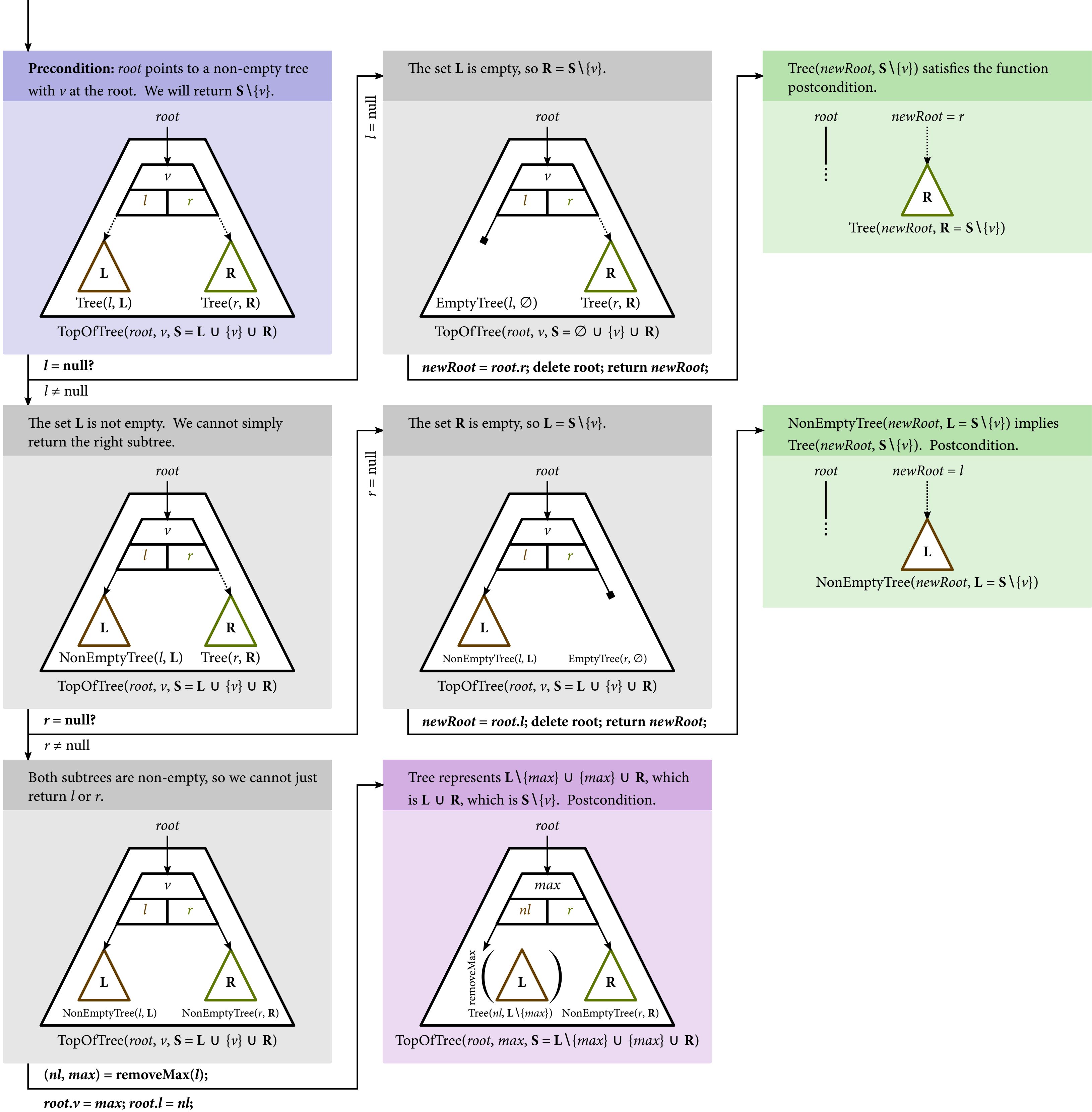
Precondition: root points to a non-empty tree with v at the root. We will return S \ {v}.
root
v
l
r
L
R
Tree(l, L)
Tree(r, R)
TopOfTree(root, v, S = L ∪ {v} ∪ R)
l = null?
l ≠ null
l = null
The set L is empty, so R = S \ {v}.
root
v
l
r
R
EmptyTree(l, ∅)
Tree(r, R)
TopOfTree(root, v, S = ∅ ∪ {v} ∪ R)
newRoot = root.r; delete root; return newRoot;
Tree(newRoot, S \ {v}) satisfies the function postcondition.
root
newRoot = r
R
Tree(newRoot, R = S \ {v})
The set L is not empty. We cannot simply return the right subtree.
root
v
l
r
L
R
NonEmptyTree(l, L)
Tree(r, R)
TopOfTree(root, v, S = L ∪ {v} ∪ R)
r = null?
r ≠ null
r = null
The set R is empty, so L = S \ {v}.
root
v
l
r
L
NonEmptyTree(l, L)
EmptyTree(r, ∅)
TopOfTree(root, v, S = L ∪ {v} ∪ R)
newRoot = root.l; delete root; return newRoot;
NonEmptyTree(newRoot, L = S \ {v}) implies Tree(newRoot, S \ {v}). Postcondition.
root
newRoot = l
L
NonEmptyTree(newRoot, L = S \ {v})
Both subtrees are non-empty, so we cannot just return l or r.
root
v
l
r
L
R
NonEmptyTree(l, L)
NonEmptyTree(r, R)
TopOfTree(root, v, S = L ∪ {v} ∪ R)
(nl, max) = removeMax(l);
root.v = max; root.l = nl;
Tree represents L \ {max} ∪ {max} ∪ R, which is L ∪ R, which is S \ {v}. Postcondition.
root
max
nl
r
removeMax
L
R
Tree(nl, L \ {max})
NonEmptyTree(r, R)
TopOfTree(root, max, S = L \ {max} ∪ {max} ∪ R)