Project of Algorithms on Condition Satisfiability

Kevin Lei

July 22, 2024

1 Main Idea

In the "Condition Satisfiability Problem", we are given the following:

1. A number of n boolean variables $x_1, x_2, \ldots, x_n$, where x_i must be either true or false.

2. A set of P "lead-to" conditions $L = \{T_1, T_2, \ldots, T_P\}$, where each T_i has $k_i \geq 0$ boolean variables to the left of its $\Rightarrow$ symbol and always one boolean variable to the right. A "lead-to" condition takes the form of $(x_{i_1} \wedge x_{i_2} \wedge \ldots \wedge x_{i_k}) \Rightarrow x_j$. The $k_i + 1$ variables for each $T_i \in L$ are unique, and the degenerate case with $k = 0$ simply means that x_j is true.

3. A set of Q "false-must-exist" conditions $F = \{M_1, M_2, \ldots, M_Q\}$, where each M_i is a cumulative logical OR of $m_i > 0$ negated boolean variables. A "false-must-exist" condition takes the form of $(\neg x_{i_1} \vee \neg x_{i_2} \vee \ldots \vee \neg x_m)$. The m_i variables for each $M_i \in F$ are unique.

We want to find the truth values of $x_1, x_2, \ldots, x_n$ such that the P conditions in L and the Q conditions in F will all evaluate to true. Such an assignment of truth values is called a "satisfying solution". If there is no such assignment, then we want to output "*No satisfying solution exists*".

The main idea behind this algorithm is to initialize a set of n boolean variables to be all false. Then, we iterate through all of the conditions until no more changes are being made. We check the "lead-to" conditions first, and if any has a true left-hand side and a false right-hand side, then we set the variable on the right-hand side to true. In each iteration, after we check all of the "lead-to" conditions, we check if any of the "false-must-exist" conditions are broken. If any of them are broken, we can early return "*No satisfying solution exists*", since there must be a contradiction in the conditions. Once no more changes can be made according to our rules, we output the truth values of $x_1, x_2, \ldots, x_n$.

2 Pseudocode

Algorithm 1: Condition Satisfiability

Input: n boolean variables, a set $L = \{T_1, T_2, \ldots, T_P\}$ of P "lead-to" conditions, and a set $F = \{M_1, M_2, \ldots, M_Q\}$ of Q "false-must-exist" conditions.
Output: A satisfying solution or "No satisfying solution exists".

```
x_1, x_2, ..., x_n = false;
changed = true;
while changed do
    changed = false;
    for each T_i ∈ L do
        if (x_{i_1} ∧ x_{i_2} ∧ ... ∧ x_{i_k}) ∧ ¬x_j then
            x_j = true;
            changed = true;
        end
    end
    for each M_i ∈ F do
        if ¬M_i then
            return "No satisfying solution exists";
        end
    end
end
return x_1, x_2, ..., x_n;
```

3 Proof of Correctness

Initially, the algorithm will set all of the variables $x_1, x_2, \ldots, x_n$ to false. In the main while loop, we enforce the conditions in L by setting the right-hand side of any $T_i \in L$ to true if and only if the entire left-hand side is true and the right-hand side is false. When this happens, the right hand side *must* be true, or else the condition would be broken. Thus, we only change the variables in L if they need to be changed. After we have iterated through all of the conditions in L, we check the conditions in F. If we break any of the conditions in F in an attempt to satisfy the conditions in L, then we early return "*No satisfying solution exists*". This is because all of the conditions in F will initially be true, and if we break any of them, then it is broken out of necessity to satisfy some condition in L. This loop will run until no more changes can be made, i.e. all of the conditions in L and F are satisfied. Thus, the algorithm will output a satisfying solution if one exists, and "*No satisfying solution exists*" otherwise. The algorithm is correct.

4 Runtime Complexity Analysis

Initializing the n boolean variables takes $O(n)$ time. The main while loop will run until no more changes can be made, so the worst case is that the loop runs n times. Inside the main while loop, the first for loop will run up to P times per main loop iteration, and the second for loop will run up to Q times per main loop iteration. The first for loop will need to evaluate k_i logical ANDs for each $T_i \in L$, and the second for loop will need to evaluate m_i logical ORs for each $M_i \in F$. We shall call $k_{\max} = \max\{k_1, k_2, \ldots, k_P\}$ and $m_{\max} = \max\{m_1, m_2, \ldots, m_Q\}$. Thus, the runtime complexity of the algorithm is $O(n(P \cdot k_{\max} + Q \cdot m_{\max}))$.