CMU 15-671 Models of Software Systems Fall 1997

Structural Induction

Garlan & Wing Handout 4 10 September 1997

1 Introduction

We often find ourselves in the situation of wanting to prove something of the form:

$\forall x : S \bullet P(x)$

That is, we want to prove that all elements of some set S have the property P. If all of the elements of S are defined by some regular structural rules, induction is frequently the proof technique of choice.

Examples of sets that can be defined by structural rules include:

- **natural numbers**: starting with the element 0, each element is the successor of some other element in the set.
- **sequences**: starting with the empty sequence, $<>$, each element is built up by appending an element to an existing list.
- **parse trees**: starting with the "terminal" nodes of a grammar, each parse true is a structure defined by one of a "non-terminal" productions of the grammar.

In the first of these examples, the proof technique is typically referred to as "natural induction". In the others, it is typically referred to as "structural induction".

In these notes we will illustrate the use of structural induction over binary trees. Other forms of structural induction work similarly. Here is how we will proceed. First, we provide a "structural" definition of trees: this determines the rules that allow us to define all trees of interest. Next, we define what we mean by "size" of a tree. Then we will propose a small theorem about calculating the size; this is what we will then prove.

2 Definition of Binary Trees

There are many ways to model binary trees using sets and functions. For this example we will use a definition that relies on some Z notation for describing recursive data structures. The details of the syntax are not terribly important at this point, but you should understand the basic idea about building up trees from other trees.

Here is the definition:

$TREE ::= leaf \mid node\langle\!\langle TREE \times TREE \rangle\!\rangle$

It says that a tree is either a leaf or is made up of two subtrees glued together with node. Usually we would associate some data with the leaf and/or nodes, but for present purposes this simple definition will suffice.

Here are some examples of the kind of structures that we can build up using this definition:

$leaf$
$node(leaf, leaf)$
$node(node(leaf, leaf), leaf)$
$node(leaf, node(node(leaf, leaf), leaf))$

3 Definition of Size

There are also many ways that we might define the size of a tree. Some definitions would count just the leaves, others just the nodes. Here we will count both.

Informally, we could say that the size of a single leaf is 1, while the size of a tree built out of two subtrees, say t_1 and t_2, is the sum of the sizes of the two subtrees plus 1 (for the joining node).

The basic idea of this definition is that we define the size of a tree inductively over the structure, saying how the size of a given tree is calculated from sizes its parts. Putting this more formally in the notation of *Software Engineering Mathematics*, we would define a function by first declaring its type (in this case $size : TREE \longrightarrow \mathbb{N}$), and then saying how it is defined in each of the two cases.

$$
\begin{array}{|l}
size : TREE \longrightarrow \mathbb{N} \\
\hline
\forall\, t_1, t_2 : TREE \bullet \\
\qquad size(leaf) = 1 \wedge \\
\qquad size(node(t_1, t_2)) = 1 + size(t_1) + size(t_2)
\end{array}
$$

[Note: I am doing things a little differently than what I tried to do in class. Here I am simply asserting that this is the definition of size; in class I tried to prove something about *this definition.]*

In a similar way, we might make other definitions about trees. Here are two useful ones:

$$
\begin{array}{|l}
leaves : TREE \longrightarrow \mathbb{N} \\
nodes : TREE \longrightarrow \mathbb{N} \\
\hline
\forall\, t_1, t_2 : TREE \bullet \\
\qquad leaves(leaf) = 1 \wedge \\
\qquad leaves(node(t_1, t_2)) = leaves(t_1) + leaves(t_2) \wedge \\
\qquad nodes(leaf) = 0 \wedge \\
\qquad nodes(node(t_1, t_2)) = 1 + nodes(t_1) + nodes(t_2)
\end{array}
$$

4 Proving a Theorem by Structural Induction

Based on what we've said about our definitions, there is a pretty obvious connection between the three definitions give above. That is, we would expect the size of a tree to be the sum of the leaves and the nodes. We can make this precise as follows:

Theorem 1 $\forall\, t : TREE \bullet size(t) = leaves(t) + nodes(t).$

We will prove this by structural induction. As with natural deduction there are two cases: a base case for leaves, and an induction case for composite trees. Here is how it goes:

Proof: Let t: TREE.

Base Case: Assume t is a leaf.

$$
\begin{array}{ll}
size(t) & \\
= & \text{[assumption]} \\
size(leaf) & \\
= & \text{[definition of } size\text{]} \\
1 & \\
= & \text{[arithmetic]} \\
1 + 0 &
\end{array}
$$

$=$ [definition *leaves*]

$leaves(t) + 0$

$=$ [definition *nodes*]

$leaves(t) + nodes(t)$

Induction Case: Assume that $t = node(t_1, t_2))$ and that the theorem hold for all trees $t' : TREE$ such that $size(t') < size(t)$.

$size(t)$

$=$ [assumption]

$size(node(t_1, t_2)))$

$=$ [definition of *size*]

$1 + size(t_1) + size(t_2)$

$=$ [induction hypothesis (applied to t_1 and t_1)]

$1 + (leaves(t_1) + nodes(t_1)) + (leaves(t_2) + nodes(t_2))$

$=$ [commutative and associative properties of $+$]

$(leaves(t_1) + leaves(t_2)) + (1 + nodes(t_1) + nodes(t_2))$

$=$ [definition of *leaves*]

$leaves(node(t_1, t_2)) + (1 + nodes(t_1) + nodes(t_2))$

$=$ [definition of *nodes*]

$leaves(node(t_1, t_2)) + nodes(node(t_1, t_2))$

$=$ [definition of t]

$leaves(t) + nodes(t)$

5 Exercises for the Reader

1. Show that $\forall\, t : TREE \bullet leaves(t) = nodes(t) + 1$.
2. Define a *mirror* function that flips a tree over. Then show that $\forall\, t : TREE \bullet size(mirror(t)) = size(t)$.