Feuille de TD n° 6 : ProLog

Exercice 1 : Préalables

TOUTE COMMANDE DANS PROLOG DOIT SE TERMINER PAR UN POINT.

L'interface de ProLog vous permet d'y faire des **requêtes** lorsque la main vous est donnée (curseur derrière `?-`).

POUR REPRENDRE LA MAIN (si vous l'avez perdue), appuyez sur `ctrl-c` puis `b`.

LES VARIABLES COMMENCENT PAR UNE LETTRE MAJUSCULE ALORS QUE LES CONSTRUCTEURS (DE TERMES ET DE PRÉDICATS ATOMIQUES) COMMENCENT PAR UNE MINUSCULE.

1. Cherchez le fichier d'exemple `likes.pl` dans le repertoire de ProLog (program files... `pl` ... `Demo`) et copiez-le dans votre repertoire favori.
2. Ouvrez le fichier avec ProLog, ou lancer ProLog et ouvrez le fichier par la commande `consult` (du menu).
3. Lancez l'editeur par la commande `edit(likes)`, et regardez le code, qui contient une bibliothèque de **clauses**.
   Les **requêtes** sont alors faites dans l'interface principale de ProLog.
   Faites des tests de requêtes sur les prédicats contenus dans `likes.pl` (des exemples de requêtes sont donnés dans les commentaires du fichier `likes.pl`).
4. Téléchargez le fichier
   `http://www.lix.polytechnique.fr/~lengrand/Work/Teaching/ESIEA/pack.txt`
   et placez-le dans votre repertoire favori.
5. Renommez-le en `pack.pl`
6. Ouvrez le fichier avec ProLog, ou lancer ProLog et ouvrez le fichier par la commande `consult` (du menu).
7. Pour chaque exercice suivant, creez un fichier `.pl`
   Pour recharger un fichier `exo.pl` modifié vous utiliserez la commande `consult(exo)`.

Exercice 2 : Arithmétique

La syntaxe des termes contient `0`, d'arité 0, et `s`, d'arité 1.

1. Programmez le prédicat `pl`, d'arité 3 (noté `pl/3`), par récurrence sur leur deuxième argument.
   L'idée est que `pl(t,u,v)` est valide si et seulement si $t + u = v$, etc...
   Pour cela, transformez les deux équations ci-dessous en clauses ProLog :

$$\begin{array}{ll} X + 0 & = X \\ X + s(Y) & = s(X + Y) \end{array}$$

2. Faites des tests dans l'interface de ProLog.
   ATTENTION : votre programme manipulera des entiers représentés sous la forme `s(...s(0)...)`.
   Tout entier représenté avec les chiffres du clavier `1,2,3,4,5,6,7,8,9` sera à proscrire.
   MAIS, pour vous simplifier les notations, vous pourrez utiliser deux prédicats définis dans `pack.pl` :
   le prédicat `n/2`, tel que `n(a,b)` est valide (a est une chaîne de caractères telle que `''10''`, b est un terme de la forme `s(...s(0)...)`) si b encode a,
   et le prédicat `print/2`, tel que `print(a,b)` est valide (a est est un terme de la forme `s(...s(0)...)` et b est une chaîne de caractères) si b encode a.
3. Programmez le prédicat `fois/3` par récurrence sur leur deuxième argument, grâce aux équations :

$$\begin{array}{ll} X \times 0 & = 0 \\ X \times s(Y) & = (X \times Y) + X \end{array}$$

   Faites des tests.

4. Programmez le prédicat `puiss/3` pour la puissance.
   Faites des tests.
5. On rajoute à la syntaxe les constructeurs de termes `a/2`, `m/2`, et `p/2` (intuitivement, ils représentent les opérateurs d'addition, de multiplication et de puissance). Programmez le prédicat `eval/2` qui évalue une expression contenant ces constructeurs en une expression ne les contenant pas.
   Par exemple, on doit pouvoir montrer `eval(a(s(0),s(0)),s(s(0)))`.
   `eval/2` fera bien sûr appel aux prédicats `pl/3`, `fois/3`, et `puiss/3` des questions précédentes.
   Faites des tests.

Exercice 3 : Fibonacci

1. Programmez la suite de Fibonacci par le prédicat `fib/2`.
   Rappel : $fib(0) = 0$, $fib(1) = 1$, $fib(n+2) = fib(n) + fib(n+1)$
2. Que pouvez-dire sur le temps de calcul de votre programme ?

Exercice 4 : Division et nombres premiers La syntaxe des termes contient `0`, d'arité 0, et `s`, d'arité 1.

1. Programmez le prédicat `ppetit/2`, tel que `ppetit(a,b)` est valide si a est strictement plus petit que b.
2. Programmez le prédicat `div/4` pour la division euclidienne : `div(a,b,c,d)` est valide si la division entière de a par b donne pour quotient c et pour reste d. On utilisera les équations

$$\begin{array}{ll} a & = (0 \times b) + d \qquad \text{si } a < b \\ a & = ((n+1) \times b) + d \qquad \text{si } a \geq b \text{ et } (a-b) = (n \times b) + d \end{array}$$

3. Programmez le prédicat `est_pas_premier/1` qui soit valide si l'argument représente un nombre qui n'est pas premier.

Exercice 5 : Notation polonaise
Nos termes représentent maintenant les formules de la logique propositionnelle. On a donc comme constructeurs `atom/1,et/2,ou/2,imp/2,neg/1` (`atom` prendra comme argument un entier, histoire de se donner un ensemble infini dénombrable de formules atomiques).

1. Programmez l'algorithme produisant la notation polonaise d'une formule.
2. Programmez l'algorithme produisant une formule à partir de sa notation polonaise.

Exercice 6 : Formes normales de formules
Toujours avec la même signature,

1. Programmez un algorithme de mise en forme normale de négation, par le prédicat `fnn/2`
2. Programmez un algorithme de mise en forme normale conjonctive, par le prédicat `fnc/2`

Exercice 7 : G3
Toujours avec la même signature,

1. Programmez un algorithme de recherche de preuve dans le système G3 propositionnel.