The 2SAT Problem

A formula of propositional logics is in *2CNF* if it has the form

$$C_1 \wedge \cdots \wedge C_n$$

where the C_i are *clauses* of size 2. A clause of size two has the form $(A \vee B)$ where A and B are *literals*. A literal has the form x or $\neg x$ where x is a *variable* of propositional calculus that can take the values true or false. CNF is a shorthand for *conjunctive normal form*. A formula of propositional calculus is satisfiable if there is an assignment of truth values to its variables so that the formula evaluates to true using the usual semantics of propositional calculus.

The 2SAT problem is to decide whether a given 2CNF formula F is satisfiable. Furthermore, if F is satisfiable, we want an assignment of truth values to its variables proving satisfiability. If F is not satisfiable, we would like to output a proof of unsatisfiability.

We will now prove the following theorem.

Theorem 1 *The 2SAT problem for a formula F with n clauses can be solved in time $\mathcal{O}(n)$.*

We transform an instance of 2SAT into a graph theoretic problem as follows. We define a directed graph $G = (V, E)$. The nodes V are the variables appearing in F, i.e., $V = \bigcup_{x \in F} \{x, \neg x\}$. Let us introduce the notation $\overline{L}$ for the negation of a literal so that $\overline{x} = \neg x$ and $\overline{\neg x} = x$. For every clause $(A \vee B)$ in F we introduce edges $(\overline{A}, B)$ and $(\overline{B}, A)$ in E. Note that a clause $(A \vee B)$ is equivalent to the implications $\overline{A} \rightarrow B$ and $\overline{B} \rightarrow A$. Hence, the edge in our graph corresponds to implications. In particular, if a literal L is true and there is an edge $\overline{L} \rightarrow L'$ in the graph then L' must also be true for F to be true.

Now consider the strongly connected components of G. Let $f(v)$ denote the component number of v where the compenent numbers are picked in such a way that they topologically sort the component graph. We have

$$\forall v, w \in V : v \overset{*}{\rightarrow} w \Rightarrow f(v) \leq f(w).$$

Lemma 1 *If there is a variable x such that $f(x) = f(\neg x)$ then F is unsatisfiable.*

Proof: Since x and $\neg x$ have the same component number they are in the same strongly connected component and hence, there is a cycle $\langle x, \ldots, \neg x, \ldots, x \rangle$. Assume that there is a satisfying assignment with $x =$true. Then all the literals on the path $\langle x, \ldots, \neg x \rangle$ must be true. But $\neg x$ is false and hence we have a contradiction. The assumption $x =$false implies $\neg x =$true which leads to a contradiction in an analogous way. Hence, there is no satisfying assignment. $\square$

If there is no cycle containing a variable and its complement we can construct a satisfying truth assignment as follows.

Lemma 2 *If $\forall x \in V : f(x) \neq f(\neg x)$ then we get a satisfying assignment by setting $x =$true if $f(x) > f(\neg x)$ and $x =$false if $f(x) < f(\neg x)$.*

PROOF: Assume to the contrary that F gets the value false using our assignment. Then there must be a clause $(A \vee B)$ in F that evaluates to false, i.e., both the literals A and B evaluate to false. By definition of our truth values we have $f(A) < f(\overline{A})$ and $f(B) < f(\overline{B})$. The clause $(A \vee B)$ contributes the edges $(\overline{A}, B)$ and $(\overline{B}, A)$ to E and hence by definition of f, we have $f(\overline{B}) \leq f(A)$ and $f(\overline{A}) \leq f(B)$. Combining these four relationships we get $f(A) < f(\overline{A}) \leq f(B) < f(\overline{B}) \leq f(A)$, i.e., $f(A) < f(A)$ — a contradiction. $\Box$

To complete the proof of the main theorem, we have to analyze the run time of our algorithm. We can construct an adjacency list representation of G using a single pass through F in time $\mathcal{O}(n)$. The graph has $2n$ edges and at most $4n$ nodes. Additionally, we build a list L of all variables x with pointers to the two the graph nodes x and $\neg x$. Strongly connected components can be computed in time $\mathcal{O}(n)$ using several algorithms. If we use the single pass algorithms explained in the lecture, we have the additional benefit that the component numbers are already in (reverse) topological order, i.e., we can set $f(v) = -\text{compNum}[\text{v}]$ and do not need an additional topological sorting phase. The truth assignment can then be computed using a single pass through the list of variables L. If we encounter a variable x with $f(x) = f(\neg x)$, we can use the strongly connected component of x and $\neg x$ to extract a set of clauses that correspond to a cycle in G and prove unsatisfiability.[1] $\blacksquare$

[1]Exercise: Explain in more detail how such a cycle can be constructed.