École Nationale Supérieure des Mines de Saint-Étienne
cycle I.C.M. / Toolbox I.A. 2014-2015
Philippe.Beaune@emse.fr

Prise en main du langage Prolog

vendredi 19 septembre 2014 - 13h30/16h45

1. Objectif et moyens

L'objectif de cette séance est de prendre en main Prolog : sa syntaxe, sa *façon de faire*, et son environnement. L'environnement choisi est celui de SWI-Prolog (http://www.swi-prolog.org/), un interprète (et compilateur) Prolog distribué sous licence LGPL. Cet environnement est disponible pour différentes plateformes (*MSWindows*, *Mac OS X*, *Linux*, …). De plus la syntaxe retenue est, en partie, conforme à la norme ISO (http://pauillac.inria.fr/~deransar/prolog/docs.html). SWI-Prolog est installé sur toutes les machines des salles de T.P. sous l'environnement **Linux**.

A l'issue de cette séance vous devrez pouvoir aborder et tenter de résoudre des problèmes simples avec Prolog, ce que nous ferons ensemble lors des séances suivantes. Néanmoins, une connaissance approfondie de ce langage vous demandera non seulement de la pratique, comme pour tout langage informatique, mais aussi et surtout de la lecture, notamment sur les aspects théoriques de ce langage, et les ouvrages ne manquent pas !

Cette séance est organisée en 5 *séquences* supposées durer chacune environ 30 minutes : les 5 séquences mises bout à bout tiennent donc forcément en 3h, mais …

2. Lancement de Prolog et premières interactions

Pour lancer Prolog, et avoir quelques extensions fenêtrées, utilisez la commande **`swipl`**. Vous devriez obtenir quelques informations puis l'invite de commande suivante : « **`?-`** » Toutes les commandes données à l'interprète (en fait ce ne sont que des prédicats ou compositions de prédicats) doivent se terminer par « **`.`** ». Essayez « **`help.`** ». Une fenêtre a dû surgir : gardez la car elle vous servira tout au long du T.P. De plus l'interprète vous a rendu la main sans omettre d'écrire « **`true.`** », ce qui signifie qu'il a réussi à *prouver* le prédicat. Pour savoir ce qu'il répond quand ce n'est pas le cas essayez « **`1=2.`** ».

Pour ajouter un fait ou une règle, vous pouvez utiliser les prédicats prédéfinis suivants : « **`assert/1`** », « **`asserta/1`** » et « **`assertz/1`** ». Le « **`/1`** » signifie que ces prédicats utilisent 1 seul argument. Regardez dans la documentation ce que font ces prédicats et essayez « **`assert(leMeilleurEst(philippe)).`** », ce qui signifiera pour nous que Philippe est le meilleur. D'ailleurs vous pouvez le demander à l'interprète : « **`leMeilleurEst(philippe).`** » : vous voyez bien que c'est vrai !

Rappel : les noms de prédicats et de constantes commencent par des minuscules. Et les noms de variables commencent par des majuscules. Donc pour demander à l'interprète qui est le meilleur, il suffit de lui donner « **`leMeilleurEst(Qui).`** », et il essaiera de le prouver en instanciant **`Qui`**.

Entrez d'autres meilleurs et re-testez tout ça … Il trouve un premier meilleur, mais il ne vous rend pas la main, car non content de l'avoir prouvé 1 fois, il vous propose de voir s'il peut le prouver une autre fois : pour qu'il essaie entrez juste un espace. Pour lui demander de s'arrêter là, entrez juste « **`.`** ».

Il y a plus facile pour donner des règles et des faits à l'interprète : saisissez les dans un fichier suffixé « **`.pl`** » et utilisez le prédicat prédéfini « **`consult/1`** » (regardez dans la documentation la syntaxe abrégée de ce prédicat). Pour le tester, créez un fichier contenant ce code et donnez le à l'interprète :

```
inc(X,Y) :- Y is X + 1.
```

Vous pouvez maintenant tester « **`inc(3,4)`** », puis « **`inc(3,X)`** » : ça marche. Mais si vous testez « **`inc(X,4)`** » : problème. Pourquoi ? Regardez dans la documentation ce qui est dit sur l'opérateur « **`is`** », et aussi sur « **`=`** », « **`\=`** » et « **`=:=`** ». Essayez la même règle en remplaçant « **`is`** » par « **`=`** » : pour éditer un fichier vous pouvez utiliser le prédicat « **`edit/1`** » (un éditeur *à la Emacs*). Pour prendre en compte les modifications d'un fichier déjà consulté, utilisez « **`make/0`** ». Vous pouvez ajouter (et ce sera apprécié à sa juste valeur par tous les lecteurs humains de vos programmes) des commentaires dans vos fichiers : entre « **`/*`** » et « **`*/`** »

N.B. : les prédicats prédéfinis de la famille « **assert/1** » peuvent aussi avoir pour argument une règle. Et dans un même fichier externe vous pouvez saisir plusieurs règles et plusieurs faits : la seule restriction est que **toutes les clauses d'un même prédicat soient regroupées**.

Pour tracer ce que fait Prolog, vous pouvez utiliser « **trace/0** ». Pour en sortir : « **notrace/0** ». En sortie de trace, éventuellement vous vous trouvez en mode *debug* : pour en sortir utilisez « **nodebug/0** ». Enfin, pour quitter proprement Prolog : « **halt.** ».

3. Généalogie

Entrez plusieurs faits décrivant des personnes par leur prénom (prédicats « **homme/1** » et « **femme/1** ») puis quelques liens parents/enfants entre ces personnes (prédicat « **parent/2** ») de façon à obtenir une arborescence d'au moins 4 niveaux.

Entrez les règles décrivant les prédicats suivants (testez les au fur et à mesure) : « **enfant/2** », « **pere/2** », « **fils/2** » , « **fille/2** » « **mere/2** » , « **frere/2** » , « **soeur/2** » , « **grand_parent/2** », « **grand_pere/2** » , « **grand_mere/2** » , « **oncle/2** » , « **tante/2** » , « **grand_oncle/2** », « **grand_tante/2** » et enfin « **cousin_germain/2** » et « **cousine_germaine/2** ».

Ne passez pas plus de 20 minutes sur cette séquence (arrêtez avant la fin si besoin).
Attention aux doublons : par exemple *Toto* ne doit pas pouvoir être son propre frère !

4. Ça boucle !

Maintenant, considérez l'énoncé suivant : « *A est l'aïeul de X si A est l'aïeul de Z et que Z est parent de X* ». Codez exactement cela en Prolog avec un prédicat « **aieul/2** » et testez. Que se passe-t-il ? Tracez pour voir : en mode trace, pour sortir de la boucle de résolution, entrez « **a** ». Corrigez en ajoutant une clause au prédicat « **aieul/2** » et testez en demandant tous ceux et toutes celles dont celui qui est en haut de votre arborescence est l'aïeul : que se passe-t-il encore ? Corrigez …

S'il vous reste du temps codez la factorielle (*grand classique de la récursivité*). Sinon, faites le chez vous *(essayez vraiment de le faire vous-même, ne copiez pas bêtement n'importe quoi trouvé sur le web, ça n'a pas d'intérêt)*.

5. Traitements de listes

Codez les prédicats suivants :

- **liste/1** : réussit si l'argument est une liste
- **prems/2** : réussit si le 1er argument est le 1er élément de la liste passée en 2nd argument
- **membre/2** : réussit si le 1er argument est élément de la liste passée en 2nd argument
- **der/2** : réussit si le 1er argument est le dernier élément de la liste passée en 2nd argument
- **concat/3** : réussit si la concaténation des 2 listes passées en arguments 1 et 2 est la liste passée en argument 3
- **pref/2** : réussit si la liste passée en argument 1 est le préfixe de la liste passée en argument 2
- **suff/2** : réussit si la liste passée en argument 1 est le suffixe de la liste passée en argument 2
- **tri/2** : réussit si la liste passée en argument 2 est la liste triée passée en argument 1. Vous ferez un *tri rapide* en prenant le 1er élément de la liste comme pivot. Pour prendre en compte l'éventualité où le 1er argument est une variable libre, vous utiliserez le prédicat prédéfini « **var/1** ».
- facultatif (s'il vous reste un peu de temps) : **renv/2** : réussit si la liste passée en argument 2 est la liste inversée passée en argument 1 (même remarque que ci-dessus pour la variable libre)

6. Amusement

Codez la conjecture de Syracuse et s'il reste du temps : la fameuse conjecture de Goldbach.