2008 Formosan Summer School on Logic, Language, and Computation

Program Construction and Reasoning Exercises for Day 1

Shin-Cheng Mu

July 4th, 2008

1 In-Class Exercises

1.1 The Expand/Reduce Transformation

1. (a) What does this function do?

$$\begin{array}{lcl} \mathit{descend}\, 0 & = & [\,] \\ \mathit{descend}\,(n+1) & = & (n+1) : \mathit{descend}\, n \end{array}$$

Ans:
$\mathit{descend}\, n = [n, n-1, (n-2), \cdots, 1]$

(b) Consider the definition $f = \mathit{sum} \cdot \mathit{descend}$, synthesise a recursive definition of f.

Ans:
Case 0:

$$\begin{array}{ll} & f\, 0 \\ = & \quad \{\text{ def. of } f \ \} \\ & \mathit{sum}\,(\mathit{descend}\, 0) \\ = & \quad \{\text{ def. of } \mathit{descend} \ \} \\ & \mathit{sum}\,[\,] \\ = & \quad \{\text{ def. of } \mathit{sum} \ \} \\ & 0 \end{array}$$

Case $n+1$:

$$\begin{array}{ll} & f\,(n+1) \\ = & \quad \{\text{ def. of } f \ \} \\ & \mathit{sum}\,(\mathit{descend}\,(n+1)) \\ = & \quad \{\text{ def. of } \mathit{descend} \ \} \\ & \mathit{sum}\,((n+1) : \mathit{descend}\, n) \end{array}$$

$$
\begin{array}{ll}
= & \{\text{ def. of } \mathit{sum} \} \\
 & (n+1) + \mathit{sum}\,(\mathit{descend}\; n) \\
= & \{\text{ def. of } f \} \\
 & (n+1) + f\; n
\end{array}
$$

Thus we have constructed:

$$
\begin{array}{lll}
f\,0 & = & 0 \\
f\,(n+1) & = & n + 1 + f\; n.
\end{array}
$$

2. Recall the datatype definition for internally labelled binary trees:

$$
\mathbf{data}\; \mathit{ITree}\;\alpha \quad = \quad \mathit{Null} \mid \mathit{Node}\;\alpha\;(\mathit{ITree}\;\alpha)\;(\mathit{ITree}\;\alpha).
$$

(a) Consider the function *mapiTree* defined below:

$$
\begin{array}{l}
\mathit{mapiTree}\,f\;\mathit{Null} = \mathit{Null}, \\
\mathit{mapiTree}\,f\;(\mathit{Node}\;x\;t\;u) = \\
\qquad \mathit{Node}\;(f\;x)\;(\mathit{mapiTree}\,f\;t)\;(\mathit{mapiTree}\,f\;u).
\end{array}
$$

What does this function do?

Ans:

The function call $\mathit{mapiTree}\,f\;t$ applies f to every element in t.

(b) Define a function *sumiTree* computing the sum of all node values in an *iTree*.

Ans:

$$
\begin{array}{lll}
\mathit{sumiTree}\;\mathit{Null} & = & 0 \\
\mathit{sumiTree}\;(\mathit{Node}\;a\;t\;u) & = & a + \mathit{sumiTree}\;t + \mathit{sumiTree}\;u
\end{array}
$$

(c) The function $\mathit{one}\;x = 1$ returns 1, whatever the input is. The function *sizeiTree* is specified by:

$$
\mathit{sizeiTree} \quad = \quad \mathit{sumiTree} \cdot \mathit{mapiTree}\;\mathit{one}.
$$

What does this function do? Derive a definition of *sizeiTree* which does not construct an intermediate tree.

Ans:

The function call $\mathit{sizeiTree}\;t$ computes the size of t.

Case Null:

$$
\begin{array}{ll}
 & \mathit{sizeiTree}\;\mathit{Null} \\
= & \{\text{ def. of } \mathit{sizeiTree} \} \\
 & \mathit{sumiTree}\;(\mathit{mapiTree}\;\mathit{one}\;\mathit{Null}) \\
= & \{\text{ def. of } \mathit{mapiTree} \} \\
 & \mathit{sumiTree}\;\mathit{Null} \\
= & \{\text{ def. of } \mathit{sumiTree} \}
\end{array}
$$

Case $Node\ a\ t\ u$:

$$\begin{array}{ll} & sizeiTree\,(Node\ a\ t\ u) \\ = & \quad \{\text{ def. of } sizeiTree\ \} \\ & sumiTree\,(mapiTree\ one\,(Node\ a\ t\ u)) \\ = & \quad \{\text{ def. of } mapiTree\ \} \\ & sumiTree\,(Node\,(one\ a)\,(mapiTree\ one\ u)\,(mapiTree\ one\ t)) \\ = & \quad \{\text{ def. of } sumiTree \text{ and } one\ \} \\ & 1 + sumiTree \cdot mapiTree\ one\ u + sumiTree \cdot mapiTree\ one\ t \\ = & \quad \{\text{ def. of } sizeiTree\ \} \\ & 1 + sizeiTree\ u + sizeiTree\ t \end{array}$$

Therefore,

$$\begin{array}{lcl} sizeiTree\ Null & = & 0 \\ sizeiTree\,(Node\ a\ t\ u) & = & 1 + sizeiTree\ u + sizeiTree\ t. \end{array}$$

3. Recall the datatype definition for externally labelled binary trees:

$$\mathbf{data}\ ETree\ \alpha \quad = \quad Tip\ \alpha \mid Bin\,(ETree\ \alpha)\,(ETree\ \alpha).$$

(a) What does this function do?

$$\begin{array}{lcl} mineTree\,(Tip\ x) & = & x \\ mineTree\,(Bin\ t\ u) & = & mineTree\ t \downarrow mineTree\ u \end{array}$$

Ans:
The function call $mineTree\ t$ returns the minimum element in t.

(b) What does this function do?

$$\begin{array}{lcl} repeTree\ x\,(Tip\ y) & = & Tip\ x \\ repeTree\ x\,(Bin\ t\ u) & = & Bin\,(repeTree\ x\ t)\,(repeTree\ x\ u) \end{array}$$

Ans:
The function call $repeTree\ x\ t$ replaces every element in t by x.

(c) What does this function do?

$$\begin{array}{lcl} repbymin\ t & = & \mathbf{let}\ m = mineTree\ t \\ & & \mathbf{in}\ repeTree\ m\ t \end{array}$$

How many times does this program traverse the input tree?
Ans:
The function call $repbymin\ t$ replaces every element in t by its minimum element. It traverses t twice, once for computing the minimum element, once for the replacement.

(d) Consider this definition:

$$repmin\ x\ t \quad = \quad (repeTree\ x\ t, mineTree\ t)$$

Construct a recursive definition of *repmin* that traverses the tree only once.
Ans:
Case $Tip\ y$:

$$
\begin{array}{ll}
 & repmin\ x\ (Tip\ y) \\
= & \quad \{ \text{ def. of } repmin \ \} \\
 & (repeTree\ x\ (Tip\ y), mineTree\ (Tip\ y)) \\
= & \quad \{ \text{ def. of } repeTree \text{ and } mineTree \ \} \\
 & (Tip\ x, y)
\end{array}
$$

Case $Bin\ t\ u$:

$$
\begin{array}{ll}
 & repmin\ x\ (Bin\ t\ u) \\
= & \quad \{ \text{ def. of } repmin \ \} \\
 & (repeTree\ x\ (Bin\ t\ u), mineTree\ (Bin\ t\ u)) \\
= & \quad \{ \text{ def. of } repeTree \text{ and } mineTree \ \} \\
 & (Bin\ (repeTree\ x\ t)\ (repeTree\ x\ u), mineTree\ t \downarrow mineTree\ u) \\
= & \quad \{ \text{ giving names to some sub-expressions } \} \\
 & \mathbf{let}\ (t', y) = (repeTree\ x\ t, mineTree\ t) \\
 & \qquad (u', z) = (repeTree\ x\ u, mineTree\ u) \\
 & \mathbf{in}\ (Bin\ t'\ u', y \downarrow z) \\
= & \quad \{ \text{ def. of } repmin \ \} \\
 & \mathbf{let}\ (t', y) = repmin\ x\ t \\
 & \qquad (u', z) = repmin\ x\ u \\
 & \mathbf{in}\ (Bin\ t'\ u', y \downarrow z)
\end{array}
$$

Thus we have derived the definition:

$$
\begin{array}{lcl}
repmin\ x\ (Tip\ y) & = & (Tip\ x, y) \\
repmin\ x\ (Bin\ t\ u) & = & \mathbf{let}\ (t', y) = repmin\ x\ t \\
 & & \qquad (u', z) = repmin\ x\ u \\
 & & \mathbf{in}\ (Bin\ t'\ u', y \downarrow z)
\end{array}
$$

(e) Redefine *repbymin* as:

$$
\begin{array}{lcl}
repbymin'\ t & = & \mathbf{let}\ (t', m) = repmin\ m\ t \\
 & & \mathbf{in}\ t'.
\end{array}
$$

How many times does this definition of *repbymin* traverse the tree?
Ans:
The function call $repbymin\ t$ traverses t only once. The replacement is done during searching for the minimum element.

1.2 Proof by Induction

1. Prove $(xs +\!\!+\ ys) +\!\!+\ zs = xs +\!\!+\ (ys +\!\!+\ zs)$. Hint: induction on xs.
Ans:

Case $[\,]$:

$$
\begin{array}{cl}
 & ([\,] +\!\!+\; ys) +\!\!+\; zs \\
= & \quad \{ \text{ def. of } (+\!\!+) \text{ } \} \\
 & ys +\!\!+\; zs \\
= & \quad \{ \text{ def. of } (+\!\!+) \text{ } \} \\
 & [\,] +\!\!+\; (ys +\!\!+\; zs)
\end{array}
$$

Case $x{:}xs$: Suppose the equality hold for xs,

$$
\begin{array}{cl}
 & ((x{:}xs) +\!\!+\; ys) +\!\!+\; zs \\
= & \quad \{ \text{ def. of } (+\!\!+) \text{ } \} \\
 & (x{:}(xs +\!\!+\; ys)) +\!\!+\; zs \\
= & \quad \{ \text{ def. of } (+\!\!+) \text{ } \} \\
 & x{:}((xs +\!\!+\; ys) +\!\!+\; zs) \\
= & \quad \{ \text{ induction hypothesis } \} \\
 & x{:}(xs +\!\!+\; (ys +\!\!+\; zs)) \\
= & \quad \{ \text{ definition of } (+\!\!+) \text{ } \} \\
 & (x{:}xs) +\!\!+\; (ys +\!\!+\; zs)
\end{array}
$$

2. The function *concat* concatenates a list of lists:

$$
\begin{array}{lcl}
concat\,[\,] & = & [\,], \\
concat\,(xs : xss) & = & xs +\!\!+\; concat\; xss.
\end{array}
$$

E.g. $concat\,[[1,2],[3,4],[5]] = [1,2,3,4,5]$. Prove that:

$$sum \cdot concat = sum \cdot map\; sum.$$

Hint: you may need one of the properties proved in the lecture.

Ans:

Case $[\,]$:

$$
\begin{array}{cl}
 & (sum \cdot concat)\,[\,] \\
= & \quad \{ \text{ def. of } (\cdot) \text{ } \} \\
 & sum(concat\,[\,]) \\
= & \quad \{ \text{ def. of } concat \text{ } \} \\
 & sum\,[\,] \\
= & \quad \{ \text{ def. of } map \text{ } \} \\
 & sum(map\; sum[\,]) \\
= & \quad \{ \text{ def. of } (\cdot) \text{ } \} \\
 & (sum \cdot map\; sum)\,[\,]
\end{array}
$$

Case $xs{:}xss$: suppose the equality holds for xss,

$$
\begin{array}{ll}
 & (\mathit{sum} \cdot \mathit{concat})\,(xs{:}xss) \\
= & \quad \{ \text{ def. of } (\cdot) \text{ } \} \\
 & \mathit{sum}\,(\mathit{concat}\,(xs{:}xss)) \\
= & \quad \{ \text{ def. of } \mathit{concat} \text{ } \} \\
 & \mathit{sum}\,(xs \mathbin{+\!\!+} \mathit{concat}\ xss) \\
= & \quad \{ \text{ since } \mathit{sum}\,(xs \mathbin{+\!\!+} ys) = \mathit{sum}\ xs + \mathit{sum}\ ys \text{ } \} \\
 & \mathit{sum}\ xs + \mathit{sum}\,(\mathit{concat}\ xs) \\
= & \quad \{ \text{ induction hypothesis } \} \\
 & \mathit{sum}\ xs + \mathit{sum}\,(\mathit{map}\ \mathit{sum}\ xss) \\
= & \quad \{ \text{ def. of } \mathit{sum} \text{ } \} \\
 & \mathit{sum}\,((\mathit{sum}\ xs){:}\mathit{map}\ \mathit{sum}\ xss) \\
= & \quad \{ \text{ def. of } \mathit{map} \text{ } \} \\
 & \mathit{sum}\,(\mathit{map}\ \mathit{sum}\,(xs{:}xss)) \\
= & \quad \{ \text{ def. of } (\cdot) \text{ } \} \\
 & (\mathit{sum} \cdot \mathit{map}\ \mathit{sum})\,(xs{:}xss)
\end{array}
$$

3. Prove that $\mathit{map}\, f \cdot \mathit{map}\, g = \mathit{map}\,(f \cdot g)$.
   Ans:
   The case for $[\,]$ is trivial. For the inductive case, suppose the equality holds for xs. We reason:

$$
\begin{array}{ll}
 & (\mathit{map}\, f \cdot \mathit{map}\, g)\,(x{:}xs) \\
= & \quad \{ \text{ def. of } (\cdot) \text{ } \} \\
 & \mathit{map}\, f\,(\mathit{map}\, g\,(x{:}xs)) \\
= & \quad \{ \text{ def. of } \mathit{map} \text{ } \} \\
 & \mathit{map}\, f\,((g\ x){:}\mathit{map}\ g\ xs) \\
= & \quad \{ \text{ def. of } \mathit{map} \text{ } \} \\
 & (f\ (g\ x)){:}\mathit{map}\, f\,(\mathit{map}\ g\ xs) \\
= & \quad \{ \text{ def. of } (\cdot) \text{ } \} \\
 & ((f \cdot g)\ x){:}\mathit{map}\, f\,(\mathit{map}\ g\ xs) \\
= & \quad \{ \text{ induction hypothesis } \} \\
 & ((f \cdot g)\ x){:}\mathit{map}\,(f \cdot g)\ xs \\
= & \quad \{ \text{ def. of } \mathit{map} \text{ } \} \\
 & \mathit{map}\,(f \cdot g)\,(x{:}xs)
\end{array}
$$

4. The function $swapTree$ is defined by:

$$\begin{array}{lcl} swapiTree\ Null & = & Null, \\ swapiTree\ (Node\ a\ t\ u) & = & Node\ a\ (swapiTree\ u)\ (swapiTree\ t). \end{array}$$

Prove that $swapiTree\ (swapiTree\ t) = t$ for all t.
Ans:
The $Null$ case is easy: $swapTree\ (swapTree\ Null) = swapTree\ Null = Null$. To prove the case for $Node\ a\ t\ u$, suppose the equality holds for t and u. For any a, by definition of $swapTree$, we have:

$$\begin{array}{ll} & swapTree\ (swapTree\ (Node\ a\ t\ u)) \\ = & swapTree\ (Node\ a\ (swapTree\ u)\ (swapTree\ t)) \\ = & Node\ a\ (swapTree\ (swapTree\ t))\ (swapTree\ (swapTree\ u)) \\ = & \quad \{\text{ induction hypothesis }\} \\ & Node\ a\ t\ u \end{array}$$

1.3 Accumulating Parameters

1. Recall the standard definition of factorial:

$$\begin{array}{lcl} fact\ 0 & = & 1, \\ fact\ (n+1) & = & (n+1) \times fact\ n. \end{array}$$

This program also implicitly uses space linear to n in the call stack.

(a) Introduce $factit\ n\ m = \ldots$ where m is an accumulating parameter.
Ans:
We accumulate the chain of products $(n+1) \times n \times \cdots$ in the parameter m:

$$factit\ n\ m = m * fact\ n$$

(b) Express $fact$ in terms of $factit$.
Ans:
$fact\ n = factit\ n\ 1$

(c) Construct a space efficient implementation of $factit$.
Ans:
Case 0:

$$\begin{array}{ll} & factit\ 0\ m \\ = & \quad \{\text{ def. of } factit\ \} \\ & m * fact\ 0 \\ = & m \end{array}$$

Case $n+1$:

$$
\begin{array}{ll}
& \mathit{factit}\,(n+1)\,m \\
= & \quad \{ \text{ def. of } \mathit{factit} \ \} \\
& m * \mathit{fact}\,(n+1) \\
= & \quad \{ \text{ def. of } \mathit{fact} \ \} \\
& m * (n+1) * \mathit{fact}\,n \\
= & \quad \{ \text{ def. of } \mathit{factit} \ \} \\
& \mathit{factit}\,n\,(m * (n+1))
\end{array}
$$

Thus,

$$
\begin{array}{lcl}
\mathit{factit}\,0\,m & = & m \\
\mathit{factit}\,(n+1)\,m & = & \mathit{factit}\,n\,(m * (n+1))
\end{array}
$$

2. Recall the standard definition of Fibonacci:

$$
\begin{array}{l}
\mathit{fib}\,0 = 0 \\
\mathit{fib}\,1 = 1 \\
\mathit{fib}\,(n+2) = \mathit{fib}\,(n+1) + \mathit{fib}\,n
\end{array}
$$

Let us try to derive a linear-time, tail-recursive algorithm computing *fib*.

(a) Given the definition $\mathit{fibit}\,n\,x\,y = \mathit{fib}\,n \times x + \mathit{fib}\,(n+1) \times y$. Express *fib* using *fibit*.
Ans:
$\mathit{fib}\,n = \mathit{fibit}\,n\,1\,0$

(b) Derive a linear-time version of *fibit*.
Ans:
Case 0:

$$
\begin{array}{ll}
& \mathit{fibit}\,0\,x\,y \\
= & \quad \{ \text{ def. of } \mathit{fibit} \ \} \\
& \mathit{fib}\,0 \times x + \mathit{fib}\,1 \times y \\
= & \quad \{ \text{ def. of } \mathit{fib} \ \} \\
& 0 \times x + 1 \times y \\
= & \quad \{ \text{ arithmetics } \} \\
& y
\end{array}
$$

Case $n+1$:

$$
\begin{array}{ll}
& \mathit{fibit}\,(n+1)\,x\,y \\
= & \quad \{ \text{ def. of } \mathit{fibit} \ \} \\
& \mathit{fib}\,(n+1) \times x + \mathit{fib}\,(n+2) \times y \\
= & \quad \{ \text{ def. of } \mathit{fib} \ \} \\
& \mathit{fib}\,(n+1) \times x + (\mathit{fib}\,(n+1) + \mathit{fib}\,n) \times y
\end{array}
$$

$$
\begin{aligned}
&= \quad \{\ \text{arithmetics}\ \} \\
&\quad\ \mathit{fib}\, n \times y + \mathit{fib}\,(n+1) \times (x+y) \\
&= \quad \{\ \text{def. of } \mathit{fibit}\ \} \\
&\quad\ \mathit{fibit}\, n\, y\, (x+y)
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
\mathit{fibit}\, 0\, x\, y &= y \\
\mathit{fibit}\, (n+1)\, x\, y &= \mathit{fibit}\, n\, y\, (x+y)
\end{aligned}
$$

2 Take-Home Exercise (Due Date: July 10th)

You need to complete only one of the two exercises. Exercise 1 is worth 35 points while exercise 2 is worth 40 points.

1. Given an *iTree*, the following function *flatten* returns a list of all labels in the tree, in left-to-right order:

$$
\begin{aligned}
\mathit{flatten}\ \mathit{Null} &= [\,], \\
\mathit{flatten}\, (\mathit{Node}\, x\, t\, u) &= \mathit{flatten}\, t \mathbin{+\!\!+} [x] \mathbin{+\!\!+} \mathit{flatten}\, u.
\end{aligned}
$$

   Unfortunately, *flatten* is slow. Let us try to improve it. Introduce $\mathit{flatcat}\, t\, xs = \mathit{flatten}\, t \mathbin{+\!\!+} xs$.

   (a) Express *flatten* in terms of *flatcat*.
   Ans:
   $\mathit{flatten}\, t = \mathit{flatcat}\, t\, [\,]$

   (b) Construct an efficient implementation of *flatten*. You will need some properties of $(\mathbin{+\!\!+})$ proved in one of the exercises.
   Ans:

$$
\begin{aligned}
&\quad\ \mathit{flatcat}\ \mathit{Null}\ xs \\
&= \quad \{\ \text{def. of } \mathit{flatcat}\ \} \\
&\quad\ \mathit{flatten}\ \mathit{Null} \mathbin{+\!\!+} xs \\
&= \quad \{\ \text{def. of } \mathit{flatten} \text{ and } \mathbin{+\!\!+}\ \} \\
&\quad\ xs
\end{aligned}
$$

$$
\begin{aligned}
&\quad\ \mathit{flatcat}\, (\mathit{Node}\, a\, t\, u)\, xs \\
&= \quad \{\ \text{def. of } \mathit{flatcat}\ \} \\
&\quad\ \mathit{flatten}\, (\mathit{Node}\, a\, t\, u) \mathbin{+\!\!+} xs \\
&= \quad \{\ \text{def. of } \mathit{flatten}\ \} \\
&\quad\ (\mathit{flatten}\, t \mathbin{+\!\!+} a \mathbin{+\!\!+} \mathit{flatten}\, u) \mathbin{+\!\!+} xs \\
&= \quad \{\ \text{assoc. of } \mathbin{+\!\!+}\ \}
\end{aligned}
$$

$$\begin{aligned}
& \mathit{flatten}\ t \mathbin{+\!\!+} ([a] \mathbin{+\!\!+} (\mathit{flatten}\ u \mathbin{+\!\!+} xs)) \\
= & \quad \{ \text{ def. of } \mathit{flatcat} \ \} \\
& \mathit{flatten}\ t \mathbin{+\!\!+} ([a] \mathbin{+\!\!+} \mathit{flatcat}\ u\ xs) \\
= & \quad \{ \text{ def. of } \mathbin{+\!\!+} \ \} \\
& \mathit{flatten}\ t \mathbin{+\!\!+} (a{:}\mathit{flatcat}\ u\ xs) \\
= & \quad \{ \text{ def. of } \mathit{flatcat} \ \} \\
& \mathit{flatcat}\ t\ (a{:}\mathit{flatcat}\ u\ xs)
\end{aligned}$$

I.e.

$$\begin{aligned}
\mathit{flatcat}\ \mathit{Null}\ xs &= xs \\
\mathit{flatcat}\ (\mathit{Node}\ a\ t\ u)\ xs &= \mathit{flatcat}\ t\ (a{:}\mathit{flatcat}\ u\ xs)
\end{aligned}$$

Hint:

(a) To see the specification running, load `mu-code.hs` into `Hugs` or `GHCi`, and try `flatten testTree1 1`. Run your derived program to check whether it produces the same output as the specification.

(b) The derivation works in a way similar to how *revcat* was constructed in the class. You may need to perform some steps more than once.

2. This problem considers labelling an internally-labelled binary tree:

$$\mathbf{data}\ \mathit{iTree}\ \alpha \quad = \quad \mathit{Null} \mid \mathit{Node}\ \alpha\ (\mathit{iTree}\ \alpha)\ (\mathit{iTree}\ \alpha).$$

Given such a tree, for example (the labels in the tree does not matter, so let us assume they are just ()):

$$\begin{aligned}
t \quad = \quad & \mathit{Node}\ ()\ (\mathit{Node}\ ()\ (\mathit{Node}\ ()\ \mathit{Null}\ \mathit{Null}) \\
& \qquad\qquad\qquad (\mathit{Node}\ ()\ \mathit{Null}\ \mathit{Null})) \\
& \qquad\quad (\mathit{Node}\ ()\ \mathit{Null} \\
& \qquad\qquad\qquad (\mathit{Node}\ ()\ (\mathit{Node}\ ()\ \mathit{Null}\ \mathit{Null}) \\
& \qquad\qquad\qquad\qquad \mathit{Null})),
\end{aligned}$$

the task is to number the nodes, in depth-first order:

$$\begin{aligned}
t \quad = \quad & \mathit{Node}\ 1\ (\mathit{Node}\ 2\ (\mathit{Node}\ 3\ \mathit{Null}\ \mathit{Null}) \\
& \qquad\qquad\qquad (\mathit{Node}\ 4\ \mathit{Null}\ \mathit{Null})) \\
& \qquad\quad (\mathit{Node}\ 5\ \mathit{Null} \\
& \qquad\qquad\qquad (\mathit{Node}\ 6\ (\mathit{Node}\ 7\ \mathit{Null}\ \mathit{Null}) \\
& \qquad\qquad\qquad\qquad \mathit{Null})).
\end{aligned}$$

The following function *label* specifies how to label a tree, starting from a given initial number n:

$$\begin{aligned}
\mathit{label}\ \mathit{Null}\ n &= \mathit{Null}, \\
\mathit{label}\ (\mathit{Node}\ _\ t\ u)\ n &= \mathit{Node}\ n\ (\mathit{label}\ t\ (1+n)) \\
& \qquad\qquad (\mathit{label}\ u\ (1+n+\mathit{sizeiTree}\ t)),
\end{aligned}$$

where *size* is defined by:

$$\begin{array}{lcl} \mathit{sizeiTree}\ \mathit{Null} & = & 0, \\ \mathit{sizeiTree}\ (\mathit{Node}\ x\ t\ u) & = & 1 + \mathit{sizeiTree}\ t + \mathit{sizeiTree}\ u. \end{array}$$

Due to repeated call to *size*, the above definition of *label* is rather inefficient. Define:

$$\mathit{labeltl}\ t\ n \quad = \quad (\mathit{label}\ t\ n, n + \mathit{size}\ t),$$

derive a recursive definition for *labeltl* that runs in time linear to the size of the tree. Hint:

(a) To see the specification running, load `mu-code.hs` into `Hugs` or `GHCi`, and try `label testTree2 1`. Run your derived program to check whether it produces the same output as the specification.

(b) *labeltl* may need to call itself more than once in the recursive definition. You may need to introduce **let** in the definition, perhaps more than once.

Ans:
Case analysis on t.
Case *Null*:

$$\begin{array}{ll} & \mathit{labeltl}\ \mathit{Null}\ n \\ = & \quad \{ \text{ def. of } \mathit{labeltl} \ \} \\ & (\mathit{label}\ \mathit{Null}\ n, n + \mathit{sizeiTree}\ \mathit{Null}) \\ = & \quad \{ \text{ def. of } \mathit{label} \text{ and } \mathit{sizeiTree} \ \} \\ & (\mathit{Null}, n) \end{array}$$

Case $\mathit{Node}\ x\ t\ u$:

$$\begin{array}{ll} & \mathit{labeltl}\ (\mathit{Node}\ x\ t\ u)\ n \\ = & \quad \{ \text{ def. of } \mathit{labeltl} \ \} \\ & (\mathit{label}\ (\mathit{Node}\ x\ t\ u)\ n, n + \mathit{sizeiTree}\ (\mathit{Node}\ x\ t\ u)) \\ = & \quad \{ \text{ def. of } \mathit{label} \text{ and } \mathit{sizeiTree} \ \} \\ & (\mathit{Node}\ n\ (\mathit{label}\ t\ (n+1))\ (\mathit{label}\ u\ (1 + n + \mathit{sizeiTree}\ t)), \\ & \ \ n + 1 + \mathit{sizeiTree}\ t + \mathit{sizeiTree}\ u) \\ = & \quad \{ \text{ introducing local identifiers } \} \\ & \mathbf{let}\ (t', n_1) = (\mathit{label}\ t\ (n+1), n + 1 + \mathit{sizeiTree}\ t) \\ & \mathbf{in}\ (\mathit{Node}\ n\ t'\ (\mathit{label}\ u\ n_1), n_1 + \mathit{sizeiTree}\ u) \\ = & \quad \{ \text{ introducing local identifiers } \} \\ & \mathbf{let}\ (t', n_1) = (\mathit{label}\ t\ (n+1), n + 1 + \mathit{sizeiTree}\ t) \\ & \quad\ \ (u', n_2) = (\mathit{label}\ u\ n_1, n_1 + \mathit{sizeiTree}\ u) \\ & \mathbf{in}\ (\mathit{Node}\ n\ t'\ u', n_2) \end{array}$$

$$
\begin{array}{ll}
= & \{\ \text{def. of } \mathit{labeltl}\ \} \\
& \textbf{let}\ (t', n_1) = \mathit{labeltl}\ t\ (n+1) \\
& \quad\ (u', n_2) = \mathit{labeltl}\ u\ n_1 \\
& \textbf{in}\ (\mathit{Node}\ n\ t'\ u', n_2)
\end{array}
$$

Therefore we have derived:

$$
\begin{array}{lcl}
\mathit{labeltl}\ \mathit{Null}\ n & = & (\mathit{Null}, n) \\
\mathit{labeltl}\ (\mathit{Node}\ x\ t\ u)\ n & = & \textbf{let}\ (t', n_1) = \mathit{labeltl}\ t\ (n+1) \\
& & \quad\ (u', n_2) = \mathit{labeltl}\ u\ n_1 \\
& & \textbf{in}\ (\mathit{Node}\ n\ t'\ u', n_2)
\end{array}
$$