Homework 2 - Converting a formula to DNF

Yunsubin21800491

1. Problem Definition

An Input must be propositional formula logic that consists only of ‘a_n’, ‘and’, ‘or’ and ‘not’, where n is a integer (0<n<1024). Also, input has to able to represent tree data structure using the recursive program and convert to text again. If input can’t represent a data structure, it has the wrong syntax. In this case, show the error message.

Negation Normal Form (NNF) is the propositional logic that has no negation in front of conjunction or disjunction. For making NNF logic, Using De Morgan's law, not only disjunction in the propositional logic can be converted as conjunction, but also conjunction can be converted as disjunction.

Disjunctive Normal form (DNF) is the propositional formula that consists of disjunction. Simply, it is the propositional formula that has no conjunction inner any disjunction. Have to convert NNF into DNF. It must be processed by the recursive program.

After making a DNF formula, now determine whether it is satisfiability or not like z3 program.

Output is consist of numbers and line. Each line separates with conjunction. Each line consists of a positive integer and a negative integer which means atomic proposition value a_n and ¬a_n. And the last line shows proposition value True or False. If the arbitrary number is negative, it means false. If the arbitrary number is positive, it means true. The front line of the last line must print ‘0’ in order to distinguish with the DNF formula.

2. Solving Problem

1. Translate NNF formula to DNF

To convert NNF to DNF, downgrade all conjunction recursively as possible. To solve it, use distribute rule to convert conjunction to disjunction.

```
Tree DNF(Tree tree) {
  switch(tree->type)
    case Atom, Not:
      return tree;
    case Or:
      return DNF(tree.left) ∨ DNF(tree.right);
    case And:
      return Distribute(DNF(tree.left), DNF(tree.right);
}
```

```
Tree Distribute(t1, t2) {
  if(t1.type = Or)
    return Distribute(t1.left, t2) ∨ Distribute(t1.right, t2);
  if(t2.type = Or)
    return  Distribute(t2.left, t1) ∨ Distribute(t2.right, t1);
  return    t1 ∧ t2;
}
```

2. Decide Solution of DNF.

DNF consist of a disjunction between each clause. And each literal in a clause connects conjunction. Assume that X is DNF and C_i is a clause, where i is an integer(0<i≤n). X can be represented by C_1 ∨ C_2 ∨ C_3…. C_i consists of a literal connected by conjunction. In order to make DNF to be satisfiable, at least one of the clauses must be true. The case of DNF to be unsatisfiable is when there are opposite values of a certain literal and a literal within all clauses. That means, if clause C_i includes l_x and ¬l_x, clause C_i becomes false. Therefore, If clause C_i is false about all numbers, DNF determines UNSAT.

The first step is Checking all clause whether exist l_x and ¬l_x. If all clause has l_x and ¬l_x, then return UNSAT. If not, move on to the next step. The second step is making a decision each literal whether True or False. Define the literal in the first clause that doesn’t include l_x and ¬l_x. And then for the rest of literal, define true. Because I defined at least one clause to be true, we don’t care about the rest.

3. Prove Validation

#	input	output	#	input	output	#	input	output	#	input	output	#	input	output
1	(or (and a1 (not a1) a3) (and a2 a5 (not a2)) (and a7 a6 (not a6)))	1 -1 3 2 5 -2 7 6 -6 0 UNSAT	**2**	(or (and a1 a2 a3) (and a4 a5 a6) (and a6 a7 a8))	1 2 3 4 5 6 6 7 8 0 1 2 3 4 5 6 7 8	**3**	(or a1 (not (or (not (or a2 a3)) a4)))	1 2 -4 3 -4 0 1 2 3 4	**4**	(or (and a1 (not a1) a3) (and a2 (not a2)))	1 -1 3 2 -2 0 UNSAT	**5**	(or (and a11 a12) (and a13 a14) (and a25 a26 a27))	11 12 13 14 25 26 27 0 11 12 13 14 25 26 27

Mostly test cases passed well. UNSAT detection is perfectly correct.

4. Discussion

When initially writing the code, only sequential values from a_1 to a_n were considered. However, after conducting several test cases, there were others where the starting value was other than a_1 and were not composed of sequential values. Then, in order to know the starting value and the last value, the min and max values had to be obtained. Also, because they were not sequential, an array had to be declared separately and saved in the process of parsing the result values. Although the answer is correct, it is a shame that it is thought to be less effective in terms of code efficiency and readability.

I also thought for a moment about the solver of CNF. CNF, unlike DNF, should have a true value for all clauses. In order to do so, at least one of the literals in each clause must be true. In addition, In the case of UNSAT, thought to be more complex than DNF because, unlike DNF, it is necessary to consider the literal value of all clause.