Internal Note #241
Computational Logic, Inc.
1717 West Sixth Street, Suite 290
Austin, Texas 78703-4776
August 7, 1991 ** Draft **

Using NQTHM to Verify Insert, Search, and Traversal Functions for Search Trees

John R. Cowles*

Abstract

This note reports on a mechanical proof that a tree insertion algorithm, similar to one shown to work correctly on ordered trees by Boyer and Moore, works correctly on search trees. A tree search algorithm and a tree traversal algorithm are also shown to be correct.

Trees

In [1], Boyer and Moore defined the abstract data type "ordered tree", gave a tree insertion algorithm defined as a recursive function in the Boyer-Moore logic, and proved mechanically that this insertion function works correctly. Their definitions and theorems can be found in Appendix A.

Recursively define a (binary) *tree* (of numbers) as either (i) the empty tree or (ii) an ordered triple containing a nonnegative integer, called the *root* of the tree, and two trees, called the *left* and *right* subtrees of the tree.

```
(ADD-SHELL TREE-CONS
           EMPTY-TREE
           TREEP
           ((ROOT (ONE-OF NUMBERP) ZERO)
            (LEFT (ONE-OF TREEP) EMPTY-TREE)
            (RIGHT (ONE-OF TREEP) EMPTY-TREE) ))
```

*On leave from the Department of Computer Science, University of Wyoming, Laramie, WY 82071.
This work was supported in part at Computational Logic, Inc., by the Defense Advanced Research Projects Agency, ARPA Order 7406. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., the Defense Advanced Research Projects Agency or the U.S. Government.

An *ordered tree* is a tree that is either empty or one that satisfies the following:

1. Both the left and right subtrees are ordered trees.
2. The root (if it exists) of the left subtree is less than the root of the tree which is less than the root (if it exists) of the right subtree.

```
(DEFN ORDERED-TREE-P ( X )
  (IF ( NOT (TREEP X))
      F
      (IF (EQUAL X (EMPTY-TREE))
          T
          (AND (ORDERED-TREE-P (LEFT X))
               (ORDERED-TREE-P (RIGHT X))
               (OR (EQUAL (LEFT X)(EMPTY-TREE))
                   (LESSP (ROOT (LEFT X))(ROOT X)))
               (OR (EQUAL (RIGHT X)(EMPTY-TREE))
                   (LESSP (ROOT X)(ROOT (RIGHT X))))))))
```

A *search tree* is a tree that is either empty or one that satisfies the following:

1. Both the left and right subtrees are search trees.
2. Every number (if any) in the left subtree is less than the root of the tree which is less than every number (if any) in the right subtree.

The next two `DEFN`'s capture the notion that all the numbers in a given tree, `TREE`, are either (i) less than or (ii) greater than a given number, `NBR`. With these two definitions, it is possible to define search trees in the logic.

```
(DEFN LESSP-TREE-NBR ( TREE NBR )
  (IF (EMPTY-TREEP TREE)
      T
      (AND (LESSP (ROOT TREE) NBR)
           (LESSP-TREE-NBR (LEFT TREE) NBR)
           (LESSP-TREE-NBR (RIGHT TREE) NBR))))

(DEFN LESSP-NBR-TREE ( NBR TREE )
  (IF (EMPTY-TREEP TREE)
      T
      (AND (LESSP NBR (ROOT TREE))
           (LESSP-NBR-TREE NBR (LEFT TREE))
           (LESSP-NBR-TREE NBR (RIGHT TREE)))))

(DEFN SEARCH-TREE-P ( X )
  (IF (NOT (TREEP X))
      F
      (IF (EQUAL X (EMPTY-TREE))
          T
          (AND (SEARCH-TREE-P (LEFT X))
               (SEARCH-TREE-P (RIGHT X))
               (LESSP-TREE-NBR (LEFT X)(ROOT X))
               (LESSP-NBR-TREE (ROOT X)(RIGHT X))))))
```

Ordered trees are very similar to search trees:

- Every search tree is also an ordered tree.

```
(PROVE-LEMMA SEARCH-TREES-ARE-ORDERED-TREES
  NIL
  (IMPLIES (SEARCH-TREE-P X)
           (ORDERED-TREE-P X)))
```

- Not every ordered tree is a search tree.

```
(PROVE-LEMMA EXAMPLE-IS-ORDERED-TREE
  NIL
  (ORDERED-TREE-P
   (TREE-CONS 2
              (TREE-CONS 1
                         (EMPTY-TREE)
                         (TREE-CONS 3
                                    (EMPTY-TREE)
                                    (EMPTY-TREE)))
              (EMPTY-TREE))))

(PROVE-LEMMA EXAMPLE-IS-NOT-SEARCH-TREE
  NIL
  (NOT
   (SEARCH-TREE-P
    (TREE-CONS 2
               (TREE-CONS 1
                          (EMPTY-TREE)
                          (TREE-CONS 3
                                     (EMPTY-TREE)
                                     (EMPTY-TREE)))
               (EMPTY-TREE)))))
```

Insertion into Trees

Next, a recursive function, `INSERT`, is defined which, given a tree, `TREE`, and a number, `NBR`, inserts the number into the tree so as to preserve both the properties of being an ordered tree and being a search tree. That is, (i) `INSERT` always returns a tree that contains `NBR`, the numbers in `TREE`, and no other numbers; (ii) an ordered tree is returned if `TREE` is an ordered tree; and (iii) a search tree is returned if `TREE` is a search tree.

```
(DEFN EMPTY-TREEP ( X )
  (OR (NOT (TREEP X))
      (EQUAL X (EMPTY-TREE)) ) )

(DEFN INSERT ( NBR TREE )
  (IF (EMPTY-TREEP TREE)
      (TREE-CONS NBR (EMPTY-TREE)(EMPTY-TREE))
      (IF (LESSP NBR (ROOT TREE))
          (TREE-CONS (ROOT TREE)
```

```
                    (INSERT NBR (LEFT TREE))
                    (RIGHT TREE))
          (IF (LESSP (ROOT TREE) NBR)
              (TREE-CONS (ROOT TREE)
                        (LEFT TREE)
                        (INSERT NBR (RIGHT TREE)))
              TREE))))
```

Verify that the tree constructed by INSERT contains exactly the numbers it should:

```
(DEFN OCCUR-IN-TREE ( NBR TREE )
  (IF (EMPTY-TREEP TREE)
      F
      (IF (EQUAL NBR (ROOT TREE))
          T
          (OR (OCCUR-IN-TREE NBR (LEFT TREE))
              (OCCUR-IN-TREE NBR (RIGHT TREE))))))

(PROVE-LEMMA OCCUR-INSERT
  (REWRITE)
  (IMPLIES (AND (NUMBERP I)
                (NUMBERP J))
           (EQUAL (OCCUR-IN-TREE I (INSERT J TREE))
                  (OR (EQUAL I J)(OCCUR-IN-TREE I TREE)))))
```

Demonstrate that INSERT does not destroy ordered trees and search trees:

```
(PROVE-LEMMA ORDERED-TREE-INSERT
  (REWRITE)
  (IMPLIES (AND (NUMBERP NBR)
                (ORDERED-TREE-P TREE))
           (ORDERED-TREE-P (INSERT NBR TREE))))

(PROVE-LEMMA SEARCH-TREE-INSERT
  (REWRITE)
  (IMPLIES (AND (NUMBERP NBR)
                (SEARCH-TREE-P TREE))
           (SEARCH-TREE-P (INSERT NBR TREE))))
```

Searching a Search Tree

The special structure of search trees makes it possible to give a more efficient algorithm for searching through these trees. The algorithm can compute which subtree to look in, and need not examine both subtrees.

```
(DEFN OCCUR-IN-SEARCH-TREE ( NBR TREE )
  (IF (EMPTY-TREEP TREE)
      F
      (IF (LESSP NBR (ROOT TREE))
          (OCCUR-IN-SEARCH-TREE NBR (LEFT TREE))
          (IF (LESSP (ROOT TREE) NBR)
              (OCCUR-IN-SEARCH-TREE NBR (RIGHT TREE))
              T))))
```

Verify that, when used on search trees, the newer search function produces the same rsults as the older one:

```
(PROVE-LEMMA OCCUR-SEARCH-TREE-LEFT
  (REWRITE)
  (IMPLIES (AND (NUMBERP NBR)
                (SEARCH-TREE-P TREE)
                (NOT (EQUAL TREE (EMPTY-TREE)))
                (LESSP NBR (ROOT TREE)))
           (EQUAL (OCCUR-IN-TREE NBR TREE)
                  (OCCUR-IN-TREE NBR (LEFT TREE)))))

(PROVE-LEMMA OCCUR-SEARCH-TREE-RIGHT
  (REWRITE)
  (IMPLIES (AND (NUMBERP NBR)
                (SEARCH-TREE-P TREE)
                (NOT (EQUAL TREE (EMPTY-TREE)))
                (LESSP (ROOT TREE) NBR))
           (EQUAL (OCCUR-IN-TREE NBR TREE)
                  (OCCUR-IN-TREE NBR (RIGHT TREE)))))

(PROVE-LEMMA OCCUR-TREE-SEARCH-TREE
  (REWRITE)
  (IMPLIES (AND (NUMBERP NBR)
                (SEARCH-TREE-P TREE))
           (EQUAL (OCCUR-IN-TREE NBR TREE)
                  (OCCUR-IN-SEARCH-TREE NBR TREE))))
```

Traversing a Search Tree

One of the nice properties of search trees is that the inorder traversal of such a tree produces an ordered list of the numbers in the tree. In this section such a traversal function is defined and the proof that it is correct discussed. A more complete listing of the events in the proof is in Appendix B.

- The recursive definition of the traversal:

```
(DEFN INORDER-TRAV ( TREE )
  (IF (EMPTY-TREEP TREE)
      NIL
      (APPEND (INORDER-TRAV (LEFT TREE))
              (CONS (ROOT TREE)
                    (INORDER-TRAV (RIGHT TREE)) ) ) ) )
```

- The definition of an ordered list:

```
(DEFN ORDERED-LIST ( LIST )
  (IF (EQUAL LIST NIL)
      T
      (IF (EQUAL (CDR LIST) NIL)
          T
          (IF (LISTP LIST)
```

```
          (AND (LESSP (CAR LIST)(CADR LIST))
               (ORDERED-LIST (CDR LIST)))
          F))))
```

- Two lemmas that verify that the traversal produces an ordered list of numbers and that the numbers in that list are exactly the numbers contained in the input tree:

```
(PROVE-LEMMA INORDER-ORDERED
  (REWRITE)
  (IMPLIES (SEARCH-TREE-P TREE)
           (ORDERED-LIST (INORDER-TRAV TREE))))

(PROVE-LEMMA OCCUR-MEMBER-TRAV
  (REWRITE)
  (EQUAL (OCCUR-IN-TREE NBR TREE)
         (MEMBER NBR (INORDER-TRAV TREE))))
```

Appendix A

These six events are essentially the input given by Boyer and Moore in [1] to the prover. (The only change is that the .'s in the original lemma and function names have been replaced by -'s.)

Note that the last event remains a theorem when the hypothesis, `(ORDERED-TREE X)`, is replaced by `(OTP X)`.

```
(ADD-SHELL OT
           ET
           OTP
           ((LT (ONE-OF OTP) ET)
            (LABEL (ONE-OF NUMBERP) ZERO)
            (RT (ONE-OF OTP) ET)))

(DEFN ORDERED-TREE ( X )
  (IF (NOT (OTP X))
      F
      (IF (EQUAL (ET) X)
          T
          (AND (ORDERED-TREE (LT X))
               (ORDERED-TREE (RT X))
               (OR (EQUAL (LT X)(ET))
                   (LESSP (LABEL (LT X))(LABEL X)))
               (OR (EQUAL (RT X)(ET))
                   (LESSP (LABEL X)(LABEL (RT X))))))))

(DEFN INSERT ( L X )
  (IF (NOT (OTP X))
      (ET)
      (IF (EQUAL X (ET))
          (OT (ET) L (ET))
          (IF (LESSP (LABEL X) L)
              (OT (LT X)
                  (LABEL X)
                  (INSERT L (RT X)))
              (IF (LESSP L (LABEL X))
                  (OT (INSERT L (LT X))
                      (LABEL X)
                      (RT X))
                  X)))))

(DEFN OCCUR-TREE ( L X )
  (IF (NOT (OTP X))
      F
      (IF (EQUAL X (ET))
          F
          (IF (EQUAL L (LABEL X))
              T
              (OR (OCCUR-TREE L (LT X))
                  (OCCUR-TREE L (RT X)))))))

(PROVE-LEMMA ORDERED-TREE-INSERT
  (REWRITE)
  (IMPLIES (AND (NUMBERP L)
                (ORDERED-TREE X))
           (ORDERED-TREE (INSERT L X))))

(PROVE-LEMMA OCCUR-TREE-INSERT
  (REWRITE)
  (IMPLIES (AND (NUMBERP L)
                (NUMBERP K)
                (ORDERED-TREE X))
           (EQUAL (OCCUR-TREE L (INSERT K X))
                  (OR (EQUAL L K)
                      (OCCUR-TREE L X)))))
```

Appendix B

The successful submission to the prover of these events, together with a few of the events listed earlier in the paper, constitute a mechanical proof that the traversal function is correct.

```
(DEFN INORDER-TRAV ( TREE )
  (IF (EMPTY-TREEP TREE)
      NIL
      (APPEND (INORDER-TRAV (LEFT TREE))
              (CONS (ROOT TREE)
                    (INORDER-TRAV (RIGHT TREE)) ) ) ) ) )

(DEFN ORDERED-LIST ( LIST )
  (IF (EQUAL LIST NIL)
      T
      (IF (EQUAL (CDR LIST) NIL)
          T
          (IF (LISTP LIST)
              (AND (LESSP (CAR LIST)(CADR LIST))
                   (ORDERED-LIST (CDR LIST)))
              F))))

(DEFN EMPTY-LISTP ( X )
  (OR (NOT (LISTP X))
      (EQUAL X NIL)))

(DEFN LESSP-LIST-NBR (LIST NBR)
  (IF (EMPTY-LISTP LIST)
      T
      (AND (LESSP (CAR LIST) NBR)
           (LESSP-LIST-NBR (CDR LIST) NBR))))

(DEFN LESSP-NBR-LIST (NBR LIST)
  (IF (EMPTY-LISTP LIST)
      T
      (AND (LESSP NBR (CAR LIST))
           (LESSP-NBR-LIST NBR (CDR LIST)))))

(PROVE-LEMMA ORDERED-APPEND
  (REWRITE)
  (IMPLIES (AND (LESSP-LIST-NBR LIST1 NBR)
                (LESSP-NBR-LIST NBR LIST2)
                (ORDERED-LIST LIST1)
                (ORDERED-LIST LIST2))
           (ORDERED-LIST (APPEND LIST1 (CONS NBR LIST2)))))

(PROVE-LEMMA LESSP-LIST-NBR-INORDER
  (REWRITE)
  (IMPLIES (LESSP-TREE-NBR TREE NBR)
           (LESSP-LIST-NBR (INORDER-TRAV TREE) NBR)))

(PROVE-LEMMA LESSP-NBR-LIST-INORDER
  (REWRITE)
  (IMPLIES (LESSP-NBR-TREE NBR TREE)
           (LESSP-NBR-LIST NBR (INORDER-TRAV TREE))))

(PROVE-LEMMA INORDER-ORDERED
  (REWRITE)
  (IMPLIES (SEARCH-TREE-P TREE)
           (ORDERED-LIST (INORDER-TRAV TREE))))

(PROVE-LEMMA OCCUR-MEMBER-TRAV
  (REWRITE)
  (EQUAL (OCCUR-IN-TREE NBR TREE)
         (MEMBER NBR (INORDER-TRAV TREE))))
```

References

[1] R. S. Boyer and J S. Moore, "A fully automatic proof of the correctness of an ordered tree insertion function," Technical Report ICSCA-CMP-36, Institute for Computing Science and Computer Applications, University of Texas at Austin, Austin, Texas, 1983.