W2.4 Equational Reasoning

Subtask 2.4.1

Prove the following property:

```
forall (xs :: [a]) (ys :: [a]) .
length (xs ++ ys) = length xs + length ys
```

- First case: `xs = []`:

```
length (xs ++ ys)
  = length ([] ++ ys)       -- assumption
  = length ys               -- definition of (++)
  = 0 + length ys           -- arithmetic
  = length [] + length ys   -- definition of length
  = length xs + length ys   -- assumption
```

- Second case: Let `x :: a`, let `xs, ys :: [a]`,
  and assume `length (xs ++ ys) = length xs + length ys`:

```
length ((x : xs) ++ ys)
  = length (x : (xs ++ ys))       -- definition of (++)
  = 1 + length (xs ++ ys)         -- definition of length
  = 1 + (length xs + length ys)   -- assumption
  = (1 + length xs) + length ys   -- arithmetic
  = length (x : xs) + length ys   -- definition of length
```

- Third case: `xs = undefined`:

```
length (xs ++ ys)
  = length (undefined ++ ys)       -- assumption
  = length undefined               -- definition of (++)
  = undefined                      -- definition of length
  = undefined + length ys          -- definition of (+)
  = length undefined + length ys   -- definition of length
  = length xs + length ys          -- assumption
```

Subtask 2.4.2

Prove the following property:

```
forall (t :: Tree a) .
length (flatten t) = size t
```

- First case: `t = Leaf a`:

```
length (flatten t)
  = length (flattern (Lead a))  -- assumption
```

```
  = length (a : [])              -- definition of flatten
  = 1                            -- definition of length (twice)
  = size (Leaf a)                -- definition of size
  = size t                       -- assumption
```

- Second case: Let `l, r :: Tree a`, let `t = Node l r`, and assume `length (flatten l) = size l` and `length (flatten r) = size r`:

```
length (flatten t)
  = length (flatten (Node l r))               -- assumption
  = length (flatten l ++ flatten r)           -- definition of flatten
  = length (flatten l) ++ length (flatten r)  -- Subtask 2.4.1
  = sile l + size r                           -- assumption
  = size (Node l r)                           -- definition of size
  = size t                                    -- assumption
```

- Third case: `t = undefined`:

```
length (flatten t)
  = length (flatten undefined)  -- asumption
  = length undefined            -- definition of flatten
  = undefined                   -- definition of length
  = size undefined              -- definition of size
  = size t                      -- assumption
```