TDS : AlgDem.

Exercice 2 : Insertion triée récursive terminale

Q 2.1 Insertion

La fonction `insererAux` est utilisée pour effectuer l'insertion au cours de l'algorithme de tri par insertion.

```
revAux :: [Int] -> [Int] -> [Int]
revAux [] l = l
revAux (x:l') l = revAux l' (x:l)

insererAux :: Int -> [Int] -> [Int] -> [Int]
insererAux k [] acc = revAux acc [k]
insererAux k (n:l) acc | k <= n     = revAux acc (k:n:l)
                       | otherwise = insererAux k l (n:acc)
```

Cette fonction utilise la fonction `revAux` étudiée en cours et dont nous avons montré que `revAux l l' = reverse l ++ l'`.

Q 2.3 Spécification et démonstration

Voici la spécification de `insererAux k l acc` :

Pré-condition `triée(revAux acc l)` et `k` est strictement plus grand que tous les éléments de `acc`.

Post-condition `triée(insererAux k l acc)`, il existe `l1` et `l2` tels que `l = l1 ++ l2` et `insererAux k l acc = revAux acc (l1 ++ [k] ++ l2)`.

Variant `length l`

Nous admettrons les propriétés suivante :

— lorsque `k` est plus grand que tous les éléments de `l1` et plus petit que tous les éléments de `l2` et que `revAux l1 l2` et triée, alors `revAux l1 (k:l2)` est triée,

— lorsque `revAux l1 l2` est triée, `l2` est triée.

Démontrez que `insererAux` vérifie sa spécification.

(Pre1) (Pre2)

(HI) ∀l'. length l' < length l ∧ triée (revAux acc []) ∧ k > acc[i] =>

(P1) ∧ triée (insererAux k l' acc)

(P2) ∧ ∃l1' l2'. l' = l1' ++ l2' ∧ insererAux k l' acc = revAux acc (l1' ++ k:l2')

P3 P4

Nous admettrons les propriétés suivante :

— lorsque `k` est plus grand que tous les éléments de `l1` et plus petit que tous les éléments de `l2` et que `revAux l1 l2` et triée, alors `revAux l1 (k:l2)` est triée,

— lorsque `revAux l1 l2` est triée, `l2` est triée.

Démontrez que `insererAux` vérifie sa spécification.

Cas 1: l = []:

insererAux k l acc = revAux acc [k] (pareil que C(21) en gros)

Cas 2: l = n:l' (C(2)

Cas 21: k <= n (C21)

insererAux k l acc = revAux acc (k:l)

=

triée (RevAux acc l). Comme K > acc[i] et K <= n, on a que

triee (Reverse acc ++ k:l) = triée (RevAux acc k:l) (P1)

On prend l1 = [] et l2 = l, ce qui donne (P2).

Cas 22 : K > n. ((22)

insererAux K (l acc = insererAux K l' (n:acc)

- Le variant décroit par (C2). length l' < length l.
- On a K > acc[i]. Comme K > n, il vient que K > (n:acc)[i]. ok.

On a triee (RevAux acc l) ⇒? triée (RevAux n:acc l')

Oui car RevAux acc n:l' = RevAux n:acc l' (par déf).

Non, il faut utiliser (P3)

On applique (H1) :

- triée (insererAux K l' (n:acc)) (direct donc ✓)
- ∃l1'.∃l2'. l' = l1'++l2' ∧ insererAux K l' (n:acc) = RevAux (n:acc) (l1'++[k]++l2')

= RevAux acc n:(l1'++[k]++l2')

On prend l1 = n:l1' et l2 = l2', ce qui conclut la preuve.