Preuves Formelles avec Coq

laurent.thiry@uha.fr

Ce sujet s'intéresse aux collections ordonnées qui jouent un rôle très important dans la recherche d'information. Imaginez rechercher un mot dans un dictionnaire si les mots n'étaient pas rangés par ordre lexicographique.

1 Booléen et tautologies

1. Définir, en Coq, le type booléen (Bool) avec les valeurs vrai (T) et faux (F).
2. Définir les opérateurs/fonctions de négation (not), de conjonction (and) et de disjonction (or). Prouver alors que or x T=T, et or (not x) x = T. Démontrer enfin que la disjonction est associative.
3. Définir l'implication (imply) où imply x y = (not x) or y. Prouver alors que ((x and (imply x y)) imply y)=T. Nb. Une proposition toujours vraie s'appelle une "tautologie".

2 Entiers et relation d'ordre

1. Définir le type entier (Int) en Coq. On rappelle que ce type est défini par la constante 0 (Z), et la fonction successeur (S). Définir aussi les trois valeurs v1=1, v2=2 et v3=3.
2. Définir l'addition (add) entre 2 entiers et vérifier que v2+v1=v3. Prouver alors que add n (S m) = S (add d m).
3. Définir les relations d'égalité (equ) et montrer qu'elle est réflexive, i.e. equ x x=T.
4. Définir la relation d'ordre (inf) et montrer que si x<y alors z+x<z+y. On utilisera pour cela une induction sur z.
5. Définir la relation "inférieur ou égal" (infEqu) à l'aide des opérations précédentes.

3 Listes ordonnées

1. Définir le type liste d'entiers (List). On rappelle que ce type est défini par la liste vide (E), et l'opérateur d'ajout d'un élément à une liste (A). Définir aussi les trois valeurs l1=[v1,v2], l2=[v3] et l3=[v1,v2,v3].
2. Définir la concaténation (cat) de deux listes, i.e. cat [x1,...,xn] [y1,...,ym] = [x1,...,xn,y1,...,ym], et vérifier que cat l1 l2 = l3. Montrer que cette opération est bien associative. Est-elle commutative ?

3. Définir la longueur d'une liste (len) et montrer que len (cat x y) = add (len x) (len y).

4. Définir une fonction qui insère (insert) un élément à la "bonne place" dans une liste supposée triée suivant la relation inf. Montrer alors que len (insert x y) = (len y)+1.

5. Définir une fonction qui tri les éléments d'une liste (sort) en s'appuyant sur la fonction précédente. Prouver alors que len (sort x) = len x.

6. Définir une fonction (allSup) qui permet de savoir si tous les éléments d'une liste sont supérieurs ou égaux à une valeur v. Définir alors une fonction (isSorted) qui permet de savoir si une liste est triée: a) la liste vide est supposée trié, b) dans le cas d'un ajout (A x y), y doit être triée et doit avoir des valeurs supérieures/égales à x.

7. En admettant que (isSorted x) -> (isSorted (insert x y)), prouver alors que isSorted (sort x)=T.

4 Arbres binaires de recherche

1. Définir le type arbre binaire d'entiers (Tree) et pouvant être soit des feuilles (L) soit des noeuds (N) composés d'une valeur entière et de 2 sous-arbres appelés lhs (left hand side) et rhs (right hand side). Définir aussi la valeur t3 = N 2 (N 1 L L) (N 3 L L).

2. Définir la taille d'un arbre (size) comme le nombre d'éléments (entiers) contenus dans un arbre. Vérifier que size t3 = v3.

3. Définir la mise à plat d'un arbre (flatten) qui consiste à transformer un arbre en liste. Les éléments de lhs sont au début de la liste, puis il y a la valeur du noeud, et enfin les valeurs de rhs.

4. Prouver alors que size x = len (flatten x).

5. En supposant que les valeurs de lhs sont inférieures ou égales à la valeur d'un noeud, et que les valeurs de rhs sont toutes supérieures aux valeurs précédentes, définir une fonction (insertT) qui insère un élément à la "bonne place" dans un arbre.

6. En utilisant la fonction précédente, définir une fonction (toTree) qui transforme une liste en arbre. Prouver alors que size (toTree x) = len x.

7. En admettant que (isSorted (flatten x)) -> (isSorted (flatten (insertT x y))), prouver alors que isSort (flatten (toTree x))=T

5 Test d'appartenance

1. Définir une fonction (has) permettant de savoir si un élément est contenu dans une liste. Démontrer alors que has x (insert x y)=T. Démontrer aussi que has x (cat y z)=or (has x y) (has x z).

2. Définir une fonction (hasT) permettant de savoir si un élément est contenu dans un arbre. Démontrer alors que hasT x (insertT x y)=T.