CIS 1210—Data Structures and Algorithms—Spring 2025

Deterministic 2-SAT—Spring 2025

2-SAT Problem Statement

Definition: A literal is boolean variable α it in generic form (ie not specifically true or false). A variable is a literal either in its true α or false $\overline{\alpha}$ form.

Definition: A clause in 2-CNF is at most 2 variables separated by an "or" ($\vee$) operator.

Definition: A boolean formula ϕ is in 2-CNF if it is one or more clauses separated by an "and" ($\wedge$) operator.

For example: $(x_1 \vee \overline{x}_2) \wedge (x_1 \vee \overline{x}_3) \wedge (x_2 \vee x_3)$ is in 2-CNF but $(x_1 \vee \overline{x}_2 \vee x_3)$ or $\neg(x_1 \vee x_2)$ is not.

Now for the formal problem statement.

Input: A boolean formula ϕ in 2-CNF with n literals and m clauses.

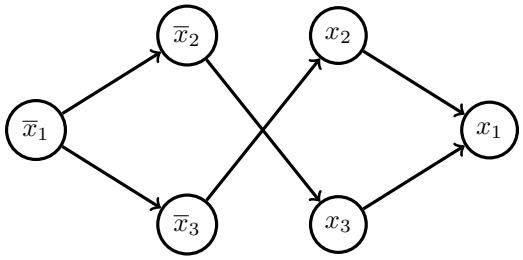
Goal: Determine if there exists an assignment to literals to make ϕ evaluate to true.

A Deterministic Algorithm

To solve the problem we will reduce it to a graph problem. This means we are going to take our input ϕ and represent it as an equivalent graph that encodes the same information as ϕ, and solve a problem related to that graph instead. For notation let ϕ contain the n literals $x_1, x_2, \ldots, x_n$.

Let $G = (V, E)$ be the implication graph of a 2-CNF formula ϕ. G is a directed graph containing $2n$ vertices where each vertex $v_{x_1}, v_{\overline{x}_1}, v_{x_2}, v_{\overline{x}_2}, \ldots v_{x_n}, v_{\overline{x}_n}$ represents a corresponding variable in ϕ. Then for a clause $(\alpha \vee \beta)$ where α and β can be either x_i or $\overline{x}_i$ we add the edges $(v_{\overline{\alpha}}, v_\beta)$ and $(v_{\overline{\beta}}, v_\alpha)$. You can think of an edge from $\overline{\alpha}$ to β in the implication graph as meaning "if $\overline{\alpha}$ is true then β must also be true in a satisfying assignment to ϕ", hence why it is called the implication graph.

For example, for the following formula $(x_1 \vee \overline{x}_2) \wedge (x_1 \vee \overline{x}_3) \wedge (x_2 \vee x_3)$, this is our implication graph.



Note: For the rest of this proof we will use the vertex label v_{x_i} or v_α interchangeably with the variables x_i or α.

Theorem 1. If there exists an x_i such that there is a path from x_i to $\overline{x}_i$ and a path from $\overline{x}_i$ to x_i in the implication graph of ϕ, then ϕ is not satisfiable.

Proof. Assume for contradiction that $\overline{x}_i$ to x_i and x_i to $\overline{x}_i$ paths exist yet ϕ is satisfiable. Let σ be a satisfying assignment to ϕ.

Case 1: x_i = true in σ: Let the corresponding variables along the path from x_i to $\overline{x}_i$ be $p_1, p_2, \ldots p_k$ where $p_1 = x_i$ and $p_k = \overline{x}_i$. Since x_i = true, $\overline{x}_i$ must be false. Thus there must exist some edge on this path (p_j, p_{j+1}) where p_j is true and p_{j+1} is false in σ. Since $(p_j, p_{j+1}) \in E$ then $(\overline{p_j} \vee p_{j+1}) \in \phi$. However this clause evaluates to false in σ a contradiction that σ is a satisfying assignment for ϕ.

Case 2: x_i = false in σ: This case follows by the same logic as case 1 but we consider the path from $\overline{x}_i$ to x_i instead. We leave this an exercise for the reader.

To show the converse we will provide algorithm such that given the implication graph of ϕ we will construct an assignment σ such that ϕ is satisfied.

```
DetermineSatisfyingAssignment(G)
   Run Kosarajus on G to get G^SCC = (V^SCC, E^SCC)
   Compute the topological ordering of G^SCC
   Let σ be an empty assignment

   for each v in V^SCC in reverse topological order
        for each variable α corresponding to vertices in v
            if ᾱ has not been assigned, assign α to be true in σ
                and ᾱ to be false in σ

    return σ
```

In english we create the implication graph G. Then we create G^{SCC} and topologically sort it. Then in reverse topological order we assign all the variables in a sink SCC to be true, and their opposite variables to be false, which we will show later must be in a source SCC. We repeat this process through the graph. By assumption α and $\overline{\alpha}$ will not be in the same SCC so this will assign all literals to either true or false.

To prove the correctness of this algorithm we first need to show the following lemmas.

Lemma 1. For any two variables α and β if there is a path from α to β then there is a path from $\overline{\beta}$ to $\overline{\alpha}$.

Proof. Consider the path from α to β, and let the vertices on the path be $p_1, p_2, \ldots, p_k$ where $p_1 = \alpha$ and $p_k = \beta$. For an arbitrary edge along this path (p_i, p_{i+1}) we added this edge to G since the clause $(\overline{p}_i \vee p_{i+1}) \in \phi$. Since that clause is in ϕ we also know that the edge $(\overline{p}_{i+1}, \overline{p}_i)$ is also in G.

Thus for the edges

$$(p_1, p_2), (p_2, p_3), \ldots (p_{k-1}, p_k)$$

there are corresponding edges

$$(\overline{p}_k, \overline{p}_{k-1}), (\overline{p}_{k-1}, \overline{p}_{k-2}), \ldots, (\overline{p}_2, \overline{p}_1)$$

Thus a path from $\overline{\beta}$ to $\overline{\alpha}$. □

Lemma 2. If α and β are in the same SCC then so are $\overline{\alpha}$ and $\overline{\beta}$.

Proof. If α and β are in the same SCC then there is a path from α to β and one from β to α. Thus by Lemma 1 there are also paths from $\overline{\alpha}$ to $\overline{\beta}$ and $\overline{\beta}$ to $\overline{\alpha}$. Thus $\overline{\beta}$ and $\overline{\alpha}$ are in the same SCC. □

Lemma 3. If α and β are in a sink SCC then $\overline{\alpha}$ and $\overline{\beta}$ are in a source SCC.

Proof. We know that if α and β are in the same SCC call it C then by Lemma 2, $\overline{\alpha}$ and $\overline{\beta}$ are in the same SCC call it $\overline{C}$. It remains to show that if C is a sink then $\overline{C}$ is a source.

Assume towards contradiction that $\overline{C}$ is not a source. This implies there are two vertices $\overline{\gamma} \notin \overline{C}$ and $\overline{u} \in \overline{C}$ such that $(\overline{\gamma}, \overline{u}) \in E$. By the same logic as Theorem 1, we know $(u, \gamma) \in E$. We also know that by Lemma 2. that u and γ can not be in the same SCC and that $u \in C$. Thus C has an outgoing edge to γ making C not a sink. Contradiction! □

With the above lemmas we have what we can return to the converse of Theorem 1 but in a new form.

Theorem 2. If ϕ is satisfiable our algorithm produces a valid assignment σ that evaluates ϕ to be true.

Proof. First note that our assignment σ is valid since for each literal it either assigns it true or false (ie. it never assigns x_i and $\overline{x}_i$ to both be true). It remains to show that ϕ is satisfied by σ.

Assume towards contradiction that ϕ is satisfiable but σ results in ϕ being false. This implies that there exists some clause $(\alpha \vee \beta)$ such that $\alpha =$ false and $\beta =$ false in σ. In the execution of our algorithm W.L.O.G assume that α was assigned to false first.

Since $(\alpha \vee \beta) \in \phi$, we know that $(\overline{\alpha}, \beta) \in E$ and $(\overline{\beta}, \alpha) \in E$. Let us consider the instance in the algorithm that α was assigned false. This was caused by the variable $\overline{\alpha} =$ true.

Case 1: $\overline{\alpha}$ and $\overline{\beta}$ are in the same SCC: By Lemma 2 We know that β and α are in the same SCC. We also know that the edges $(\overline{\alpha}, \beta)$ and $(\overline{\beta}, \alpha)$ exist. Thus either α and $\overline{\alpha}$ are in the same SCC which by Theorem 1 ϕ is not satisfiable a contradiction! Or α and $\overline{\alpha}$ are in different SCCs. However since there is a path from $\overline{\alpha}$ to β and a path from β to α, this implies that α's SCC must come later in the topologcal ordering of G^{SCC}. Thus we would have assigned $\alpha =$ true first a contradiction!

Case 2: $\overline{\alpha}$ and $\overline{\beta}$ are in different SCCs: By Lemma 2 we know that α and β are in different SCCs as well. Let us consider the order that the SCCs can fall in the topological ordering. Since we assumed $\overline{\alpha}$ was assigned before $\overline{\beta}$ this implies that $\overline{\alpha}$ comes after $\overline{\beta}$ in the topological ordering. Since the edges $(\overline{\alpha}, \beta)$ and $(\overline{\beta}, \alpha)$ that only leaves with the following three orderings:

$$\overline{\beta}, \overline{\alpha}, \alpha, \beta \qquad \text{or} \qquad \overline{\beta}, \overline{\alpha}, \beta, \alpha \qquad \text{or} \qquad \overline{\beta}, \alpha, \overline{\alpha}, \beta$$

However in the first two cases α appears after $\overline{\alpha}$ in the topological ordering. Thus α would have been assigned to true before $\overline{\alpha}$ was assigned to true, a contradiction. In the third case β would have been assigned to true, also a contradiction. □

Implementation and Running Time

To implement the entire algorithm we first need to create the implication graph, and run Kosaraju's algorithm on it. Graph has $2n$ vertices and since each clause contributes 2 edges, $O(m)$ edges. Thus the overall runtime of graph construction and Kosaraju's is $O(n+m)$.

Then we need to check if for each x_i if x_i and $\overline{x}_i$ appear in the same SCC. Let us assume our x_i are mapped to the range $[1, 2n]$ where $x_i \to 2i$ and $\overline{x}_i \to 2i+1$ and appear in that order in the adjacency list. Thus in if we build an array $S[1..2n]$ where $S[i]$ is the SCC of vertex i, we just need to ensure that $S[i] \neq S[i+1]$ for $i = 1, 3, 5, \ldots, 2n-1$. Constructing the array involves traversing the DFS forest in $O(n)$ time and iterating through S in $O(n)$ time.

Lastly we run `DetermineSatisfyingAssignment` which involved running an additional toposort in $O(n+m)$ time, and iterating through all and all literals of each SCC. Note that we can store our assignment σ in an array $\sigma[1..n]$ where $\sigma[i]$ =true if x_i = true. We can also do the same for a "has been assigned" array. Thus each literal can be assigned and checked if has already been assigned in $O(1)$ time, leading to an overall runtime of $O(n+m)$.