Arbres rouge et noir penchés

DM Coq, théorie de la programmation

Envoyez par mail un fichier *individuel*, nommé `nomfamille.v`,
à `enguerrand.prebet@ens-lyon.fr, alexandre.talon@ens-lyon.fr, alexis.ghyselen@ens-lyon.fr`,
le mercredi 6 novembre à 23h59 au plus tard

Ce devoir porte sur les arbres rouge et noir, dont vous trouverez facilement la définition (soit en ligne, soit au chapitre 13 du livre classique de Cormen, Leiserson et Rivest).

On s'intéresse plus particulièrement à l'insertion dans un arbre rouge et noir *qui penche à gauche* : on ajoute aux invariants définissant les arbres rouge et noir la contrainte qu'un nœud ne peut avoir un fils droit rouge.

Il vous est demandé de travailler avec la définition Coq suivante pour représenter des arbres binaires, dans lesquels on stocke des entiers, avec une information de couleur aux nœuds :

```
Inductive color : Set := red | black.

Inductive tr : Set :=
 | L : tr
 | N : tr -> nat -> color -> tr -> tr.
```

1 Coder les invariants : fonctions

Définissez quatre fonctions `inv1, inv2, inv3, inv4`, toutes quatre de type `tr -> bool`, correspondant aux quatre invariants définissant les arbres rouge et noir qui penchent à gauche : 1) être un arbre binaire de recheche, 2) pas de parent rouge pour un nœud rouge, 3) la "profondeur noire" est la même, 4) pas de fils droit rouge.

2 Coder les invariants : prédicat

Définissez un prédicat inductif `rbl : tr -> nat -> Prop` capturant (d'un coup) les quatre propriétés, le `nat` étant la "profondeur noire". Pas le droit d'utiliser `inv1 inv2 inv3 inv4` dans la définition de `rbl`.

3 Prouver l'équivalence

Prouvez en Coq qu'un arbre est rouge et noir penchant à gauche suivant l'approche "fonctions" si et seulement s'il l'est suivant l'approche "prédicat".

4 Insérer un élément

Définissez une fonction `insert : nat -> t -> t` qui ajoute un élément dans un arbre binaire rouge et noir penchant à gauche de telle manière que le résultat soit rouge et noir penchant à gauche.

Vous pourrez vous inspirer de la fonction d'insertion dans les arbres rouge et noir standard, en l'adaptant au cas penchant à gauche.

5 Facultatif : valider `insert`

Prouvez un théorème exprimant le fait que la fonction `insert` préserve la propriété "arbre rouge et noir penchant à gauche".

Comme on dit parfois familièrement, ça n'est pas de la tarte (*it is no picnic*). Vous pouvez adoucir les choses en vous focalisant sur l'un des quatre invariants mentionnés plus haut.

Bonus (facultatif). Définir un type inductif `ltr:Set` qui code "en dur" les invariants des arbres rouge et noir penchant à gauche.

Consignes de rédaction pour la preuve Coq.

Il ne s'agit pas nécessairement de donner toutes les preuves in extenso (tout particulièrement pour ce qui est de la validation de l'insertion).

Une preuve *convainquante* devra comporter :

- la ou les bonnes inductions ;
- le traitement de quelques cas emblématiques, raisonnablement différents les un des autres, avec, en particulier, les bons appels aux hypothèses d'induction ;
- des `admit.` là où il s'agit juste de dérouler des preuves qui sont soit purement calculatoires, soit très proches de preuves que vous fournissez par ailleurs.

Vous pouvez terminer une preuve comportant des `admit.` (qui "terminent" un sous-but) par un `Admitted.` final, afin de forcer Coq à accepter une preuve incomplète.

En fonction de votre dextérité en Coq et du temps que vous pouvez y passer, vous pourrez creuser plus ou moins les méandres des preuves que vous fournissez. Bien entendu, une preuve avec trop d'`admit.` rapportera moins de points qu'une preuve convainquante.

Il vous est demandé d'utiliser les tactiques vues en cours/TP, sans faire appel aux constructions sophistiquées que propose Coq, et qui n'ont pas été abordées (*the tactic language* dans le manuel en ligne).

Important : pensez à commenter votre code Coq, tout particulièrement les endroits où vous mettez des `admit`.

Re-important : pensez à découper vos preuves en isolant des lemmes pertinents. Une preuve qui fonce tout droit pendant 100 lignes est généralement peu lisible et peu robuste (sans surprise, ce même principe vaut pour du code au sens habituel du terme).