Quiz Section 8: Trees

Task 1 – One, Two, Tree...

The problem makes use of the following inductive type, representing a *left-leaning* binary tree

$$\begin{aligned}\textbf{type}\ \text{Tree} := \quad & \text{empty} \\ \mid \quad & \text{node}(\text{val} : \mathbb{Z},\ \text{left} : \text{Tree},\ \text{right} : \text{Tree}) \quad \text{with } \text{height}(\text{left}) \geqslant \text{height}(\text{right})\end{aligned}$$

The "with" condition is an *invariant* of the node. Every node that is created must have this property, and we are allowed to use the fact that it holds in our reasoning.

The height of a tree is defined recursively by

$$\text{height} : \text{Tree} \to \mathbb{Z}$$

$$\begin{aligned}\text{height}(\text{empty}) &:= -1 \\ \text{height}(\text{node}(x, S, R)) &:= 1 + \text{height}(S)\end{aligned}$$

In a general binary tree, the height of a non-empty tree is the length of the *longest* path to a leaf. With a left-leaning tree, we know the longest path is the one that always travels toward the left child.

We can define the size of a tree, the number of values stored in it, as follows:

$$\text{size} : \text{Tree} \to \mathbb{N}$$

$$\begin{aligned}\text{size}(\text{empty}) &:= 0 \\ \text{size}(\text{node}(x, S, R)) &:= 1 + \text{size}(S) + \text{size}(R)\end{aligned}$$

Prove by structural induction that, for any left-leaning tree T, we have

$$\text{size}(T) \leqslant 2^{\text{height}(T)+1} - 1$$

Task 2 – How Do I Love Tree, Let Me Count the Ways

The following is the definition of a binary search tree:

$$\begin{aligned} \textbf{type}\ \text{BST} := \quad & \text{empty} \\ \mid \quad & \text{node}(x : \mathbb{Z},\ S : \text{BST},\ R : \text{BST}) \end{aligned}$$

Suppose that we wanted to have a way to refer to a specific node in a BST. One way to do so would be to give directions from the root to that node. If we define these types:

$$\begin{aligned} \textbf{type}\ \text{Dir} \quad & := \text{LEFT} \mid \text{RIGHT} \\ \textbf{type}\ \text{Path} \quad & := \text{List}\langle \text{Dir} \rangle \end{aligned}$$

then a Path tells you how to get to a particular node where each step along the path (item in the list) would be a direction pointing you to keep going down the LEFT or RIGHT branch of the tree.

For example, LEFT :: RIGHT :: nil says to select the "LEFT"child of the parent and then the "RIGHT" child of that node, giving us a grand-child of the root node.

(a) Define a function "find(p : Path, T : BST)" that returns the node (a BST) at the path from the root of T or undefined if there is no such node.

(b) Define a function "remove(p : Path, T : BST)" that returns T except with the node at the given path replaced by empty.

Task 3 – Let's Blow This Point

Suppose we had the following interface for a Point class that represents a point in 2D space:

```
/** Represents a point with coordinates in (x,y) space. */
interface Point {
    /** @returns the x coordinate of the point */
    getX: () => number;

    /** @returns the y coordinate of the point */
    getY: () => number;

    /**
     * Returns the distance of this point to the origin.
     * @returns Math.sqrt(obj.x*obj.x + obj.y*obj.y)
     */
    distToOrigin: () => number;
}
```

The following is an implementation of the Point interface:

```
class SimplePoint implements Point {
    // RI: <TODO>
    // AF: <TODO>
    readonly x: number;
    readonly y: number;
    readonly r: number;

    // Creates a point with the given coordinates
    constructor(x: number, y: number) {
     this.x = x;
     this.y = y;
     this.r = Math.sqrt(x*x + y*y);
    }

    getX = (): number => this.x;
    getY = (): number => this.y;
    distToOrigin = (): number => this.r;
}
```

(a) Define the representation invariant (RI) and abstraction function (AF) for the SimplePoint class.

(b) Use the RI or AF to prove that the distToOrigin method of the SimplePoint class is correct.

(c) The following problem will make use of this math definition that rotates a point around the origin (x, y) by an angle θ:

$$\text{rotate} : (\text{Point},\ \mathbb{R}) \to \text{Point}$$

$$\text{rotate}((x, y), \theta) \quad = \quad (x \cdot \cos(\theta) - y \cdot \sin(\theta),\ x \cdot \sin(\theta) + y \cdot \cos(\theta))$$

Suppose we have the following implementation of the rotate method:

/** @returns rotate(obj, θ) */

```
rotate = (theta: number): Point => {
    const newX = this.x * Math.cos(theta) - this.y * Math.sin(theta);
    const newY = this.x * Math.sin(theta) + this.y * Math.cos(theta);
    return new SimplePoint(newX, newY);
}
```

Prove that the rotate method is correct using the RI or AF.

Task 4 – Going Back and Length

The following problem will make use of the following functions that operate on lists:

$$\text{len} : \text{List} \to \mathbb{N}$$

$$\begin{aligned} \text{len}(\text{nil}) &:= 0 \\ \text{len}(x :: L) &:= 1 + \text{len}(L) \end{aligned}$$

$$\text{rev} : \text{List} \to \text{List}$$

$$\begin{aligned} \text{rev}(\text{nil}) &:= \text{nil} \\ \text{rev}(x :: L) &:= \text{rev}(L) {+}{+} [x] \end{aligned}$$

Suppose we also have the fact Lemma 1: $\text{len}(\text{rev}(L) + +[x]) = \text{len}(\text{rev}(L)) + \text{len}(x :: nil)$ for any list L and element x.

Prove by Structural Induction that $\text{len}(\text{rev}(L)) = \text{len}(L)$ for any list L. You may find that you need to use Lemma 1 in your proof.