Answers to the final exam on Algebraic Specification

Christian Rinderknecht

17 June 2005

1 Binary tree specification

Let us recall an algebraic specification of binary trees over nodes. Let us call it BIN-TREE(node). Here is the signature:

- **Defined types**
  - The type of the binary trees is noted t.
  - The type of the nodes is noted node.
- **Constructors**
  - EMPTY : t
    The term EMPTY represents the empty tree, i.e. the tree that contains no node.
  - MAKE : node $\times$ t $\times$ t $\rightarrow$ t
    The term MAKE(r, t_1, t_2) denotes the tree whose root is r, left subtree is t_1 and right subtree is t_2. Graphically:
- **Functions**
  1. MEM : t $\times$ node $\rightarrow$ bool
     The term MEM(t, e) is TRUE if node e occurs in tree t, otherwise it is equal to FALSE. (We assume we have equality on nodes.)
  2. MIN-DEPTH : t $\times$ node $\rightarrow$ int
     The term MIN-DEPTH(t, e) denotes the minimum depth at which node e occurs in tree t. (The root is at depth 0.) If e is not in t, the value is unspecified. In case the node e appears several

times in t, the value is the smallest depth of occurrence. (We assume we can compare nodes for equality and that we have the function $\textsc{Min} : \mathsf{int} \times \mathsf{int} \rightarrow \mathsf{int}$ which returns the smallest integer argument.) For example, if t is

$$\text{then } \begin{cases} \textsc{Min-Depth}(t, 7) & \text{is unspecified (i.e. undefined)} \\ \textsc{Min-Depth}(t, 1) = 0 \\ \textsc{Min-Depth}(t, 3) = 2 \\ \textsc{Min-Depth}(t, 5) = \textsc{Min}(1, 2) = 1 \end{cases}$$

3. $\textsc{Inv} : \mathsf{t} \rightarrow \mathsf{t}$
   The term $\textsc{Inv}(t)$ denotes the tree t in a mirror, i.e. the left subtrees of t are the right subtrees of $\textsc{Inv}(t)$ and the right subtrees become the left subtrees. Therefore $\textsc{Inv}(\textsc{Inv}(t)) = t$. For example, the following trees are mirrors of each other:

4. $\textsc{Sum} : \textsc{Bin-Tree}(\mathsf{int}).\mathsf{t} \rightarrow \mathsf{int}$
   The term $\textsc{sum}(t)$ denotes the sum of all (integer) nodes in tree t. If t is empty, the sum is unspecified. For example, if t is

   then $\textsc{Sum}(t) = 1 + 5 + 3 + 4 + 6 + 5 = 24$.

Question 1. Complete this signature with equations defining the functions.

Answer 1.

1. $\textsc{Mem} : \mathsf{t} \times \mathsf{node} \rightarrow \mathsf{bool}$
   This function takes an argument of type $\mathtt{t}$, i.e. a binary tree. Since

binary trees can be constructed in two different ways, the first step is to write the left-hand sides with these constructors:

$$\begin{aligned}\textsc{Mem}(\textsc{Empty}, e) &= \textbf{?}\\ \textsc{Mem}(\textsc{Make}(r, t_1, t_2), e) &= \textbf{?}\end{aligned}$$

The first equation is easy to complete: by definition, there is no node in the empty tree. Therefore

$$\begin{aligned}\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False}\\ \textsc{Mem}(\textsc{Make}(r, t_1, t_2), e) &= \textbf{?}\end{aligned}$$

In the last equation, we notice that there are two nodes, r and e, so we can compare them for equality and split the equation correspondingly:

$$\begin{aligned}\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False}\\ \textsc{Mem}(\textsc{Make}(e, t_1, t_2), e) &= \textbf{?}\\ \textsc{Mem}(\textsc{Make}(r, t_1, t_2), e) &= \textbf{?} && \text{if } r \neq e\end{aligned}$$

We can now complete the second equation because we know, by definition of the case, that e is in t (it is its root):

$$\begin{aligned}\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False}\\ \textsc{Mem}(\textsc{Make}(e, t_1, t_2), e) &= \textsc{True}\\ \textsc{Mem}(\textsc{Make}(r, t_1, t_2), e) &= \textbf{?} && \text{if } r \neq e\end{aligned}$$

In the case of the last equation, we know that e is not the root of t. Therefore, we must check whether e is not at some other place in t by means of recursive calls in the left and right subtrees of t:

$$\begin{aligned}\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False}\\ \textsc{Mem}(\textsc{Make}(e, t_1, t_2), e) &= \textsc{True}\\ \textsc{Mem}(\textsc{Make}(r, t_1, t_2), e) &= \textsc{Mem}(t_1, e) \parallel \textsc{Mem}(t_2, e) && \text{if } r \neq e\end{aligned}$$

Actually, we can simplify the equations by merging back the third and second equations:

$$\begin{aligned}\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False}\\ \textsc{Mem}(\textsc{Make}(r, t_1, t_2), e) &= (r = e) \parallel \textsc{Mem}(t_1, e) \parallel \textsc{Mem}(t_2, e)\end{aligned}$$

2. $\textsc{Min-Depth} : \mathsf{t} \times \mathsf{node} \to \mathsf{int}$
   The first step is to note that $\textsc{Min-Depth}$ takes an argument of type t, i.e. a binary tree, hence we can consider the two constructors for tree as this argument:

$$\begin{aligned}\textsc{Min-Depth}(\textsc{Empty}, e) &= \textbf{?}\\ \textsc{Min-Depth}(\textsc{Make}(r, t_1, t_2), e) &= \textbf{?}\end{aligned}$$

What is the minimum depth of any node e in the empty tree? Since, by definition, there is no node in the empty tree, there is no depth for any node: it is unspecified. Therefore, the first equation is meaningless, and we can remove it. Now remains

$$\textsc{Min-Depth}(\textsc{Make}(r, t_1, t_2), e) = \textbf{?}$$

There are two nodes, r and e, in the left-hand side of this equation, so we can compare them for equality and split the equation accordingly:

$$\begin{aligned} &\textsc{Min-Depth}(\textsc{Make}(e, t_1, t_2), e) = \textbf{?} \\ &\textsc{Min-Depth}(\textsc{Make}(r, t_1, t_2), e) = \textbf{?} \qquad \text{if } r \neq e \end{aligned}$$

In the first equation, the node we are looking for is the root of t. Since the text explaining the signature tells that depth of the root is always 0, we just found the right-hand side:

$$\begin{aligned} &\textsc{Min-Depth}(\textsc{Make}(e, t_1, t_2), e) = 0 \\ &\textsc{Min-Depth}(\textsc{Make}(r, t_1, t_2), e) = \textbf{?} \qquad \text{if } r \neq e \end{aligned}$$

In the remaining equation, the node we search, e, is not the root of t. Thus we must search the left and right subtrees recursively and take the minimum of the two depth **plus one**:

$$\begin{aligned} &\textsc{Min-Depth}(\textsc{Make}(e, t_1, t_2), e) = 0 \\ &\textsc{Min-Depth}(\textsc{Make}(r, t_1, t_2), e) = 1 + \textsc{Min}(\quad \textsc{Min-Depth}(t_1, e), \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textsc{Min-Depth}(t_2, e)) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if } r \neq e \end{aligned}$$

But something is wrong with the last equation... What if e is not in t_1 or not in t_2? Then the values $\textsc{Min-Depth}(t_1, e)$ or $\textsc{Min-Depth}(t_2, e)$ would be undefined, but what is the minimum of two undefined values? It is preferable then to constrain e to be in t in order to get a specified value $\textsc{Min-Depth}(t, e)$:

$$\begin{aligned} &\textsc{Min-Depth}(\textsc{Make}(e, t_1, t_2), e) = 0 \\ &\textsc{Min-Depth}(\textsc{Make}(r, t_1, t_2), e) = 1 + \textsc{Min}(\quad \textsc{Min-Depth}(t_1, e), \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textsc{Min-Depth}(t_2, e)) \\ &\qquad\qquad\qquad\qquad \text{if } r \neq e \text{ and } \textsc{Mem}(\textsc{Make}(r, t_1, t_2), e) \end{aligned}$$

3. $\textsc{Inv} : \mathsf{t} \rightarrow \mathsf{t}$
   The sole argument of $\textsc{Inv}$ is of type `t`, i.e. it is a binary tree, so we start by enumerating the tree constructors:

$$\begin{aligned} \textsc{Inv}(\textsc{Empty}) &= \textbf{?} \\ \textsc{Inv}(\textsc{Make}(r, t_1, t_2)) &= \textbf{?} \end{aligned}$$

The first equation specifies the image of the empty tree in a mirror (inverse). Since, by definition, there is no node in the empty tree, its image can only be itself:

$$\begin{aligned}\textsc{Inv}(\textsc{Empty}) &= \textsc{Empty}\\ \textsc{Inv}(\textsc{Make}(r, t_1, t_2)) &= \textbf{?}\end{aligned}$$

Now what about the non-empty tree case? First, we should understand that the invariant part is the root: the image of the root is at the same place:

$$\begin{aligned}\textsc{Inv}(\textsc{Empty}) &= \textsc{Empty}\\ \textsc{Inv}(\textsc{Make}(r, t_1, t_2)) &= \textsc{Make}(r, \textbf{?}, \textbf{?})\end{aligned}$$

The subtrees have to permute, i.e. we swap them in order to get the reverse order **and** we reverse them also with a recursive call:

$$\begin{aligned}\textsc{Inv}(\textsc{Empty}) &= \textsc{Empty}\\ \textsc{Inv}(\textsc{Make}(r, t_1, t_2)) &= \textsc{Make}(r, \textsc{Inv}(t_2), \textsc{Inv}(t_1))\end{aligned}$$

4. $\textsc{Sum} : \textsc{Bin-Tree}(\mathsf{int}).\mathsf{t} \rightarrow \mathsf{int}$
We note that $\textsc{Sum}$ as a unique argument of type $\textsc{Bin-Tree}(\mathsf{int}).\mathsf{t}$, i.e. it is a binary tree over positive integers. So we start by enumerating the tree constructors for this argument:

$$\begin{aligned}\textsc{Sum}(\textsc{Empty}) &= \textbf{?}\\ \textsc{Sum}(\textsc{Make}(n, t_1, t_2)) &= \textbf{?}\end{aligned}$$

The first equation refers to the sum of the nodes of th empty tree. Since, by definition, there is no node in the empty tree, the sum is therefore undefined. Hence we should remove this equation, but... what if the empty tree is part of a non-empty tree (like a tree just made of a root)? For instance, $\textsc{Sum}(\textsc{Make}(5, \textsc{Empty}, \textsc{Empty})) = 5$. In this case we expect the sum of the empty tree to be 0, because it should no modify the whole sum, so 5 actually comes from $5 + 0 + 0$. Thus we have to stick with

$$\begin{aligned}\textsc{Sum}(\textsc{Empty}) &= 0\\ \textsc{Sum}(\textsc{Make}(n, t_1, t_2)) &= \textbf{?}\end{aligned}$$

We already know that we have to sum n to the rest of the nodes:

$$\begin{aligned}\textsc{Sum}(\textsc{Empty}) &= 0\\ \textsc{Sum}(\textsc{Make}(n, t_1, t_2)) &= n + \textbf{?}\end{aligned}$$

We have to add now the sum of the left subtree and the sum of the right subtree nodes, as recursive calls:

$$\begin{aligned}\textsc{Sum}(\textsc{Empty}) &= 0\\ \textsc{Sum}(\textsc{Make}(n, t_1, t_2)) &= n + \textsc{Sum}(t_1) + \textsc{Sum}(t_2)\end{aligned}$$

Question 2. Why a breadth-first search is better than a depth-first search in defining Min-Depth?

Answer 2. If the tree is not well-balanced, i.e., if not all the leaves lay at the same level, then a breadth-first search would stop as soon as a leaf is found, whereas a deep-first search has to visit all the nodes.