2D Challenge 2017
2-SAT algorithm implementation
Group 3-2

Introduction

Our 2-SAT algorithm implementation uses the notion of strongly connected components from graph theory and Tarjan's algorithm.

Implication Graph

We first constructed an implication (directed) graph for the formula. Each literal in the formula is represented as a vertex, i.e. two for each variable, V(A) and V(!A). As a CNF clause is (A or B), the only way it can be satisfied is that if one of its literal is FALSE, the other one has to be TRUE. The clause can therefore be rewritten as the pair (not A --> B) and (not B --> A). This restriction corresponds to two edges in the implication graph for each clause, E(!A, B) and E(!B, A). For a unit clause, V(A) and E(!A --> A).

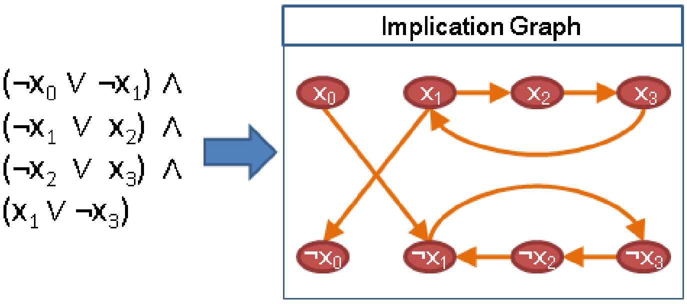

Figure 1: An implication graph of a 2-SAT CNF formula.

From the implication graph, it can be seen that to satisfy all clauses, if a literal is TRUE, all the outgoing edges from its corresponding vertex enforce the adjacent literals to also be TRUE. Also, there must not be any problematic variables where there is a path from it to its negation and vice versa.

Strongly Connected Components

Next, we proceeded to partition the implication graph into strongly-connected components using Tarjan's algorithm. A depth first search is first conducted on an arbitrary node and adds nodes to a stack in the order in which they are visited. Each node is assigned an index and lowlink integer value, where index represents the order it was discovered and lowlink represents the smallest index of all its reachable nodes.

Tarjans' algorithm exhibits stack invariance, where a visited node remains on the stack if and only if it has a path in the graph to an earlier node on the stack. This means that at the end of a call that visits a node, if the node has a path to an earlier node, the call returns but the node is left on the stack. Otherwise, the node is the root of a strongly connected component and itself and all the later nodes on the stack that are part of the same path are popped and returned as a connected component.

During the recursive call of DFS, if the lowlink of a node is smaller than its index, then that node is down a current path and will not popped. Only when the lowlink is equivalent to the index will a node be popped as part of the root of a strongly connected component, as will the later nodes on the stack as part of the connected component.

Once subsequent DFSs are conducted to remaining unvisited nodes and all nodes are visited exactly once, all the strongly connected components in the implication graph would have been recovered. Tarjan's algorithm thus produces a DFS forest of strongly connected component subtrees where each subtree root corresponds to the root node with lowlink == index.

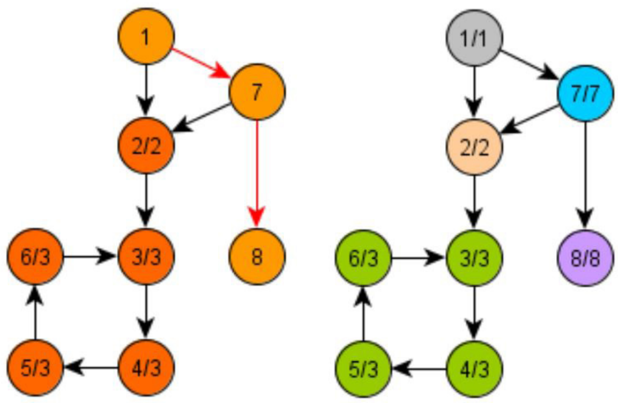

Figure 2: Depth first search explores all nodes and assigns (index/lowlink) values to each node in the order they are visited.
Figure 3: A forest of strongly connected component subtrees found by Tarjan's algorithm.

Checking Satisfiability

Having found the strongly connected components, to determine satisfiability, simply check if a variable and its negation exist in the same strongly connected component.

By the transitivity lemma which states that if all edges are satisfied by an assignment and there is a path from A to B, then it cannot be the case that the truth value of A!=B, all literals in the same SCC must have the same truth value.

This means that a formula is unsatisfiable if a variable and its negation both exists in a SCC as it is logically impossible for them to be either both TRUE or FALSE at the same time. Since a formula is unsatisfiable if and only the above situation occurs, we can conclude that the formula is satisfiable otherwise.

Solving

To find a satisfiable assignment, we first topologically sort the components in reverse order. Since we used Tarjan's algorithm, this has already been done for us.

Next we assign TRUE to the first component that does not already have an assignment. By the transitivity lemma, all the literals in that component would be assigned the same truth value, and the literals in the complement component would be set to FALSE. This is done for all the components later in the reverse topological order and the result would be an assignment that satisfies the CNF formula.

Analysis

Given a CNF formula with n variables and m clauses, where the parameters n and m represent its size, the corresponding implication graph would have 2n vertices and 2m edges.

The time complexity to obtain the strongly connected components would then be $O(n + m)$ as depth first search is done once for each node and each edge is visited at most once by Tarjan's algorithm.

This means that it takes polynomial (or rather linear) time to check for satisfiability.