COMPSCI 250 Discussion #7: Boolean Expressions

Individual Handout
David Mix Barrington and Ghazaleh Parvini
1 November 2023

Here is a real Java class definition for **boolean expressions**, somewhat similar to the definition of Lisp lists in Section 4.10 (which we aren't doing this semester anyway). 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.

If your first programming language is Python, the main difference with this Java code is that every variable and every function (called a "method") has an explicit type, and that the functions here are all declared as "public" so that they can be called from any other object.

```
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`.

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

1. A method `size` that returns an `int` giving the number of nodes in the calling expression's tree. (We'll mostly do this on the blackboard.)
2. A method `leaves` that returns an `int` giving the number of leaves in the calling expression's tree.
3. 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.
4. A method `eval` to return the boolean value of the calling expression.
5. (if time) A method `toString` that returns a `String` representing the expression, in an infix format like "`NOT ((true OR false) AND (false OR true))`". We're not worried about redundant parentheses.