**2D Project 50.004 Component – Group 25**

Ang Yu Jie -- Jyotit Kaushal -- Wang Shiyu

Nguyen Thai Huy -- Li Xueqing -- Fu Meihui

1. **Problem Statement**

A 2-satisfiability (2-SAT) is a computational problem of finding a solution for a set of Boolean variables that are configured in conjunctive normal form (CNF) for clauses that have a maximum of two literals. A CNF cluster is a product of sums, in other words an AND of ORs, for example:

The problem is satisfied (i.e. there is at least one solution to it so that the formula returns True) if every clause returns true, which requires at least one literal to be true.

1. **Proposed Solution**

The 2-SAT problem can be solved in polynomial time. Thus, our proposed solution to this problem is to use an algorithm based on analyzing the strongly connected components of the implication graph by using depth-first-search (DFS).

* **Implication Graph**

We notice that each clause in 2-SAT problem is logically equivalent to an implication from one literal to the other. For example, the second clause in the above example can be written as:

From this observation, we can write the 2-SAT problem in **implicative normal form**, in which we replace each *OR* clause in the conjunctive normal form by the two implications to which it is equivalent. We can then express this as an implication graph, where each literal is a vertex and each implication is a directed edge. Graph initialization will have time complexity of O(n).

From this, we can check for strongly connected components (SCC), where two components are said to be strongly connected if there is a path from one component to the other and vice versa. We note that for any vertex x, if x and are in the same SCC, the problem has no solution. For example, if x is assigned True, the implication tells us that is also True, which results in a contradiction. We can conclude that a 2-SAT problem has a solution if there is no contradiction in any of its SCC.

* **Value Assignment**

We now construct the algorithm to find the solution of the 2-SAT problem on the assumption that the solution exists. We do this by choosing the value (True or False) that would not generate a contradiction.

Firstly, once we initialize the implication graph containing all literals xi and their negations, we can apply Kosaraju’s algorithm to find all the SCCs. We then proceed to sort them in topological order using topological sort. This is to order every directed edge such that xi always appear before yi.

From here, we use DFS to traverse the ordered graph, which will have time complexity of O(m+n), where m is the number of vertices and n the number of edges. From here, we have two cases for each vertex xi:

* If appears before , it means , and must be True for the expression to be True
* If appears before , it means , and must be False for the expression to be True
* **Determining Satisfiability**

To determine the satisfiability of the 2-SAT problem, after SCC grouping, we check whether a contradiction exists within each SCC group, which is by checking whether and exists in the same group. In case there is, we will terminate the program at this stage and return the “UNSATISFIABLE” message. Else, we will proceed with the value assignment to find a solution and then return that along with “SATISFIABLE” message.

The satisfiability algorithm takes time to compute.

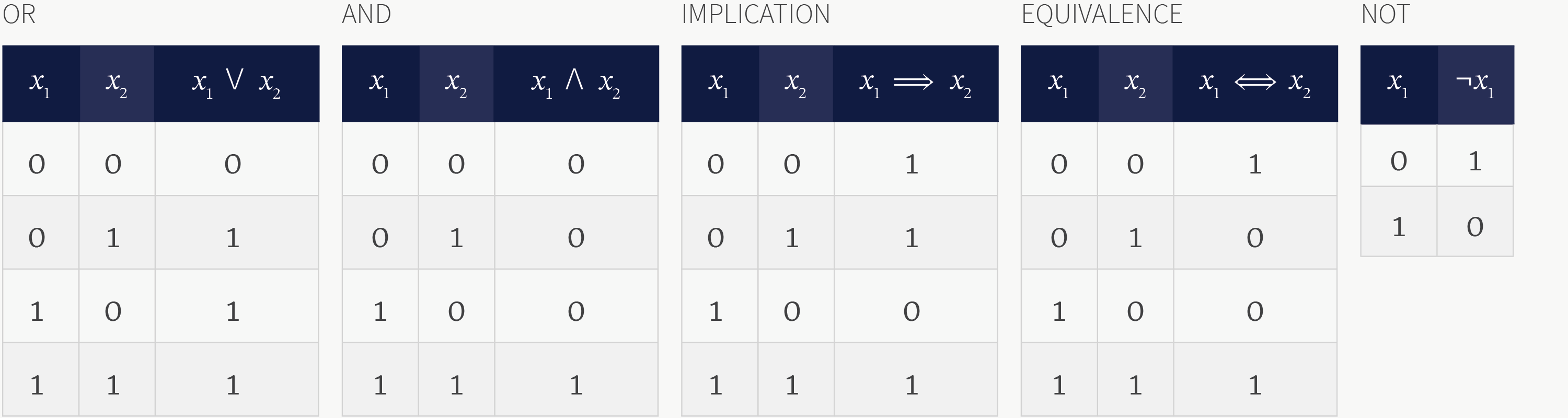
1. **Performance Analysis**

For the running time of our code, half of the time is used to check for contradiction within each SCC. Thus, we can potentially cut the running time by half if we know beforehand the satisfiability of the problem.

Furthermore, our algorithm can only work for 2-SAT problem because of the implication graph. To be more precise, our solution requires the use of implication graph and its strongly connected components to deduce that the problem has no solution if x and are SCC. This conclusion, however, only holds true if there are at most 2 literals in a clause. Should there be more than 2 literals, the observation we make is invalid and the implication graph would also be more complex.

1. **Annex**

**Truth Table of Boolean Operators**



The IMPLICATION operator evaluates whether the two inputs are consistent with the statement 'if x1 then x2'. The statement is only disobeyed when x1 is true and x2 is false and so implication returns false for this combination of inputs and true otherwise.

**2-SAT Problem**

In our problem statement, the 2-SAT problem can have at most 2 literals for each clause. While this is different from the Wikipedia definition, which states that they must have exactly 2 literals, both versions can be considered equivalent, in the sense that they can both be solved in polynomial time complexity.

As shown above, our algorithm can solve the “at most” version in polynomial time. Thus, for the “exactly” version, the process would be exactly the same, thus also taking polynomial time.

1. **Reference**

<https://en.wikipedia.org/wiki/2-satisfiability>

<https://en.wikipedia.org/wiki/Kosaraju%27s_algorithm>

<https://en.wikipedia.org/wiki/Strongly_connected_component>

<https://codeforces.com/blog/entry/16205>

<https://realpython.com/command-line-interfaces-python-argparse/>