Data Structures and Algorithms

Deep Copying

Alan P. Sexton

University of Birmingham

Spring 2019

Assignment vs Deep Copy

From student questions I have received, I believe that there are still quite a few students who are confused about the differences between simply assigning a `PLTreeNode` object to a `child1` or `child2` field in a `PLTreeNode` object, or assigning a deep copy of such a `PLTreeNode` to the child field in question. That is, the difference between:

- `child1 = child2;`

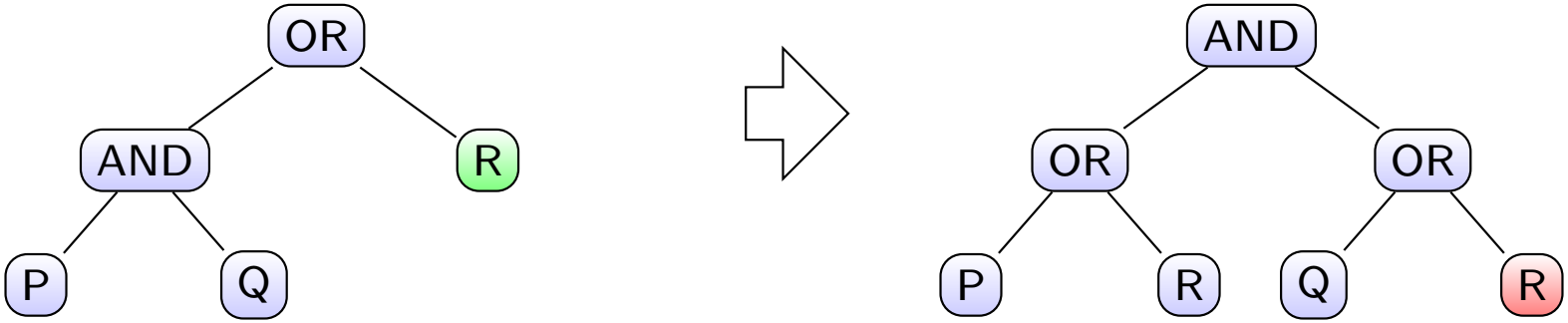
and

- `child1 = new PLTreeNode(child2);`

The short answer is that, if you are leaving the old copy in the tree, you should use the deep copy constructor. If you are **NOT** leaving the old copy in the tree, just assign it.

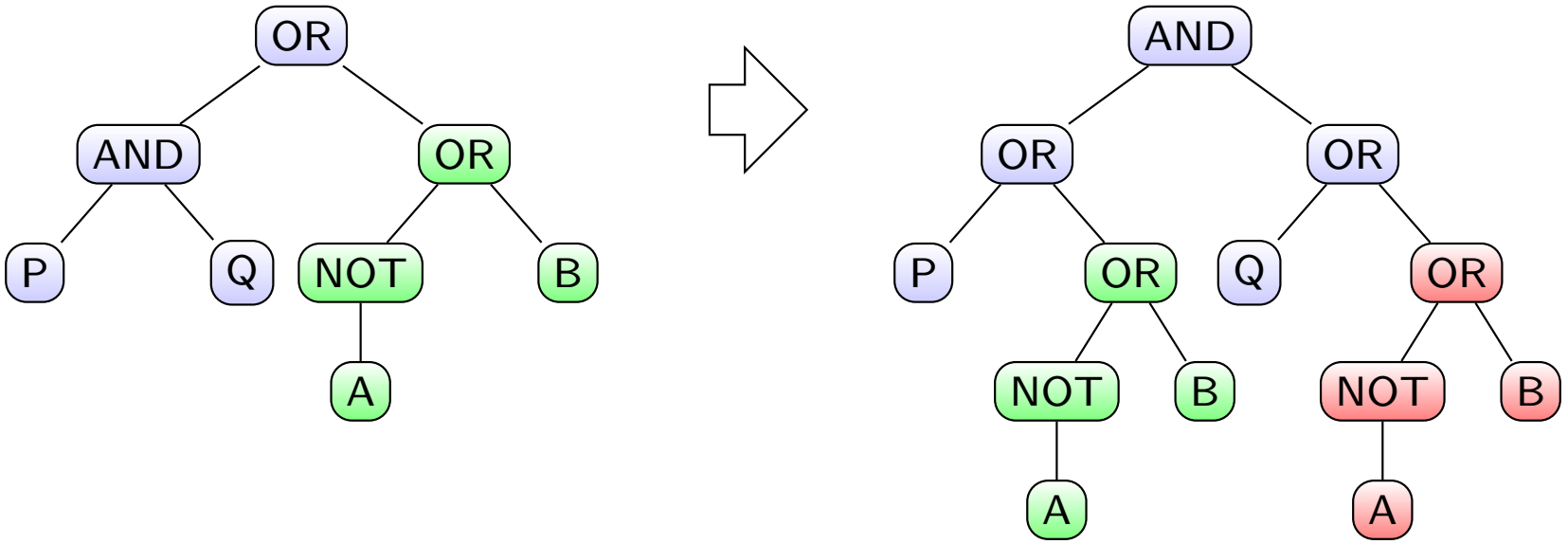
pushOrBelowAnd()

One of the cases that has to be dealt with in `pushOrBelowAnd()` is as follows:



pushOrBelowAnd(): larger case

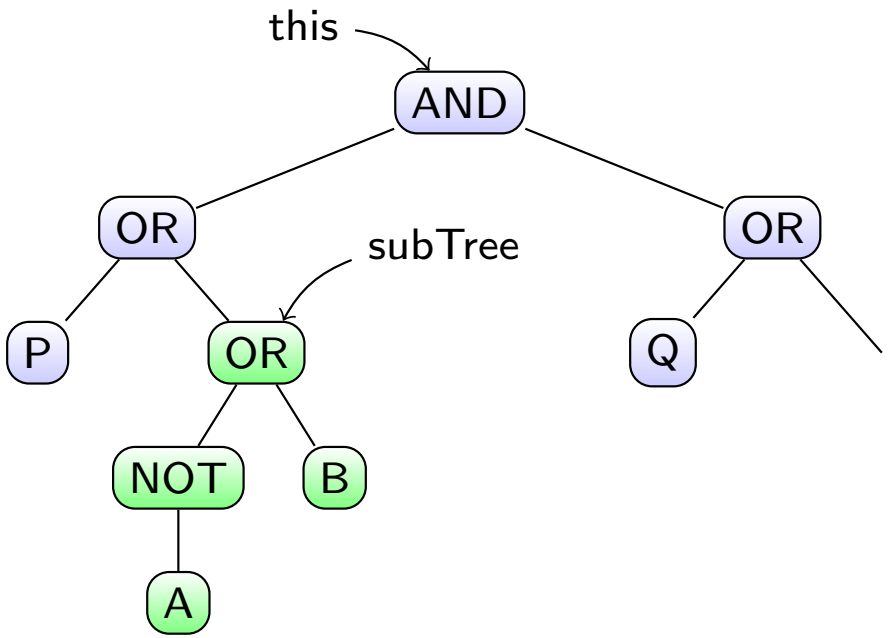
Consider a larger sub-tree in place of R:



`pushOrBelowAnd()`: partially completed

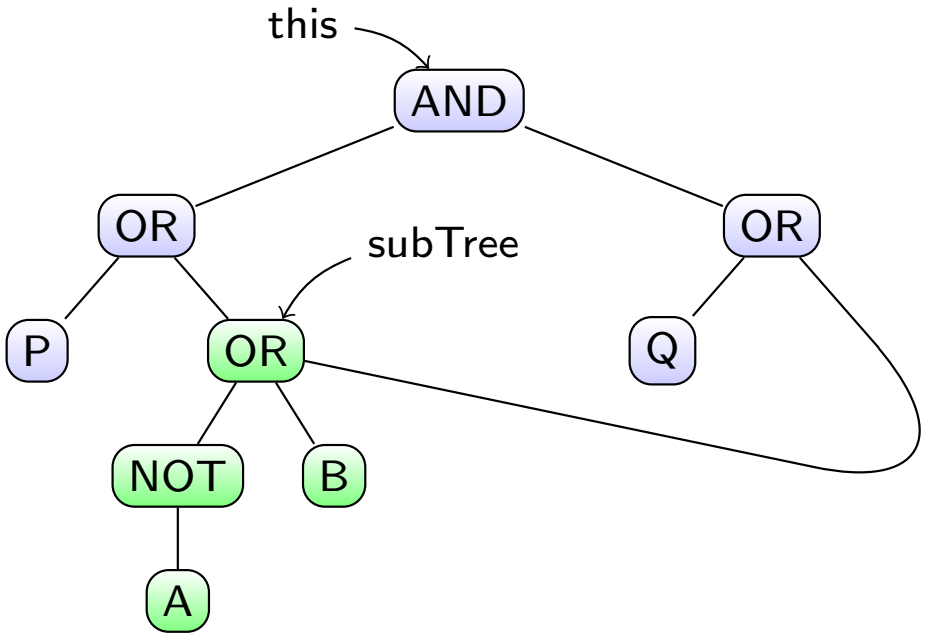
Now consider the situation when you are partially through the transformation:

- "`this`" is pointing to the current `PLTreeNode`
- You have already restructured the tree correctly, but have still to put the copy of the green sub-tree into position
- The variable `subTree`, which is the same as `child1.child2`, is pointing to the green sub-tree.



Simple assignment

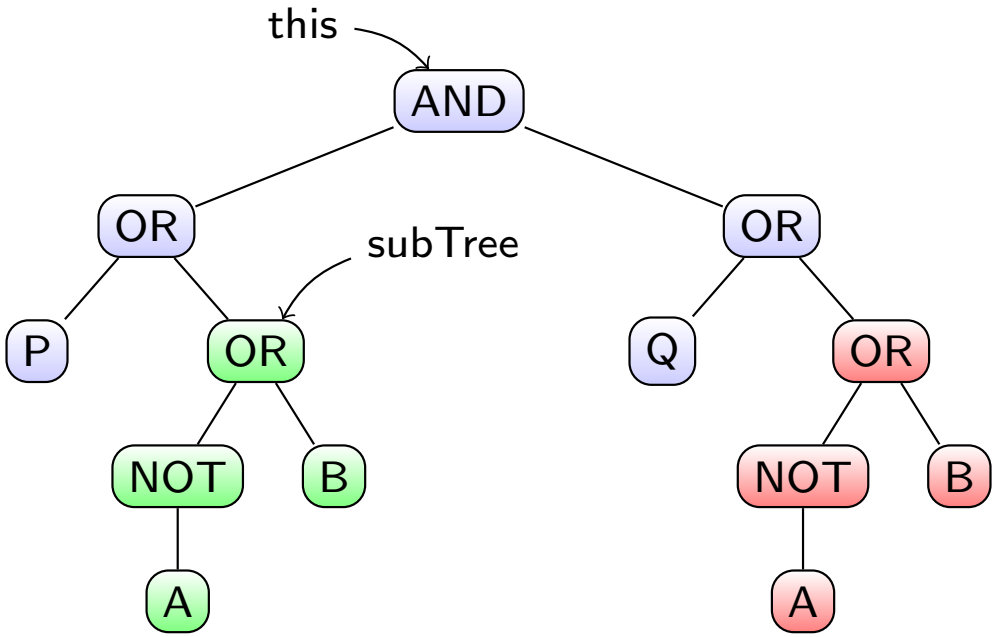
If you now execute: `child2.child2 = subTree;`



- `child1.child2` and `child2.child2` point to the **same** `PLTreeNode`
- This means that things seem okay, e.g. `toString...` methods still work, but anything that changes the subtree `child1.child2`, **ALSO** changes the `child2.child2` subtree

Deep Copy

If you execute: `child2.child2 = new PLTreeNode(subTree);`

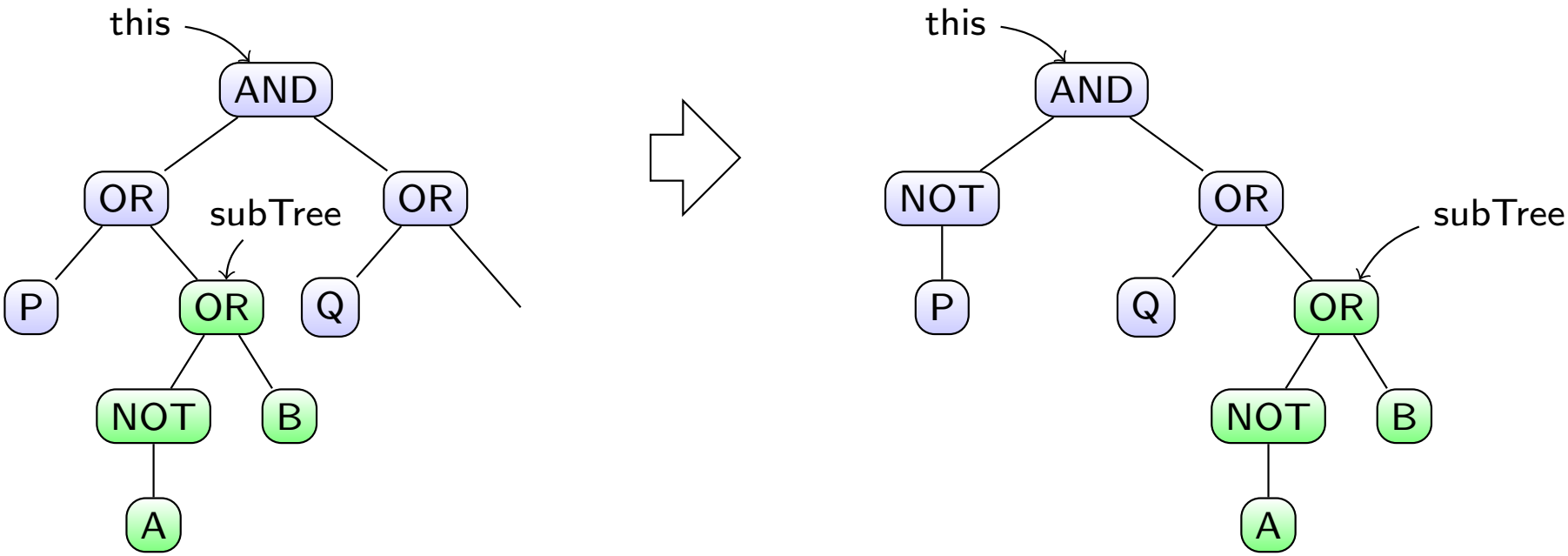


- The red sub-tree is a **DEEP COPY** of the green sub-tree: the nodes have the same `type` values, and the same tree structure, but they are different objects in memory
- `child1.child2` and `child2.child2` point to **DIFFERENT** sub-trees that have the same values
- No more problems with modifying one sub-tree changing another

Moving a subtree

What if you want to **MOVE** a subtree: say turn the left `OR` node into a `NOT`, and move the current `child1.child2` subtree to the right child of the right `OR` node:

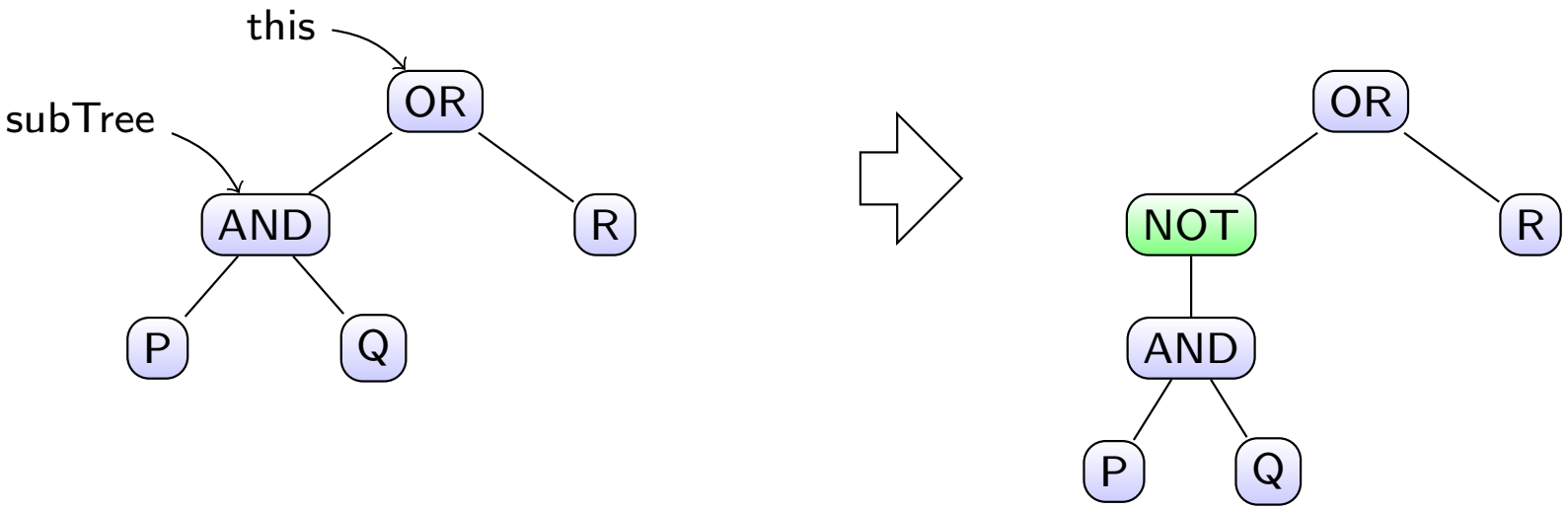
```
child2.child2 = subTree ;
child1.child2 = null;
child1.type = NodeType.NOT;
```



Since we are **MOVING** the subtree, we can safely assign `subTree` to `child2.child2` because we will not end up with two different pointers to the same sub-trees in different parts of the whole tree

Inserting a new node above a sub-tree

What if you want to insert a new node **ABOVE** a subtree?



Here we do need to create a new `PLTreeNode` for the `NOT`, but we are just moving the old `subTree`, so we do **not** need to make a deep copy:

```
this.child1 = new PLTreeNode(EntryNode.NOT,
    subTree, null);
```