Prise en main de SWI-Prolog

Introduction

Prolog est un langage de *programmation logique*. Ceci veut dire qu'un programme ne se présentera pas sous forme d'une suite d'instruction (comme dans la *programmation impérative*), mais sous la forme d'un certain nombre d'"affirmations sur le monde", formant une *base de connaissances*.

L'utilisation du programme se fera alors en *interrogeant* cette base de connaissances. Prolog dispose d'un puissant "moteur" lui permettant de déduire certains faits à partir de ceux que contient sa base de connaissances.

Une autre manière de voir Prolog est de dire qu'il s'agit d'un système expert générique en chaînage arrière.

Au cours des TP's qui viennent, nous étudierons en détails les différentes caractéristiques de Prolog. Pour l'instant, nous nous contenterons d'une approche "expérimentale" pour nous familiariser avec le langage.

Nous utiliserons pour ce faire SWI-Prolog, qui est une implémentation libre (sous licence LGPL) du langage Prolog.

1 Installation

Commencez par installer SWI-Prolog en récupérant l'installeur pour votre plate-forme sur `http://www.swi-prolog.org/download/stable` ou par un commende du type `sudo apt-get install swi-prolog`.

2 Base de faits

Dans un fichier texte, entrez le code suivant[1] :

```
man(john).
woman(frida).
woman(nelly).
duck(donald).
loves(john,frida).
loves(frida,donald).
loves(nelly,donald).
loves(donald,nelly).
```

[1] Faites attention à la casse ! pas de majuscule pour les noms ! (nous verrons pourquoi par la suite)

Un tel fichier s'appelle *base de faits*, parce qu'il regroupe des faits que l'on déclare être vrais.

Sauvez le fichier sous le nom `base1.pl`.

Si l'extension `.pl` est associée au compilateur `swi-prolog`, vous pouvez simplement double-cliquer sur le fichier ci-dessus pour lancer Prolog[2]. Sinon, trouvez le nom de l'exécutable (probablement `pl` ou `swipl` sous Unix et `plwin` sous Windows) et tapez une commande du type

```
swipl -s base1.pl
```

Prolog vous indique qu'il a chargé et compilé votre fichier et vous présente son invite de commande.

```
% base1.pl compiled 0.00 sec, 2,964 bytes
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.55)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?-
```

Si vous avez fait une erreur de syntaxe, celle-ci sera signalée à ce moment-là. Attention cependant : l'erreur sera affichée tout en haut, *avant* la ligne commençant par `% base1.pl`... si on n'est pas averti, on peut facilement la manquer !

3 Requêtes

Lorsque vous vous trouvez face à l'invite de Prolog, vous pouvez lui poser des questions. Dans la terminologie Prolog, on les appelle des *requêtes*.

Entrez

```
?- woman(nelly).
```

Si tout va bien, Prolog devrait vous répondre

true.

et revenir à l'invite de commandes.

En cas de problème, vérifier que vous avez bien recopié la base de faits et la requête. Vérifiez en particulier

1. Que vous n'avez pour l'instant utilisé aucune majuscule.
2. Que tout vos faits et votre requête *se terminent par un point*.

Testez de la même manière les autres faits de votre base.

Essayez maintenant la requête

```
?- woman(laura).
```

Que répond Prolog ?

Vous constatez donc que Prolog considère que tout ce qu'il ne peut pas déduire de la base de faits est faux (même si à nos yeux cela semble vrai !). En particulier, il ne répondra jamais "I don't know".

Essayez encore

[2] L'extension `.pl` est également souvent associée au code source Perl.

```
?– cat(mistigri).
```

La situation est donc différente si on se trouve face à une “propriété” connue mais dont on ne sait pas si elle est vraie ou face à une “propriété” inconnue. Relevons au passage que les “propriétés” définies dans la base de fait s’appellent des *prédicats*.

Vous pouvez maintenant quitter Prolog :

```
?– halt.
```

4 Variables

Jusqu’à maintenant, vous n’êtes probablement pas très impressionnés par la puissance de Prolog. Pour obtenir des choses plus intéressantes, nous allons introduire les variables.

Relancez Prolog en ouvrant votre fichier `base1.pl`.

À l’invite de commandes, tapez

```
?– man(X).
```

Prolog vous répond

`X = john`.

et revient à l’invite de commande. Ce passionnant dialogue pourrait se lire

– Connais-tu un X qui soit un homme ?

– Oui, John !

Ou plus précisément :

– “Connais-tu une valeur de X pour laquelle on peut prouver l’affirmation `homme(X)` ?”

– “Oui, cela marche avec `X=John`.”

Bon, jusque là, rien de bien passionnant. Essayons maintenant

```
?– woman(X).
```

Prolog vous répond

`X = frida`

et *ne revient pas immédiatement* à l’invite. Tapez un espace, Prolog affiche `X = nelly`. et revient cette fois-ci à l’invite de commandes.

Que s’est-il passé ?

1. Prolog a identifié `X` comme étant une variable. Ceci n’est pas dû à son nom, mais au fait que l’identificateur commence par une majuscule. On aurait tout aussi bien pu entrer Var, HELLO... ou Mary[3] ! Une autre possibilité pour les noms de variables est de commencer par un soulignement : `_var`, `_Toto`, ... (_ tout seul ayant une signification particulière.)
2. Il a cherché une affectation de la variable `X` permettant de vérifier l’affirmation `woman(X)`.
3. En ayant trouvé une, il l’affiche...
4. ... mais comme il a constaté qu’il pourrait y avoir d’autres possibilités, il attend votre avis (notez *l’absence de point* après cette première réponse !). Si cette information vous suffit, tapez `enter` (ou `a`, ou `c`) et Prolog arrêtera sa recherche.
5. Si vous voulez voir les alternatives restantes, tapez `;` (ou `n`, `r`, barre d’espacement ou tabulation) et Prolog continuera sa recherche. Retour au point 2 jusqu’à ce qu’il n’y ait plus d’autre solution...

[3] Et on comprend maintenant pourquoi il ne faillait pas mettre de majuscule aux noms propres !

Essayez maintenant, sur la base de ce que nous avons vu, de répondre aux questions suivantes :

1. De qui John est-il amoureux ?
2. Qui est amoureux de Donald ?

5 Conjonction

Nous nous sommes pour l'instant limités à satisfaire un seul but à la fois, mais on peut aussi tenter de les combiner : si l'on entre plusieurs buts séparés par des virgules, Prolog essaiera de tous les satisfaire simultanément. Ainsi,

```
?- woman(X),loves(john,X).
```

nous donne bien le `X = frida` ; que nous espérions, et

```
?- woman(X),man(X).
```

nous donne heureusement `No`.

Essayez maintenant de trouver s'il existe un amour réciproque dans notre monde imaginaire.

6 Règles

Jusque là, Prolog ressemble à une sorte de base de données, mais en moins pratique et plus lent ! Pour en faire un langage de programmation, il nous manque un ingrédient essentiel : les règles.

Ouvrez à nouveau votre éditeur de texte préféré et chargez le fichier `base1.pl`. À la suite des lignes existantes, rajoutez deux lignes pour obtenir le résultat suivant :

```
man(john).
woman(frida).
woman(nelly).
duck(donald).
loves(john,frida).
loves(frida,donald).
loves(nelly,donald).
loves(donald,nelly).

% and here is our first rule:
sad(X) :- loves(X,Y), loves(Y,Z).
```

L'avant-dernière ligne est un commentaire, introduit par le caractère %.

La dernière ligne est une règle et signifie que la partie gauche (avant le `:-`) est vraie si la partie droite est vérifiée.

Dans le cas particulier, cela nous donne quelque chose comme : `X` est triste si `X` aime `Y` et `Y` aime `Z`. À noter que le nouveau prédicat `sad` est cette fois défini par une règle et non par un fait. On peut aussi mélanger les deux types de définitions pour un même prédicat.

Enregistrez votre fichier, relancez Prolog sur cette nouvelle base[4] et essayez de l'interroger sur l'existence d'une personne triste.

[4] Si vous êtes attentifs, vous remarquerez un avertissement à la compilation de cette nouvelle base. Stockez cette information dans un coin de votre tête, nous y reviendrons au paragraphe 10.

```
sad(X).
```

Si tout se passe normalement, Prolog vous proposera bien plus de réponses que prévu : non seulement John et Frida sont tristes, ce qui est bien légitime, mais aussi Donald et Nelly.

7 Débogage

Pour comprendre ce qui se passe, nous allons utiliser les capacités de débogage de SWI-Prolog[5]. Entrez

```
trace, sad(X).
```

Ceci va donc lancer la requête `sad(X).`, mais en demandant en plus une trace d'exécution. Prolog vous répondra quelque chose comme `Call: (8) sad(_G157)?`, ce qui signifie qu'il a projet d'appeler le prédicat `sad` avec une variable comme argument[6].

Pour savoir quelles sont vos possibilités à ce moment précis, tapez "h". Prolog vous donnera alors la liste des commandes disponibles. Pour l'instant, nous désirons simplement continuer l'exécution : tapez la barre d'espacement (ou *enter*).

Continuez à suivre pas à pas l'exécution de votre programme en frappant la barre d'espacement chaque fois que nécessaire, jusqu'à la fin de l'exécution[7].

Le debug en mode texte est un plaisir pour les yeux, mais SWI-Prolog (dans sa version pour Windows seulement) propose aussi une version graphique du débogueur. Dans le menu "Debug" choisissez "Graphical Debugger" (ou tapez simplement `guitracer.`), puis relancez la commande ci-dessus. Une belle fenêtre apparaît vous permettant de suivre l'exécution de votre programme[8].

Bon... avez-vous maintenant compris ce qui cloche dans notre programme ?

8 Correction du programme

Au fait, notre problème n'est pas surprenant : nous avons employé des *noms* de variables différents pour `X` et `Z`, mais rien n'empêche que ces variables prennent la même *valeur* (comme d'ailleurs dans les autres langages de programmation).

Dans votre éditeur, vous pouvez donc corriger la dernière ligne de votre programme pour exiger que les valeurs de `X` et `Z` soient différentes :

```
sad(X) :- loves(X,Y), loves(Y,Z), X\=Z.
```

Pour recharger ce nouveau programme, vous avez plusieurs solutions *à choix* :

1. Quitter Prolog et double-cliquer à nouveau sur `base2.pl`.
2. Utiliser la commande : `consult('o:/path/to/base2.pl').`[9]`...`
3. ... ou sa forme équivalente `['o:/path/to/base2.pl'].`
4. (Sous Windows) Dans le menu "file", choisir "consult" puis sélectionner votre fichier.

[5] Dans ce cas particulier, il est aussi possible de deviner ce qui se passe sans faire de débogage, mais nous le ferons tout de même pour les besoins de la démonstration !

[6] Le fait que la variable a été renommée ne retiendra pas notre attention pour l'instant.

[7] À ce moment-là, Prolog restera en mode debug. Ceci n'est pas dérangeant, mais si vous désirez retrouver votre invite de commande habituelle, tapez simplement `nodebug`.

[8] Les puristes de la console se dépêcheront donc d'entrer `noguitracer` pour revenir à un debug en mode texte !

[9] Notez l'utilisation du séparateur unix (/) au lieu du séparateur Windows (\).

5. (Sous Windows) Dans le menu “file” toujours, choisir “reload modified files”.

Testez votre nouvelle base : les amours partagées ne sont plus tristes !

Forts de ces nouvelles connaissances, essayez de rajouter à votre programme une règle disant que deux personnes aimant une même tierce personne sont jalouses.

9 Exercice

Le moment est venu de mettre à l’épreuve l’ensemble des connaissance que vous avez acquises au cours de ce document.

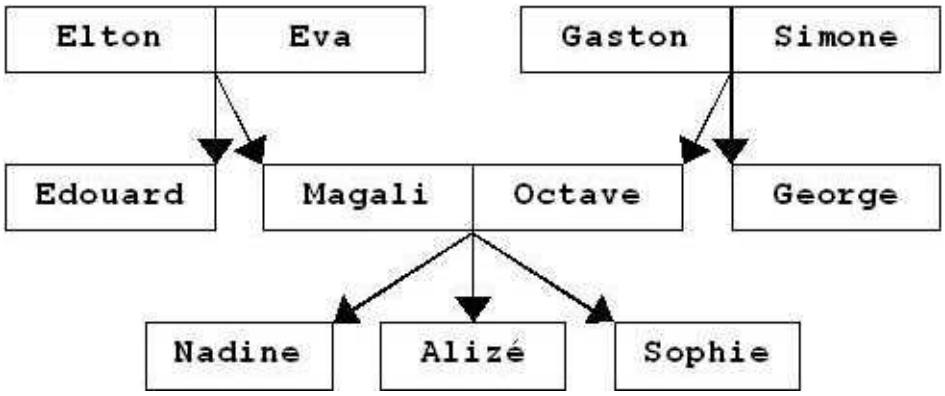

FIG. 1: Un arbre généalogique

La figure 1 présente un arbre généalogique.

1. Écrivez une série de *faits* définissant les prédicats `homme`, `femme` et `parent` pour modéliser cet arbre. `homme(X)` signifiera que `X` est un homme, `femme(X)` que `X` est une femme, et `parent(X,Y)` que `X` est un parent (père ou mère) de `Y`.
2. Rajoutez à votre base des *règles* pour définir les prédicats suivants (pour tout ces prédicats, on conviendra que `Préd(X,Y)` signifie “X est le/un Préd de Y”. Par exemple, `enfant(X,Y)` signifiera “X est un enfant de Y”) :
   a) `enfant(X,Y)`
   b) `fils(X,Y)`
   c) `fille(X,Y)`
   d) `pere(X,Y)`
   e) `mere(X,Y)`
   f) `grand_parent(X,Y)`
   g) `grand_pere(X,Y)`
   h) `grand_mere(X,Y)`
   i) `petit_enfant(X,Y)`
   j) `petit_fils(X,Y)`
   k) `petite_fille(X,Y)`

l) `frere(X,Y)`
m) `soeur(X,Y)`
n) `oncle(X,Y)`
o) `tante(X,Y)`

Pensez à tester vos règles au fur et à mesure !

10 Variable anonyme

Supposons maintenant que l'on veuille écrire un prédicat qui soit vrai si son argument est un père, sans s'intéresser à qui est l'enfant. On va naturellement essayer quelque chose comme

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

Ce qui veut dire quelque chose comme "X est père (tout court) si c'est le père de quelqu'un"...
Si vous essayez de tester ce prédicat, vous constaterez plusieurs choses :

1. Ça marche !
2. Cela montre au passage que Prolog accepte de définir deux prédicats de même nom mais de signatures différentes.
3. Prolog affiche un *Warning* à la compilation de notre fichier :

```
Warning: (genea.pl:76):
        Singleton variables: [Y]
% genea.pl compiled 0.00 sec, 7,620 bytes
Welcome to SWI-Prolog (Multi-threaded, Version 5.1.13)
[...]
```

Que signifie cet avertissement ? Simplement que la variable `Y` n'apparaît qu'une seule fois dans la règle que nous venons d'écrire.

Ceci n'est pas forcément un erreur, mais peut venir de plusieurs choses :

1. Nous avons peut-être fait une erreur de raisonnement, qui nous pousse à définir une variable que nous n'utilisons pas plus loin (ce serait la situation analogue à une définition de variable en `C` sans l'utiliser après). Ici, ce n'est pas le cas.
2. Nous aurions aussi pu faire une faute de frappe dans le nom d'une variable. Ce n'est pas le cas non plus.
3. Enfin, il se peut que nous ayons besoin d'une variable pour des raisons syntaxiques ("ici, je dois mettre quelque chose"), mais que la valeur de cette variable n'ait aucune importance à nos yeux. C'est visiblement ce qui se passe ici.

Pour éviter les cas 1 et 2, et pour s'économiser du travail inutile dans le cas 3, Prolog nous signale lorsqu'une variable n'apparaît qu'une seule fois dans une règle.

Si nous sommes bien dans le troisième cas, il est d'usage de signaler à Prolog que la variable en question n'est là que comme "placeholder". Comment ? en utilisant la *variable anonyme*, qui se résume à un soulignement (_). Notre règle devient alors

```
pere(X):-pere(X,_).
```

Testons... et oh ! miracle, la règle fonctionne toujours et le *warning* a disparu !

Remarque Chaque occurrence de la variable anonyme est indépendante. Par exemple,

```
pere(X,X).
```

cherchera une personne qui est son propre père (ce qui devrait normalement nous donner la réponse `fail`), alors que

```
pere(_,_).
```

répondra `Yes` dès qu'on pourra trouver deux personnes – n'importe lesquelles – dont l'une est le père de l'autre.