

Ce code permet de définir un fait, que Paul est le père de Jean, puis de faire une requête, demandant au système qui est le père de Jean en passant la varriable "X" et enfin sur la dernière ligne, le système trouve la valeur de "X" et la retourne.

Les variables sont écrites avec une majuscule au début contrairement aux autres langages de programmation.

Une autre différence notable étant que le point virgule courrament utilisé par les autres langages est remplacé ici par un point, une toute autre syntaxe est utilisée pour le Prolog.

Les différences syntaxiques peuvent être retrouvées dans la section 2, page page 5, pour plus d'informations.

Le langage Prolog ne possède par ailleurs pas les codes habituels, comme les boucles ou les conditions.

Il faut par ailleurs noter qu'habituellement, la base de connaissance, qui est composée des faits et des règles du programme, est enregistrée dans un fichier avant d'utiliser des requêtes dessus.

1.4 Les apports du Prolog

Le Prolog a permis de nombreuses avancées technologiques, il est la base de nombreux langages comme ceux parmi la liste ci-dessous :

- Cosmos
- Lambda Prolog
- Mercury
- Pilog

2 Les commandes essentielles en Prolog

Partie destinée aux membres du groupe pour avoir un regroupement des fonctionnalités et des connaissances en Prolog.

Voici un regroupement des différentes commandes et syntaxes, utilisées en Prolog :

Les syntaxes :

- **"."**, - Utilisé pour terminer une ligne.
- **";"**, - Correspond au mot "ou", et est la touche à appuyer dans le cas où le programme retourne plusieurs résultats.
- **","**, - Correspond au mot "et" en Prolog.
- **" :-"**, - Équivalent du mot "si" .
- **"Variable" ou "V"**, - Utilisation d'une variable, commençant toujours par une majuscule.
- **"constante" ou "c"**, - Utilisataion d'une constante, ou déclaration d'une règle par exemple.
- **"!"**, - Permet d'arrêter la requête lorsque le programme retourne une valeur, sans obtenir les autres.
- "_", - Dans une requête, permet d'obtenir toutes les occurences.
- **"\+" ou "not(X)"**, - Signifie la négation, pas dans le sens ou l'argument est faux, mais où celui-ci ne peut pas être prouvé.

Les fonctionnalités :

- **"consult(nomDuFichier)."**, - Permet d'ouvrir un fichier .pl dans l'interpréteur Prolog.

— **"listing."**, - Permet d'afficher le code source de la base de connaissances du programme courant.

— **"halt."**, - Permet de sortir de l'interpréteur Prolog.

— **"animal(chat)."**, - Exemple de fait, stipulant qu'un chat est un animal.

— **"animal(_)."**, - Exemple de requête, demandant s'il existe des animaux, le programme renverra "true" pour chaque occurence d'un animal.

— **"pere(_ , _)."**, - Exemple de requête, demandant toutes les combinaisons possibles de la base de connaissances à propos des pères.

— **"different(X, X, _) :- !, fail."**, - Exemple de règle où on établit que différent est faux si le premier et le deuxième argument sont indentiques, tandis qu'on ne s'occupe pas du dernier argument.

— **"pere(X, jean)."**, - Exemple de requête, demandant si son père existe, qui est le père de Jean.

3 Présentation d'un exemple

Voici la base du programme du projet réalisé à l'aide de l'IDE Pyzo et du langage Python :

```
from pyswip import Prolog
prolog = Prolog()

# Faits

prolog.assertz("not(traversable(mur))")
# Définit qu'un mur n'est pas traversable.

prolog.assertz("not(transparent(mur))")
# Définit qu'un mur n'est pas transparent (on ne peut pas voir au travers).


prolog.assertz("not(traversable(fenetre))")
# Définit qu'une fenetre n'est pas traversable.

prolog.assertz("transparent(fenetre)")
# Définit qu'une fenetre est transparente (on peut voir au travers).


prolog.assertz("conduit(conduiteEau)")
# Définit qu'une conduite d'eau est un conduit.

prolog.assertz("conduit(conduiteGaz)")
# Définit qu'une conduite de gaz est un conduit.

prolog.assertz("conduit(conduiteElect)")
# Définit qu'une conduite d'électricité est un conduit.

prolog.assertz("transporteEau(conduiteEau)")
# Définit qu'une conduite d'eau transporte de l'eau.

```

```
prolog.assertz("transporteGaz(conduiteGaz)")
# Définit qu'une conduite de gaz transporte du gaz.

prolog.assertz("transporteElect(conduiteElect)")
# Définit qu'une conduite d'électricité transporte de l'électricité.


prolog.assertz("meuble(lit)")
# Définit qu'un lit est un meuble.

prolog.assertz("sertADormir(lit)")
# Définit qu'un lit sert à dormir.




# Règles

prolog.assertz("espace(X) :- mur(Y)")
# X est un espace s'il possède un mur.

prolog.assertz("mur(X) :- not(traversable(X)), not(transparent(X))")
# X est un mur s'il n'est pas traversable ni transparent.

prolog.assertz("fenetre(X, Y) :- not(traversable(X)), transparent(X), mur(Y)")
# X est une fenetre sur un mur Y si X n'est pas traversable mais est transparent, et que Y est un mur.


prolog.assertz("arriveeEau(X) :- conduit(X), transporteEau(X)")
# X est une arrivée d'eau si X est une conduite et que X transporte de l'eau.

prolog.assertz("arriveeGaz(X) :- conduit(X), transporteGaz(X)")
# X est une arrivée de gaz si X est une conduite et que X transporte du gaz.

prolog.assertz("arriveeElect(X) :- conduit(X), transporteElect(X)")
# X est une arrivée d'électricité si X est une conduite et que X transporte de l'électricité.




prolog.assertz("cuisine(X) :- espace(X), arriveeEau(W), arriveeGaz(Z), arriveeElect(E)")
# X est une cuisine si X est un espace possédant une arrivée d'eau, de gaz et d'électricité.

prolog.assertz("salleDeBain(X) :- espace(X), arriveeEau(W), arriveeElect(E)")
# X est une salle de bain si X est un espace et qu'il possède une arrivée d'eau et d'électricité.

prolog.assertz("wc(X) :- espace(X), arriveeEau(W), arriveeElect(E)")
# X est un WC si X est un espace et qu'il possède une arrivée d'eau et d'électricité.

prolog.assertz("chambre(X) :- espace(X), arriveeElect(E), lit(L)")
# X est une chambre si X est un espace et qu'il possède une arrivée d'électricité et un lit.


```

```
84
85 prolog.assertz("lit(X) :- meuble(X), sertADormir(X)")
86 # X est un lit si X est un meuble et qu'il sert à dormir
```

Listing 1 – Base du projet