Agrégation de Mathématiques option Informatique
TP6

Loïg Jezequel

Mardi 24 mai 2012

1 Rappels sur Prolog, SWI Prolog

Pour lancer l'interpréteur Prolog il faut taper `swipl` dans un terminal. Normalement, après diverses informations sur l'interpréteur, le symbole `?-` doit s'afficher en début de ligne.

Rappel. *Une clause de Horn est une disjonction de formules atomiques dont au plus une est positive. Un programme logique est un ensemble fini de clauses de Horn. En prolog on note* `A:-B,C,D.` *la clause* $A \Leftarrow B \wedge C \wedge D$. *On peut aussi définir des faits :* `A.` *signifie* A *est vraie.*

Vous écrirez vos programmes logiques dans un fichier de votre choix. Pour envoyer un programme à l'interpréteur on utilise la syntaxe suivante :

```
?- ['mon_chemin/mon_fichier'].
```

Rappel. *Pour utiliser un programme logique* P *on spécifie un but* b*, qui est une formule (close ou non), l'interpréteur essaye de prouver* $P \models b$. *Il peut boucler ou rendre une réponse. S'il ne boucle pas, deux cas sont possibles : si* b *est close, la réponse est* vrai *ou* faux*, sinon la réponse est une substitution des variables libres de* b *qui permet de faire la preuve ou* faux *s'il n'existe pas de telle substitution.*

Pour donner un but à l'interpréteur il suffit de rentrer la formule correspondante, suivie d'un point.

```
?- ma_formule.
```

L'interpréteur affiche `true` ou `false` si la formule est close. Sinon il affiche `false` ou une substitution possible des variables libres de la formule. Dans ce cas il y a un choix à faire : « ; » pour demander la recherche d'une autre substitution ou « . » pour stopper la recherche de solutions.

Quelques autres notions utiles pour le TP

En Prolog il existe une structure de liste. On écrit une liste entre crochets, les éléments étant séparés par des virgules. Ainsi, `[]` est la liste vide et `[a,b,c]` une liste contenant les éléments a, b et c. Cette liste peut aussi s'écrire `[a|[b,c]]`, le signe « | » sépare donc la tête et la queue d'une liste.

Une notion importante est la coupure, notée « ! » en Prolog. Elle permet de contôler l'exploration de l'arbre de recherche des solutions, en supprimant certaines branches, empéchant ainsi leur exploration. L'appel d'une coupure présente dans la définition d'un prédicat supprime toutes les alternatives laissées par les construction des preuves des buts qui figurent à gauche de cette coupure et toutes les alternatives correspondant aux clauses listées après la clause où figure la coupure. Un exemple :

```
once(But) :- But, !.
```

permet de calculer une unique solution de `But` : une fois qu'une solution est trouvée, la coupure est appliquée, toutes les branches laissées par la preuve de `But` (menant potentiellement à d'autres solutions) sont ignorées. Si la recherche d'une preuve de `But` échoue, comme il n'y a pas de définition alternative du prédicat `once`, la recherche d'une preuve de `once(But)` échoue.

En prolog les opérations sur les entiers (+, x, mod...) sont définies. Cependant, dans ce TP nous ne les utiliserons pas. Un entier n sera, pour nous, le $n^{\text{ème}}$ successeur de 0. Par exemple, 3 s'écrira `s(s(s(0)))`.

Références

1. polycopié du cours *Programmation Logique et Calcul*, par Olivier Ridoux et Catherine Belleannée : http ://www.irisa.fr/lande/ridoux/ENS/PLC/
2. 99 problèmes de prolog : https ://staff.hti.bfh.ch/hew1/informatik3/prolog/p-99/
3. manuel de SWI Prolog : http ://www.swi-prolog.org/pldoc/refman/

2 Exercices

Exercice 1. Dire si un entier est pair ou non.

```
?- pair(s(0)).
false
```

Exercice 2. Trouver le dernier élément d'une liste.

```
?- dernier(X,[a,b,c]).
X=c
```

Exercice 3. Trouver l'avant dernier élément d'une liste.

```
?- avant_dernier(X,[a,b,c]).
X=b
```

Exercice 4. Trouver le $k^{\text{ème}}$ élément d'une liste.

```
?- element_numero(X,[a,b,c],s(0)).
X=a
```

Exercice 5. Donner le nombre d'éléments d'une liste.

```
?- nombre_elements(X,[a,b,c]).
X=s(s(s(0)))
```

Exercice 6. Dupliquer chaque élément d'une liste.

```
?- duplique(X,[a,b,c]).
X=[a,a,b,b,c,c]
```

Exercice 7. Retourner une liste.

```
?- miroir(X,[a,b,c]).
X=[c,b,a]
```

Exercice 8. Retirer les éléments identiques consécutifs d'une liste.

```
?- compresser(X,[a,a,b,c,c,c]).
X=[a,b,c]
```

Exercice 9. Implémenter la négation.

```
?- non(pair(s(0))).
true.
```

Exercice 10. Implémenter une structure conditionnelle

```
?- cond(pair(0),dernier(X,[a,b,c]),dernier(X,[d,e,f])).
X=c
```

3 Problème

On souhaite placer 8 dames sur un échiquier, de telle sorte qu'aucune dame n'en menace une autre. Proposer un programme trouvant les solutions de ce problème.