# ACE Coursework2 Report

*Shuchang Huang 20214805*

## 1. Data Structure and Problem Formulation

Tree has been selected as the data structure used in this coursework. The reason is that a logic expression can be seen as two propositions connected by a logical operator (if we take negation as the intrinsic attribute of a proposition), which can be naturally modeled as two child nodes connected by a root node. That is, we transfer the problem to a tree structure transformation problem. Here, the negation is expressed using a Boolean value 'isNegated'. A proposition with true 'isNegated' value means that negation is performed on this proposition. A logical operator with true 'isNegated' value means that negation is performed on the proposition formed by this operator and its two child nodes. An example is shown in Figure 1.

~ [ [ [ A ] & [ ~B ] ] | [ [ C ] & [ D ] ] ]

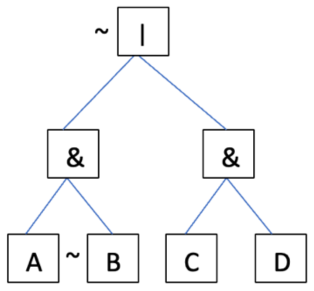


Figure 1

## 2. Algorithm and why it works correctly

The construction of the tree is based on the observation that a single and complete logical expression that can be extracted from the string to form a node always contains equal numbers of left parathesis and right parathesis. Thus, this algorithm counts the parathesis to construct the tree.

When converting to DNF, this program follows the standard procedure as illustrated in the coursework specification but performs the Collect Terms Step first to deal with cases such as ([ [ a ] | [ b ] ] & [ ~ [ [ a ] | [ b ] ] ]). In case that after one round of this procedure, some sub-expression produced can be further converted, a loop is used to repeat the procedure until the output remains unchanged. In the whole algorithm, recursion is applied. We apply the procedure ("toDNF") on the current root node and its child nodes until reach external nodes. Now, let's see how each step is performed using tree structure:

1. Eliminate (perform when -> appears)
   Here, we only need to change the value in the root node from "->" to "|" and negate the "isNegated" value of the left child node. (Recursion applied until external)

2. Negate
   The Double Negation Rule is applied all along the procedure by using the attribute of Boolean values, so no special operations need to be done considering this rule. Here, similarly, we change the value in the root node according to De Morgan's Law and change the "isNegated" value of its two child nodes. (Recursion applied until external)

3. Distribute (performs when root node == "|" and one of its child nodes == "&")
   Since in step 2, we iterate until no negation is performed on internal nodes, we do not need to consider changing the "isNegated" value on any nodes in this step. (For example, expression such as [a&(~(b|c))] won't appear) Thus, we construct new child nodes ("isNegated" value set to false by default) based on the Distributive Law. (Recursion applied until external)

4. Collect
   Here, the terms are collected according to situations listed on the coursework specification. Apart from it, one set of additional cases is also considered, an example is shown in figure 2. (Recursion applied until external)

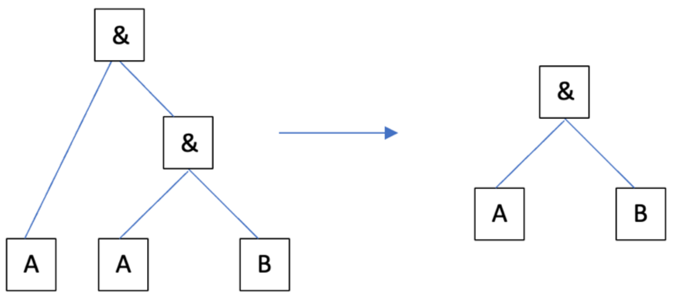


Figure 2

Constructing the output string from the tree is simply tree traversal.

3. Time Complexity (only considering the algorithm itself)

Assume there are N logical operators (excluding NOT), so the number of atomic propositions = [N+1,2N] (2N is when all left child nodes are external, N+1 is as shown in Figure 2, right). Thus, the tree we get has [2N+1,3N] nodes, of height [logN,N].

Constructing a tree is of time complexity $O(N)$, if the time for dividing a string to construct a node is 1.

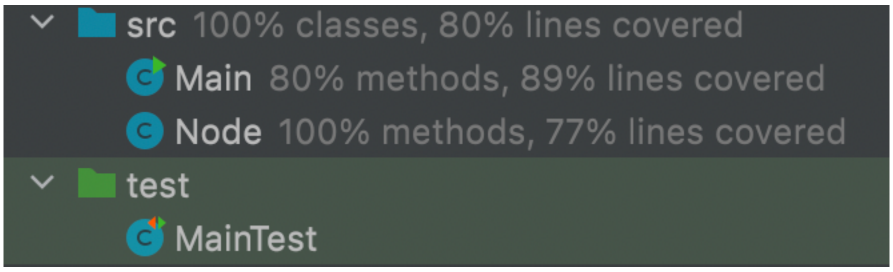
For the "toDNF" procedure, time complexity of each step is $O(N)$, since in Main.java, DNF method, we recursively call toDNF, time complexity becomes $O(N^2)$.

Constructing the output string is of time complexity $O(N)$, if the time need for forming a string from a node is 1.

In total, Time Complexity is $O(N^2)$.

4. Test

In total,43 tests are tried and they all passed, with all functional lines covered (only 80% methods in Main shown in the figure is because the input scanner function is tested manually, 23% line in Node uncovered are either blank or comments).



```
src 100% classes, 80% lines covered
    Main 80% methods, 89% lines covered
    Node 100% methods, 77% lines covered
test
    MainTest
```

Here listed some example test cases used:
1. [ [ [ A ] & [ B ] ] & [ [ C ] & [ D ] ] ] | [ [ A ] & [ E ] ]
2. [ [ A ] | [ B ] ] & [ [ C ] | [ D ] ]
3. ~ [ [ ~ [ A ] ] -> [ B ] ]