CMPSCI 250 Discussion #8: Boolean Expressions

Individual Handout
David Mix Barrington
25 March 2013

Here is a real Java class definition for **boolean expressions**, similar to the arithmetic expressions discussed in lecture and to the "propositional WFF's" in example 8 on page 350 of Rosen. Note that there are *not* separate node and tree objects – a tree can be a single leaf node or a composite node with two subtrees.

```
public class BooleanExpression {
   public static final int AND = 1; // AND operator
   public static final int OR = 2;  // OR operator
   public static final int NOT = 3; // NOT operator

   boolean isLeaf;                  // true if this is a one-node tree
   boolean leafValue;               // value of tree if it has just one node
   int operator;                    // value must be AND, OR, or NOT
   BooleanExpression left, right;   // subexpressions if !isLeaf

   public BooleanExpression (boolean arg) {
   // create one-node tree with given value
      isLeaf = true;
      leafValue = arg;
      left = right = null;}

   public BooleanExpression (int op, BooleanExpression leftArg,
                             BooleanExpression rightArg) {
   // create tree with given operator, left argument, right argument
      isLeaf = false;
      operator = op;
      left = leftArg;
      right = rightArg;}

   public boolean getIsLeaf () {return isLeaf;}
   public boolean getLeafValue ( ) {return leafValue;}
   public int getOperator () {return operator;}
   public BooleanExpression getLeft () {return left;}
   public BooleanExpression getright () {return right;}}
```

We can think of a boolean expression as a tree, where leaf nodes are labeled by boolean values and internal nodes are labeled by boolean operators. We can use recursive methods to find out properties of these expressions. Once we have a correct recursive definition for the property, it is easy to write a method that returns the correct value when `isLeaf` is true and computes the correct value from the answers to recursive calls to `right` and `left` when `isLeaf` is false.

Note that a NOT gate has only a left argument – its right argument should be `null`.

Let's see how to use recursion to determine the size (number of nodes) in a `BooleanExpression` object. We first need a base case, for when the calling tree is a leaf. This is easy, as a leaf is a tree of size 1. If the tree is not a leaf, it has a left subtree and maybe a right subtree – the latter is `null` if the operator is NOT. So we add up one for the node itself, the size of the left subtree, and (if it exists) the size of the right subtree:

```
public int size ( ) {
    if (isLeaf) return 1;
    int sum = 1;
    sum += left.size( );
    if (operator != NOT) sum += right.size( );
    return sum;}
```

Writing Exercise: Write (in real Java) the following three additional methods to be added to this class:

1. A method `leaves` that returns an `int` giving the number of leaves in the calling expression's tree.
2. A method `depth` that returns an `int` giving the depth of the tree, which is the number of nodes in the *longest* directed path from the root node to any leaf. Note that if the root node *is* a leaf, the only path from the root to a leaf has size 0 because it has no edges.
3. A method `eval` to return the boolean value of the calling expression.