**Converting a Formula to DNF**

21800294 Park Suah

<Introduction>

DNF Converter is a program that transforms a propositional formula into a Disjunctive Normal Form, called DNF. DNF is a two-level propositional formulas which has a form of where or for an atomic propositional variable . Every propositional formula can be converted to an equivalent DNF formula.   
For a given formula, my program will print out an equivalent DNF formula and any satisfiability checking result. At here, it prints out a solution if the formula is satisfiable. Otherwise, it prints “UNSAT”.

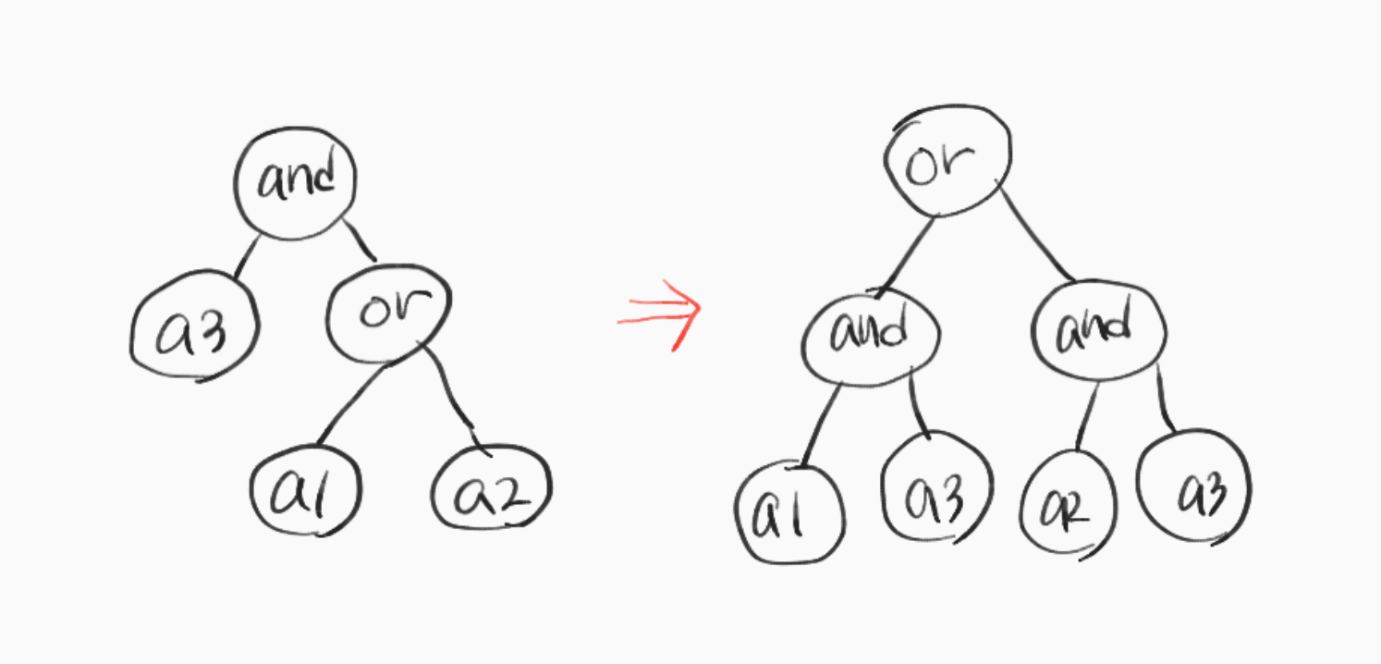
<Approach>

When given a propositional formula, program should change and print it in DNF format and print satisfiability checking result. All input must be prefixed in the parenthesis. At output expression, the numbers in the same row are all connected by conjunction, and each row is connected by disjunction. And the program must print out an error message if the given input does not follow that rule.

* Solution Design

DNF Converter follows this procedure to solve the problem.

1. Parse the propositional formula and push it in tree like structure.  
   First, input will be split by space so that each expression goes into binary tree structure. This structure has three data. (key, pointer of left node, pointer of right node) We have some rules in here. Each propositional variable will be a node and has it’s own value. (AND : 1, OR : -1, NOT : 0) First, if the node key is zero, the subtree must be created on the left node. Second, if a node key is -1 or 1, it has to check if the left node or right node is full. If the left node is empty, make a subtree on the left node, and if the left node is full, then make the subtree under the right node. Each variable has to pass this procedure.
2. Negate all child nodes of ‘NOT’ nodes and delete ‘NOT’.  
   After the all variables go into tree, the NOT node must be eliminated. Before that, we have to change the child nodes of NOT node to be negated. AND should be OR, OR should be AND, and each numerical variable which is negative number should be changed to positive number and positive number to negative number. And then, when we find the NOT node while we traverse the tree, we have to delete the node.
3. Apply the distributive law.  
   To convert NNF to DNF, the root node key must be OR node at finally. For this, we must OR node to the root, and apply the distributive law. If the root node key is AND and one of its child node key is OR, then their node should be swapped. And child of root node which is just numerical variable, should insert each of child node of child of root node. I will show the image to make it easy to understand.



1. Print the satisfiability checking result.  
   Because it is a DNF, when only one line of result is true, the other line is irrelevant to what variable comes out. We will take the results from the above output and print the numbers from the first row as they are, and then we will print the numbers not existing at the first row as they are if we find them. We may check the tautology of DNF formula by converting it into CNF.

<Discussion>

Whenever a variable comes in, it will take a long time to find a place to enter the tree from the top. I have to think of a way to reduce this useless procedure. Actually I couldn’t make this program perfectly. However, the process of fine-tuning large problems and more granularity to solve the problems that arise within the program has enabled me to develop the power to work out algorithms and think about them. Of course, it is another problem that makes things work by implementing them in code, and I didn’t get it done in the end, but I learned a lot from thinking about how to solve problems.