15-150 Assignment 04

Jack Kasbeer
jkasbeer@andrew.cmu.edu
Section K
September 29, 2015

2: Parentheses Matching

1. Prove: for all values `A : paren list, n : int,` such that `n >= 0` and `B : paren list`, if `A` is balanced, then

   `is_balanced(A @ B, n) = is_balanced(B, n)`

   Proof. By strong induction on *length* of A.
   Let $P(n)$ be the proposition that `is_balanced(A @ B, n) = is_balanced(B, n)` for all `n >= 0` and balanced integer lists `A`, where `n = length A`.
   Base Case: `length(A) = 0`. It's trivially true that `is_balanced(A @ B, 0) = is_balanced(B, 0)` since `A@B = B`.
   Inductive Hypothesis: Assume $P(k)$ is true for all integer lists with $length \neq 1$.
   Considering $P(k+1)$...
   `is_balanced(A @ B, k) = is_balanced((L::A' @ R::A'') @ B, k)`, by def. of balanced paren list.
   `=> is_balanced(L::(A'@R::A''@B), k)`, by lemma 4
   `=> is_balanced(A'@(R::A''@B), k+1)`, by proven spec of `is_balanced`
   `=> is_balanced(R::A''@B, k+1)`, by IH
   `=> is_balanced(A''@B, k)`, by `is_balanced`
   `=> is_balanced(B, k)`, by IH
   It follows that for all values `A : paren list, n : int,` such that `n >= 0` and `B : paren list`, if `A` is balanced, then

   `is_balanced(A @ B, n) = is_balanced(B, n)`

   □

2. Show: If `L : paren list` is a balanced list, then `is_balanced(L, 0) = true`

   Proof. Let `L` be a balanced list. Then, for some balanced lists `M1` and `M2`, `L = (Left::M1) @ (Right::M2)`, by def. of a balanced paren list.
   `is_balanced(L, 0) = is_balanced((Left::M1) @ (Right::M2), 0)`, by Task 2.1
   `=> is_balanced(Left::M1, 0) = is_balanced(M1, 0+1)`, but by def. M1 is balanced so this is `true`
   and `is_balanced(Right::M2, 1) = if n > 0 then is_balanced(M2, 0) else false`
   `=> is_balanced(Right::M2, 1) = is_balanced(M2, 0)` and M2 is also defined to be balanced.
   Hence, if `L : paren list` is a balanced list, then `is_balanced(L, 0) = true` □

3. `pmatch` is implemented in `hw04.sml`

Jack Kasbeer
jkasbeer@andrew.cmu.edu

3: Full Trees

1. `shape : tree * int -> tree`

2. Work to evaluate `shape(T, n)`
   $W_{shape}(Empty, n) = c_0$
   $W_{shape}(T^m, n) = c_1 + W_{shape(T-left,n+1)} + W_{shape(T-right,n+1)}$
   $W_{shape}(T^m, n) = c_1 + 2 * W_{shape(T^{m-1},n+1)}$
   $W_{shape}(T^m, n) = c_2 + 4 * W_{shape(T^{m-2},n+2)}$
   $W_{shape}(T^m, n) = O(2^m)$

3. Span to evaluate `shape(T,n)`
   $S_{shape}(Empty, n) = c_0$
   $S_{shape}(T^m, n) = c_1 + max(S_{shape(T^{m-1},n+1)}, S_{shape(T^{m-1},n+1)})$
   $S_{shape}(T^m, n) = c_1 + S_{shape(T^{m-1},n+1)}$
   $S_{shape}(T^m, n) = O(m)$

4. `census` is implemented in `hw04.sml`

4: Quicksorting a List

1. `part` is implemented in `hw04.sml`

2. Try to prove: For all integer lists, `L`, `quicksort L` evaluates to a `<`-sorted permutation of `L`. The problem in the previous implementation was that it was using the first element as a basis for the sort. The proof breaks down when trying to prove that `x` is in the right position in the final `<`-sorted list.

3. Fixed definition of `quicksort` is in `hw04.sml`.

5: Tree Traversal

1. `traversal` is implemented in `hw04.sml`.

Jack Kasbeer
jkasbeer@andrew.cmu.edu

6: Tree Size

1. Prove: for all values `t : tree`

   `size(t) = length(treeToList t)`

   Proof. By structural induction on `T`.
   Let $P(T)$ be the proposition that

   `size(T) = length(treeToList(T))`

   Base Case: $P(Empty) \Rightarrow$ `size(Empty) = 0 = length(treeToList(T)`, trivially.
   Inductive Hypothesis: Let $T = (L, x, R)$ and assume $P(L)$ and $P(R)$ are true for some tree with a non-negative depth.
   WWTS $P(T)$ is true for all integers $x \geq 0$.
   `Node(L, x, R)) = size(L) + 1 size(R)`, by def. of `size`
   `= 1 + length(treeToList(L)) + length(treeToList(R))`, by IH
   `= length(treeToList(L) + x::treeToList(R))`, by Lemma 2
   `= length(treeToList(L) @ x::treeToList(R))`, by Lemma 2
   `= length(treeToList(Node(L,x,R)))`, by def. of `treeToList`
   Hence, by structural induction on `T`, for all values `t : tree`, `size(t) = length(treeToList t)`
   □

7: Heaps and Heaps of Heaps

1. `insert` is implemented in `hw04.sml`.
2. `join` is implemented in `hw04.sml`.
3. `heapify` is implemented in `hw04.sml`.