Arbres binaires de recherche

TP 5

■ ■ ■

On considère un développement en Coq paramétré par un type de données `A : Type`.

Exercice 1. – Arbres binaires

Le type tree des arbres binaires dont les nœuds sont étiquetés par des éléments de type A est défini en Coq par :

```
Inductive tree : Type :=
 | Leaf : tree
 | Node : A → tree → tree → tree.
```

On utilisera les bibliothèques suivantes :

```
Require Import Arith Max Omega.
```

1. Observer la définition de la fonction `max : nat → nat → nat` qui retourne le plus grand de ses deux arguments. Énoncer et rechercher ses propriétésdéfinies par des Lemmes dans la bibliothèque Max.
2. Définir en Coq des fonctions `height` et `size` de type `tree → nat` qui calculent respectivement la hauteur et le nombre de nœuds d'un arbre. (On supposera que `height Leaf = size Leaf = 0`.)
3. Établir que pour tout arbre `t` on a `height t <= size t`. Sur quel lemme purement arithmétique repose l'argument central de cette preuve ?
4. On s'intéresse à un prédicat `elt : A → tree → Prop` exprimant que son premier argument appartient à l'arbre donné en second argument. Donner deux définitions de ce prédicat, l'une inductive, l'autre par point-fixe, et prouver leur équivalence.

On munit maintenant `A` d'une relation d'ordre total `less`, introduite à l'aide des déclarations suivantes :

```
Parameter less : A → A → Prop.
Notation "x <= y" := (less x y).
Axiom less_refl  : forall x : A,      x <= x.
Axiom less_trans : forall x y z : A,  x <= y → y <= z → x <= z.
Axiom less_asym  : forall x y : A,    x <= y → y <= x → x = y.
Axiom less_total : forall x y : A,    x <= y ∨ y <= x.
```

Exercice 2. – Arbres de recherche

On dit qu'un arbre `t : tree` est un arbre binaire de recherche (a.b.r.) si dans tout sous-arbre de `t` de la forme `Node x t1 t2`, tous les éléments de `t1` sont inférieurs ou égaux à `x` tandis que tous les éléments de `t2` sont supérieurs ou égaux à `x`.

1. Définir en Coq des relations `tree_le` et `tree_ge` de type `tree → A → Prop` qui expriment que tous les éléments de l'arbre donné en premier argument sont inférieurs ou égaux (resp. supérieurs ou égaux) à la valeur donnée en second argument.
2. Donner une définition inductive en Coq du prédicat `abr : tree → Prop` qui exprime que son argument est un a.b.r.

On suppose que la comparaison entre éléments de type `A` est effectuée au moyen d'une fonction `cmp` de type `A → A → bool` déclarée en Coq par :

```
Parameter cmp : A → A → bool.
Axiom cmp_less : forall x y : A, cmp x y = true ↔ x <= y.
```

Exercice 3. – Insertion dans un a.b.r.

1. À l'aide de la fonction de comparaison introduite ci-dessus, définir en Coq une fonction d'insertion `insert : A → tree → tree` qui insère un élément `x : A` dans un arbre `t : tree` tout en préservant la structure d'a.b.r. lorsque `t` est déjà un a.b.r.
2. Comment s'énonce la correction de la fonction d'insertion
   – vis-à-vis de la relation d'appartenance `elt` ?
   – vis-à-vis de la structure d'a.b.r. ?
   Énoncer ces deux invariants de correction en Coq.
3. Sans faire la preuve, dire sur quel(s) type(s) d'induction reposent les preuves des deux invariants de correction énoncés ci-dessus. L'hypothèse `less_total` (qui exprime que la relation d'ordre `less` est totale) intervient-elle dans ces deux preuves ? Si oui, où ? On illustrera chaque problème recensé par un contre-exemple bien choisi.
4. Prouver en Coq les deux invariants de correction.