Terms:
Current node = this = the node that is currently being traversed
op(*) = an Op node whose operation type is *
<op> = a function whose behavior is that of the kind described. (so like  if op = “+”, then 3 <op> 2 is just 3+2)

**Applying operations to number nodes (carl):**
If we have something like 3+3, we want to be able to replace that with just 6.
If the node we’re traversing is an Operation node, the left node is a Number node, and the right node is a Number node, then replace the current node with a Number node whose Value is equal to left.value <op> right.value.

case: on an op node and we have two children with Number nodes
action: replace current node with new number node whose value is this.left.value <op> this.right.value

**Distributing a multiplication (Susie):**
If we have 3*(x+y), we want 3*x + 3*y
If we’re on a Multiplication node (X) with any left child (Y) and the right child is a Parens node whose child is a Plus node with a left child (L) and a right child (R), then replace X with a Plus node (Z) where Z has a [left child that is a Parens node with a child Multiplication node with a left child Y and a right child L] and a [right child that is a Parens node with a child Multiplication node with a left child Y and a right child R]

(**Note that we may want to play around with the extra parens nodes and see if they’re really necessary or not. Perhaps dissolving them if if they only contain one term or something? idk)

(**Make 2 of these cases: one for distributing the left to the right, is possible, and one for distributing the right to left: For instance: (2x+3)*(4x+5) → (2x+3)*4x + (2x+3)*5 **OR** 2x*(4x+5) +3*(4x+5), depending on the way we distribute

case: on an op(*) node and either left or right is a () node with a + child or - child
action:

**Commuting operations (Bryant):**
If we have 2+3, we want to be able to do 3+2
If we’re on a Plus node or a Multiplication node, we change the left Number node to the right Number node and the right Number node to the left Number node.

case: on a op(*) or op(+) node
action:

**Commuting through tree structure (Carl):**
if we have [a*[b*c]], we want to be able to replace that with [b*[a*c]] or [c*[b*a]]
If we're on a Plus or Multiplication node, and one of the children is the same type of node, then we can swap our other child node for either of their child nodes.

If we're on a Plus or Multiplication node, and if one of the children has a parentheses node, then we swap the the left Number node with either of the grandchildren nodes of the parentheses node.

case: on a op(*) or op(+) node that has a child of the same type
action:

**Cancelling (Carl):**
3/3 =1, 3-3=0
If we are on a subtraction or division node, and our left child is isomorphic to our right child, then we can substitute this node with a 0 or a 1 (respective to the order they were just mentioned)

(**Note: use node.isEqual(otherNode))
case: on an op(-)
action:

**Useless parentheses (Bryant):**
(2)*x = 2*x
If we're on a Parens node whose child is a Function node, another Parens node, a Number node, a Variable node, or an Operation node of type Exponent, then we can dissolve the parentheses around that node.
(Parentheses, however, may be kept in there for visual purposes sometimes)

case:
action:

**Applying a hyperoperator (carl)**
If we have some kind of structure like (x+x+x+x+x) then we want to be able to change that into 5*x.

maybe a solution:
If on a Plus node, level down to the end of the tree from the left node. Keep count the number of levels you go down and add one. With that number we replace the Plus node's left child with this number and the right node will stay the same node.
<Not sure how this would work exactly. I'll think about it later. It's kinda the reverse of the distributive property>

case:
action:

**Reduce a hyperoperator (carl)**
If we have 3*x, we want to be able to turn it into x+x+x
If we have x^4, we want to be able to turn that into x*x*x*x

If on a Exponent Node or Multiplication node, we expand the tree from the left child X-2 amount of times where right right child will always be a Variable node and the all left children will be Operation nodes until the last level. At the last level, both children will be Variable nodes.

case:
action:

**Substitute a function (carl)**
If we have defined f(x,y,z) := x^2 + y^2 + z^2, then we should be able to replace it in an equation like f(2m, 3j, 5q) + 26 = 18 such that we get ((2m)^2 + (3j)^2 + (5q)^2) + 26 = 18

When on a Function node, we replace all Variable nodes with whatever it is being replaced with. When on an Operation node, the Variable nodes will be either the grandchildren or great grandchildren of this node. When traversing this part of the tree and a Variable node is found we replace this Variable node with whatever it is being replaced with.

case:
action:

**Substitute an expression (Carl)**
If we have an equation A somewhere of the form 5 + x - y = 10, and somewhere else we have the equation B like z = (5+x-y)^2, then we should be able to replace 5 + x -y with 10, so that B becomes z = (10)^2

case:
action:

**Reducing exponents (susie)**
If we have x^3^4  we should be able to replace it with x^12 or if we have x^3*x^2 we should be able to replace it with x^5

Note: exponentiation is left-associative
case:
action:

**Inverting operators (susie)**
If we have an equation 3/2 we should be able to replace it with 3*(2^-1).
(and vice versa)

case:
action:

**Turning addition of a negative into subtraction (Bryant):**
If we have 5 + (-8), then this should be able to become 5-8

case:
action:

**Turning subtraction into addition of a negative (bryant):**
if we have 5-8 then this should be able to become 5+(-8)

case:
action:

**Factor out a -1 on a negative number (susie):**
If we have -8, we should be able to make this -1*8

case: on a numberNode whose value is less than 0
action:

**Identity operations (carl):**
x*0 = 0 no matter what,
x^0 = 1 (unless x = 0 itself usually),
x*1 = x,
x+0 = x,
x-0 = x,
x/1 = x
x/x = 1, (covered in canceling also)
x-x = 0,(covered in canceling also)