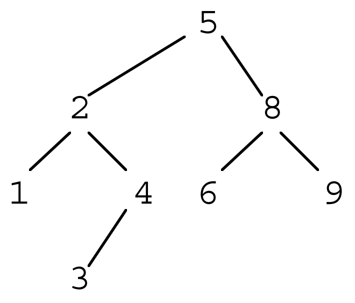
Using Hidden Functions

The following example is taken without change from the Guttag & Horowitz chapter. It is a specification of binary search trees. It is derived from a previous example — the binary tree. An (ordinary) binary tree is a familiar structure where each node has a left and right subtree. A binary search tree is a binary tree where every item in the left subtree of each node is “less than” the parent and every item in the right subtree is “greater than” the parent. Binary search trees hence support a more efficient search for whether an item occurs in a tree. In order for the specification to make sense, it is required that the type of items in the search tree have a comparison operation, written here as '<', that orders items but is otherwise unrestricted (e.g., magnitude of numbers, lexicographical order of strings, etc.). In this example it is natural to incorporate a constructor function (MAKE) for expressiveness that must be forbidden to client applications for correctness reasons.

Graphically we can depict a Bstree in diagram form, e.g.,



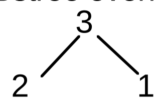
The ADT is formulated so that there are suitable equations to facilitiate a fast (log-time) implementation of the ISIN function.

```
type Bstree [item]
   declare EMPTYTREE( ) → Bstree
           *MAKE(Bstree,item,Bstree) → Bstree
            ISEMPTYTREE(Bstree) → Boolean
            LEFT(Bstree)→ Bstree
            DATA(Bstree)→ item U {UNDEFINED}
            RIGHT(Bstree)→ Bstree
            ISIN(Bstree,item) → Boolean,
            INSERT(Bstree,item) → Bstree;
   for all  l,r ∈ Bstree, d,e ∈ item let
            ISEMPTYTREE(EMPTYTREE) = true
            ISEMPTYTREE(MAKE(l,d,r)) = false
            LEFT(EMPTYTREE) = EMPTYTREE
            LEPT(MAKE(l,d,r)) = l
            DATA(EMPTYTREE) = UNDEFINED
            DATA(MAKE(l,d,r)) = d
            RIGHT(EMPTYTREE) = EMPTYTREE
            RIGHT(MAKE(l,d,r)) = r
            !SIN(EMPTYTREE,e) = false
            ISIN(MAKE(l,d,r),e) =
                 if d=e then true
                     else if d<e then ISIN(r,e) else ISIN(l,e)
            INSERT(EMPTYTREE,e) = MAKE(EMPTYTREE,e,EMPTYTREE)
            INSERT(MAKE(l,d,r),e) =
                 if d=e then MAKE(l,d,r)
                      else if d<e then MAKE(l,d,INSERT(r,e))
                                  else MAKE(INSERT(l,e),d,r)
   end
end Bstree
```

In this ADT INSERT is an ordered tree constructor, but MAKE is a general tree constructor. That is, INSERT applied to a Bstree yields a Bstree, while MAKE applied to Bstrees need not be a Bstree. For instance, the tree

```
T = MAKE(MAKE(EMPTYTREE,2,EMPTYTREE),
            3,
          MAKE(EMPTYTREE,1,EMPTYTREE))
```

is not a Bstree even though its arguments are — in diagram form it is



For such trees ISIN performs incorrectly, e.g., ISIN(T,1) = **false**.

Thus, the MAKE function cannot be permitted as an unrestricted generator operation. It is restricted by designating it as a *hidden function*. It can be used internally to describe the structure of a known Bstree, but it cannot be used as a valid (i.e., visible) Bstree operation. The set of valid constructors is {EMPTYTREE, INSERT}.

Notice that if the problematic MAKE operation were removed from the Bstree specification, the ISIN operation could still be easily expressed by the equations

```
ISIN(EMPTYTREE,x) = false
ISIN(INSERT(t,x), y) =
        if DATA(INSERT(t,x)) = y
            then true
            else if DATA(INSERT(t,x)) < y
                    then ISIN(RIGHT(INSERT(t,x)), y)
                    else ISIN(LEFT(INSERT(t,x)), y)
```

But with MAKE removed we have an “incomplete” specification — formally, it is not sufficiently complete, e.g.,

DATA(INSERT(t,x)) = ??
RIGHT(INSERT(t,x)) = ??
ISIN(LEFT(INSERT(t,x)), y) = ??