15-150 Assignment 4

Karan Sikka
ksikka@andrew.cmu.edu
G
Due 2/15/12

1: Task 3.1

Claim: For all values `n:int` and `l:int list`, if `l sorted` then `insert(n,l) sorted`.

We proceed by induction on `l`.

Case `[]`:

WTS that if `[] sorted` then `insert(n,[]) sorted`.

Given that `[]` is sorted, consider:

```
 insert(n,[])
≅case [] => n ::  [] | ...        step
≅n ::  []                         step
≅[n]                              step
```

Since `[n]` is a singleton list, it is sorted by defintion. QED.

Case `x::xs`:

Want to show that if `x::xs sorted` then `insert(n,x::xs) sorted`.

Inductive Hypothesis:

Assume that for some int list xs, if `xs sorted` then `insert(n,xs)` is also sorted.

Inductive Step:

Want to show that if the claim holds for xs, it holds for x::xs.

Consider `x::xs`. If it is not sorted, the claim is true.

Consider `insert(n,x::xs)` where `x::xs` is sorted.

```
≅insert(n,x::xs)
≅case x::xs of [] => ...  | (x ::  xs) => (case n < x of true ...           step
≅case n < x of true => n ::  (x ::  xs) | false => x ::  (insert (n , xs))   step
```

Now, `n < x` can be either true or false. Case true:

```
≅n :: (x :: xs)                              step
```

This result is sorted, because `x::xs` is sorted, and `n < x`. Now, case false:

```
≅x :: (insert (n , xs))                      step
```

`insert(n,xs)` is sorted by the IH, so therefore, the result is sorted by the definition of sortedness.

In all cases, the inductive step holds.

■

2: Task 3.2

Claim: For all valuable expressions `e:int` and `es:int` list,
if `es sorted` then `insert(e,es) sorted`.

All valuable expressions `e` and `es` will evaluate to values `x` and `xs`. Then, by the theorem proved in Task 3.1, if if `x sorted` then `insert(x,xs) sorted`. Since, `x` ≅ `e`, and `xs` ≅ `es`, the claim holds.

3: Task 3.3

Claim: For all values `l:int list`, `(isort l) sorted`.

We proceed by induction on l.

Base Case:
WTS `isort []` is sorted.

```
 isort []
≅case [] of [] => [] | ...                   step
≅[]                                          step
```

This is sorted by definition, therefore `isort []` is sorted.

Inductive Hypothesis:

Assume `isort xs` is sorted, for some `xs`.

Inductive Step:

WTS `isort x::xs` is sorted.

Consider:

`isort x::xs`	*step*
$\cong$`case x::xs of [] => [] | (x :: xs) => insert (x , isort xs)`	*step*
$\cong$`insert (x , isort xs)`	*step*

isort xs is sorted, by the IH. By the lemma proved in Task 3.1, this is also sorted.

4: Task 4.4

Claim: For all values `t1 t2:tree`, `depth (combine(t1,t2))` $\leq$ `1+max(depth t1, depth t2)`.

Base Case:

Want to show that `depth (combine(Empty,t2))` $\leq$ `1+max(depth Empty, depth t2)`.

Consider:

`depth (combine(Empty,t2))`	*step*
$\cong$`depth (case Empty of Empty => t2 | Node(l,x,r) => ...`	*step*
$\cong$`depth t2`	*step*

`depth t2` is less than `1+depth(t2)`, so the claim holds for the base case.

Inductive Hypothesis: ? **Inductive Step:** ?

5: Task 5.2

1. $W_{takeanddrop}(0) = k_0$
   $W_{takeanddrop}(d) = k_0 + W_{takeanddrop}(d-1)$
   This recurrance is $O(d)$ because it expands to the sum of d constants.

2. $S_{takeanddrop}(0) = k_0$
   $S_{takeanddrop}(d) = k_0 + S_{takeanddrop}(d-1)$
   This recurrance is $O(d)$ because there is no room for parallelism. Each step depends on something in a sequential manner.

3. $W_{halves}(d) \leq 2W_{takeanddrop}(d)$
   This recurrance is $O(d)$.

4. $S_{halves}(d) = max(S_{takeanddrop}(d), S_{takeanddrop}(d)) = S_{takeanddrop}(d)$
   This recurrance is $O(d)$.

5. $W_{rebalance}(0) = k_0$
   $W_{rebalance}(n) = k_0 + W_{halves}(n) + 2W_{rebalance}(\frac{n}{2})$
   $W_{rebalance}(n) = k_0 + log(n) + 2W_{rebalance}(\frac{n}{2})$
   By the tree method, the right-most term can be reformulated to be:

Karan Sikka
ksikka@andrew.cmu.edu

$W_{rebalance}(n) = k_0 + log(n) + \sum_{i=1}^{logn} k_0$
$W_{rebalance}(n) = k_0 + log(n) + k_0 log(n)$
Therefore, the bound for this is $O(log(n))$.

6. $S_{rebalance}(0) = k_0$
$S_{rebalance}(n) = k_0 + S_{halves}(n) + max(W_{rebalance}(\frac{n}{2}), W_{rebalance}(\frac{n}{2}))$
$S_{rebalance}(n) = log(n) + W_{rebalance}(\frac{n}{2}) S_{rebalance}(n) = log(n) + \sum_{i=1}^{log(n)} n$
$S_{rebalance}(n) = log(n) + log(n)$ This is bounded by $log(n)$.