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?

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

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

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 $\mathit{mapiTree}$ 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?

   (b) Define a function $\mathit{sumiTree}$ computing the sum of all node values in an iTree.

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

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

   What does this function do? Derive a definition of $\mathit{sizeiTree}$ which does not construct an intermediate tree.

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?

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

   (b) What does this function do?

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

   (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?

   (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.

   (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?

1.2 Proof by Induction

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

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.

3. Prove that $map\, f \cdot map\, g = map\, (f \cdot g)$.

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.

1.3 Accumulating Parameters

1. Recall the standard definition of factorial:

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

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

   (a) Introduce $\mathit{factit}\; n\; m = \ldots$ where m is an accumulating parameter.

   (b) Express *fact* in terms of *factit*.

   (c) Construct a space efficient implementation of *factit*.

2. Recall the standard definition of Fibonacci:

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

   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*.

   (b) Derive a linear-time version of *fibit*.

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*.

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

   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:

$$\textbf{data}\, iTree\ \alpha \quad = \quad Null \mid Node\ \alpha\ (iTree\ \alpha)\ (iTree\ \alpha).$$

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

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

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

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

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

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

where *size* is defined by:

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

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

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

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.