Représentation de la Connaissance
Répétition 3
10 octobre 2006

Astuce du jour

Voici quelques opérateurs intéressants à utiliser :

$X >= Y$	vrai si X est plus grand que ou égal à Y
$X =< Y$	vrai si X est plus petit que ou égal à Y
$X = Y$	vrai si X est syntaxiquement égal à Y
$X \backslash == Y$	vrai si X et Y sont syntaxiquement différents
$X =:= Y$	vrai si les valeurs de X et Y sont égales
$X = \backslash = Y$	vrai si les valeurs de X et Y sont différentes

Exemples d'utilisation :

$3 = 1 + 2.$	$-->$	No
$3 =:= 1 + 2.$	$-->$	Yes
$3 = \backslash = 1 + X.$	$-->$	Arguments are not sufficiently instantiated

PROLOG et les listes

Correction des exercices proposés

1. `inc_sublist`
Définir un prédicat `inc_sublist(Ls,Is)` vrai si et seulement si `Is` est la plus longue sous-liste croissante de la liste de nombres `Ls`. Dans le cas où `Is` n'est pas complètement instancié mais que `Ls` l'est, on demande d'écrire deux versions différentes :

- La première renverra une seule instance de `Is` telle que `inc_sublist(Ls,Is)` est vrai.
- La deuxième renverra une et une seule fois chacunes des instances de `Is` telles que `inc_sublist(Ls,Is)` est vrai.

Exemple(s) d'utilisation:

```
inc_sublist([1,2,3,1,3,6,1,4,8],X).
% 1ere version :     % 2eme version :
X = [1,2,3]; No      X = [1,2,3]; X = [1,3,6]; X = [1,4,8]; No
```

2. `alternate`
Définir un prédicat `alternate(As,Bs)`, où `As` et `Bs` sont des listes, vrai si et seulement si `Bs` est la liste `As` dont on a retiré les éléments de positions impaires. On demande que, dans le cas où `Bs` n'est pas complètement instancié mais que `As` l'est, l'instance de `Bs` pour laquelle `alternate(As,Bs)` est vrai soit générée.

3. `common_head`
Définir un prédicat `common_head(As,Bs,Cs)`, où `As`, `Bs` et `Cs` sont des listes, vrai si et seulement si `Cs` est le plus long préfixe commun à `As` et `Bs`. On demande que, dans le cas où `Cs` n'est pas complètement instancié mais que `As` et `Bs` le sont, les différentes instances de `Cs` telles que `common_head(As,Bs,Cs)` est vrai soient générées.

Exemple(s) d'utilisation:

```
common_head([1,2,3,a,b,c],[1,2,3,4,5,6],[1,2,3]).
Yes
```

4. `replace_atom`
Définir un prédicat `replace_atom(As,Bs,C,D)`, où `As` et `Bs` sont des listes et `C` et `D` des atomes, vrai si et seulement si `Bs` est `As` dont on a remplacé récursivement toute occurrence de `C` par `D`. On demande que dans le cas où une des deux listes n'est pas instanciée, mais que l'autre et les deux atomes le sont, l'instance correcte de la liste non-instanciée soit générée.

5. `replace_atom_bis`
Définir un prédicat `replace_atom_bis(As,Bs,C,D)`, où `As` et `Bs` sont des listes et `C` et `D` des atomes, vrai si et seulement si `Bs` est `As` dont on a récursivement remplacé toute occurrence de `C` par `D` et toute occurence de `D` par `C`. On demande que dans le cas où une des deux listes n'est pas instanciée, mais que l'autre et les deux atomes le sont, l'instance correcte de la liste non-instanciée soit générée.

6. `replace_atom_prof`
Définir un prédicat `replace_atom_prof(As,Bs,Cs,Ds)`, où `As` et `Bs` sont des listes et `C` et `D` des atomes, vrai si et seulement si `Bs` est `As` dont on a récursivement remplacé toute occurrence de `C` par `D` en profondeur. On demande que dans le cas où une des deux listes n'est pas instanciée, mais que l'autre et les deux atomes le sont, l'instance correcte de la liste non-instanciée soit générée.

7. `flatten2`

Ecrire un prédicat `flatten2(Ass,Bs)` vrai si et seulement si la liste `Bs` est la liste des symboles apparaissant dans la liste de liste de...`Ass` dans le même ordre. De manière intuitive : `Bs` est la liste `Ass` dont on a enlevé tous les crochets intérieurs. On demande que dans le cas où `Bs` n'est pas complètement instancié mais que `Ass` l'est, l'instance de `Bs` telle que `flatten2(Ass,Bs)` est vrai soit retournée.

Exemple(s) d'utilisation:

```
flatten2([[1,2,3,[4,5],6,[7,8,[9]]],10,11,[12]],X).
X=[1,2,3,4,5,6,7,8,9,10,11,12];
No
```

8. `threeparts`

Ecrire un prédicat `threeparts(Zs,As,Bs,Cs)` vrai si et seulement si la liste `Zs` est la concaténation des listes non-vides `As`, `Bs` et `Cs`. On demande que dans le cas où `As`, `Bs` et `Cs` ne sont pas complètement instanciés mais que `Zs` l'est, les différentes instances de `As`, `Bs`, `Cs` telles que `threeparts(Zs,As,Bs,Cs)` est vrai soient retournées. On demande aussi que dans le cas où `As`, `Bs` et `Cs` sont complètements instanciés mais que `Zs` ne l'est pas, l'instance de `Zs` rendant vrai le prédicat soit générée.

Exercices sur les arbres

Exercice introductif

9. `depth_tree`

Définir un prédicat `depth_tree(Tr,N)`, où `Tr` est un arbre binaire complet et `N` un naturel, vrai si et seulement si l'arbre `Tr` a une profondeur égale à `N`. On demande que, dans le cas où `N` n'est pas instancié mais que `Tr` l'est, l'instance de `N` telle que `depth_tree(Tr,N)` est vrai soit générée. Par convention, la racine de l'arbre est située à une profondeur de zéro. On représentera les feuilles de l'arbre par leur étiquette et les noeuds internes par une paire pointée dont le premier élément est le sous-arbre de gauche du noeud et le deuxième élément le sous-arbre de droite.

Exercices proposés

10. `parcours`

Définir un prédicat `parcours(Tr,Ls)`, où `Tr` est un arbre binaire complet dont les feuilles sont étiquetées par des naturels et `Ls` est une liste, vrai si et seulement si `Ls` est la liste des feuilles rencontrées lors d'un parcours "droite - gauche" de l'arbre, mais en remplacant les feuilles étiquetées d'un nombre impair par le nombre pair directement supérieur. On demande que, dans le cas où `Ls` n'est pas complètement instancié mais que `Tr` l'est, l'instance de `Ls` telle que `parcours(Tr,Ls)` est vrai soit générée. On représentera les arbres de la même manière que dans l'exercice précédent.

11. `depth_list`

Définir un prédicat `depth_list(Lss,N)`, où `Lss` est une liste dont les éléments sont des atomes ou des listes dont les éléments sont des atomes ou des listes dont ... et `N` est un naturel, vrai si et seulement si la liste `Lss` a une profondeur égale à `N`. On demande que, dans le cas où `N` n'est pas complètement instancié mais que `Lss` l'est, l'instance de `N` telle que `depth_list(Lss,N)` est vrai soit générée.

Exemple(s) d'utilisation:

```
depth_list([[[1],2,[3,[4]]],[]],N). => N = 4
depth_list([],N). => N = 0
```

12. `same_frame`

Définir le prédicat `same_frame(Tr1,Tr2)` vrai si et seulement si les arbres binaires complets `Tr1` et `Tr2` ont le même ensemble de feuilles (dans un premier temps, avec le même nombre d'occurrences et, dans un second temps, sans cette contrainte).

13. `simplify`

Définir le prédicat `simplify(Tr1,Tr2)` vrai si et seulement si `Tr1` et `Tr2` sont des arbres binaires complets dont les feuilles sont étiquettées par des naturels et que `Tr2` est la version simplifiée de `Tr1` obtenue en remplacant les sous-arbres ayant deux feuilles étiquettées par le même naturel par ce naturel. Dans un premier temps, on ne simplifiera qu'à un niveau de l'arbre, ensuite on simplifiera récursivement tant que les étiquettes obtenues coïncident. On demande que, dans le cas où `Tr2` n'est pas complètement instancié mais que `Tr1` l'est, l'instance de `Tr2` telle que `simplify(Tr1,Tr2)` est vrai soit générée.

Exercices arithmétiques

Exercices proposés

14. `pgcd`

Définir le prédicat `pgcd` vrai si et seulement si le troisième argument est le plus grand diviseur commun des deux premiers arguments. On demande que quand le troisième argument n'est pas instancié, mais que les deux premiers le sont, le plus grand diviseur commun soit retourné.

Exemple(s) d'utilisation:

```
gcd(23,46,N).
gcd(3289,3249,N).
gcd(3,4,N).
gcd(6,8,N).
gcd(45,125,N).
```

15. `perfect`

Définir le prédicat `perfect` vrai si et seulement si l'argument est un nombre parfait. Pour rappel, un nombre est parfait si la somme de ses diviseurs (excepté lui-même) rend le nombre.

Exemples :

```
6 = 1+2+3
28 = 1+2+4+7+14
```

Dans un premier temps le prédicat ne permettra que de tester si un nombre est parfait, dans un deuxième temps on générera les nombres parfaits (dans l'ordre croissant) lorsque l'argument n'est pas instancié.