S4: Reasoning About Correctness - Addendum

CSci 2041:

Advanced Programming Principles

University of Minnesota,
Prof. Van Wyk,
Spring 2022

sum over lists and trees — See notes in `Notes` directory.

Recall `sum`:

```
let rec sum = function
  | [] -> 0
  | x::xs -> x + sum xs
```

How about `sumt` for summing values in trees?

```
type 'a tree = Empty
             | Node of 'a tree * 'a * 'a tree

let rec sumt (t: int tree) : int =   match t with
  | Empty -> 0
  | Node (left, v, right) -> sumt left + v + sumt right

let rec tolist (t: 'a tree) : 'a list = match t with
  | Empty -> []
  | Node (l, v, r) -> tolist l @ [v] @ tolist r
```

Can we prove that ∀ `t : int tree . sumt t = sum (tolist t)` ?

Exercise S4: #1: Deriving principles of induction

What is the principle of induction for this type?

```
type 'a nelist = One of 'a
               | Cons of 'a * 'a nelist
```

Exercise S4: #2: Deriving principles of induction

What is the principle of induction for this type?

```
type 'a tree = Leaf of 'a
             | Fork of 'a tree * 'a * 'a tree
```


Exercise S4: #3: Deriving principles of induction

What is the principle of induction for this type?

```
type 'a tree = Empty
             | Leaf of 'a
             | Fork of 'a tree * 'a * 'a tree
```


Exercise S4: #4: Show `minTree t = minNeList (flatten t)`

```
type 'a nelist = One of 'a
               | Cons of 'a * 'a nelist
type 'a tree = Leaf of 'a
             | Fork of 'a tree * 'a * 'a tree
let rec minNEList (ns: 'a nelist) : 'a = match ns with
  | One x -> x
  | Cons (x, xs) -> min x (minNEList xs)
let rec minTree (t: int tree) : int = match t with
  | Leaf v -> v
  | Fork (l, v, r) -> min v (min (minTree l) (minTree r))
let rec app (l1: 'a nelist) (l2: 'a nelist) : 'a nelist =
  match l1 with
  | One v -> Cons (v, l2)
  | Cons (v,vs) -> Cons (v, app vs l2)
let rec flatten (t: 'a tree) : 'a nelist = match t with
  | Leaf v -> One v
  | Fork (l, v, r) -> app (flatten l) (app (One v) (flatten r))
```