MAT/CSC – 220: Discrete Structures
EFY

Thomas R. Cameron

March 23, 2018

Problem 1. Use structural induction to verify the correctness of the following function:

$$val\ rec\ height\colon\ tree \to\ int\ =$$
$$fn\ Leaf\ n\ \ =>\ 0$$
$$|\ Internal(n,x,y)\ \ =>\ 1 + Int.max(height(x), height(y))$$

Note that max is a member function of the integer signature of SML.

Problem 2. Use structural induction to prove the following result.

Theorem. *For any full binary tree T, the number of nodes in T is odd.*

Discussion Points. Below is a list of interesting topics that relate to trees and structural induction.

- Structural Induction makes it easy to verify the correctness of operations on recursively defined data structures. Therefore, making recursive functions elegant solutions to problems.
- Recursion is done "properly" in a functional programming paradigm. That is, by treating computation as the evaluation of mathematical functions and avoiding changing state and mutable data we are able to run recursive functions without hitting a recursion depth limit and avoiding stack overflow.
- In order to fully optimize recursion, any two term recurrence relations should be modified to a one term relation. This is what is meant by tail recursion.

  For instance, consider the example below.

$$val\ rec\ fib1\ :\ int \to int\ =$$
$$fn\ 0\ \ =>\ 1$$
$$|\ 1\ \ =>\ 1$$
$$|\ n : int\ \ =>\ fib1(n-1) + fib1(n-2)$$

Fib1 is a recursive function that evaluates Fibonacci numbers in perhaps the most natural way. However, this natural definition has an exponential cost complexity. Whereas, the Fib2 implementation of the Fibonacci numbers has a linear cost complexity.

```
local
    val rec helper : int → int * int =
    fn 0 => (1, 0)
    | 1 => (1, 1)
    | n : int =>
        let
            val (a : int, b : int) = helper(n − 1)
        in
            (a + b, a)
        end
in
    val fib2 : int → int =
    fn n : int =>
        let
            val(a : int, _) = helper(n)
        in
            a
        end
end
```