Satisfiability of 2−CNF Formulae in Propositional Logic

February 18, 2007

Let ϕ be a 2−CNF formula in propositional logic, i.e. ϕ is a conjunction of clauses, with each clause having at most two literals. We wish to determine if ϕ is satisfiable.

Clauses with exactly one literal represent *unit literals* and are easy to deal with. As discussed in class, we must set all such literals to True in any satisfying assignment of ϕ. So we assign True to all unit literals and simplify the formula ϕ. If the simplified formula has new unit literals, we assign True to them and continue until all clauses in the final simplified formula have exactly two literals. It is not hard to see that the above process must terminate in $O(n.m)$ time, where n and m are the no. of propositions and no. of clauses, respectively in the original 2−CNF formula ϕ.

In the following discussion, we will assume that the above simplification has been done, and so ϕ has exactly two literals per clause.

In class, we discussed that each two-literal clause can be viewed as an implication, since $(l_i \vee l_j)$ is semantically equivalent to $(\neg l_i \rightarrow l_j)$ as well as to $(\neg l_j \rightarrow l_i)$. Thus, we can view our formula ϕ as a conjunction of implications, where two implications are obtained from each clause in the original 2−CNF formula.

We can now build an implication graph G_ϕ for Boolean Constraint Propagation (or BCP), as discussed in class. Essentially, we build a graph, where there is a node for each literal (i.e. each proposition and its negation). For each clause $(l_i \vee l_j)$ in ϕ, we add two edges in this graph: one from the node representing $\neg l_i$ to the one representing l_j, and the other from the node representing $\neg l_j$ to the one representing l_i. Note that it is important that **both** these edges be drawn for each clause.

The graph G_ϕ may have cycles in general. If there is a cycle C such that for some literal l_k, the nodes corresponding to both l_k and $\neg l_k$ are present in C, we will call C an *inconsistent cycle*. Note also that whenever there is an edge from l_i to l_j in G_ϕ, there must also be an edge from $\neg l_j$ to $\neg l_i$. Thus, if there is a path from l_r to l_s in G_ϕ, then there also exists a path from $\neg l_s$ to $\neg l_r$ in G_ϕ, in which each node corresponds to the negation of the corresponding node on the path from l_r to l_s. We will call this the *reversible negated paths* property.

Theorem: ***A 2−CNF formula ϕ is unsatisfiable if and only if there is an inconsistent cycle in G_ϕ.***

Proof: [If part:] Let there be an inconsistent cycle C in G_ϕ. Let l_C be a literal such that the nodes corresponding to both l_C and $\neg l_C$ are present in C (since C is inconsistent, there must exist at least one such literal). We will now show that ϕ is unsatisfiable using proof by contraditction.

Assume for the time being that ϕ is satisfiable, and let A be a satisfying assignment of ϕ, i.e. an assignment of truth values to all propositions in ϕ that makes ϕ evaluate to True. Since ϕ evaluates to true, each clause in ϕ evaluates to true with assignment A. Thus, the implication corresponding to each edge in the inconsistent cycle C evaluates to True with assignment A. Suppose assignment A sets l_C to True and $\neg l_C$ to False. Since all implications corresponding to edges in C are satisfied by assignment A, and since the node corresponding to l_C is in the cycle C, the literals corresponding to all nodes in C must be set to True by assignment A. However, we know that $\neg l_C$ is in cycle C, and the assignment A sets $\neg l_C$ to False. Hence, we have a contradiction. If assignment A had set l_C to False and $\neg l_C$ to True, we arrive at a contradiction using the same reasoning. Therefore, our assumption must be wrong, i.e., ϕ is not satisfiable, and there doesn't exist any satisfying assignment A of ϕ.

[Only if part:] In this part, we will show that if G_ϕ has no inconsistent cycles, then we can use the following algorithm to derive a satisfying assignment of ϕ from G_ϕ.

GetSatisfyingAssignment(G_ϕ)
    Let X = Set of literals l_i such that there is a path from $\neg l_i$ to l_i in G_ϕ

    Insert new nodes $\top$ and $\perp$ in G_ϕ;
    Add an edge from $\top$ to every node in X;
    Add an edge from every node in X to $\perp$;
    TrueNode := $\top$;
    while (TrueNode != $\perp$)
        Set all literals reachable (in G_ϕ) from TrueNode to True;
        Set negations of above literals to False;
        if (all literals not assigned truth value)
            TrueNode := randomly chosen unassigned literal;
        else
            TrueNode := $\perp$;

It is not hard to see that when the **while** loop terminates, all literals are assigned truth values. We now claim that as the algorithm proceeds, it is *never* the case that the graph G_ϕ has an edge from a literal assigned True to a literal assigned False. Thus, the assignment of truth values to literals given by the above algorithm satisfies all implications represented by the edges of G_ϕ, and hence satisfies the formula ϕ.

In the above algorithm, if a literal $l \in X$, then there exists a path from $\neg l$ to l in G_ϕ. This means that $\neg l \rightarrow l$ must be satisfied in any satisfying assignment of ϕ. This is possible only if l is set to True in the satisfying assignment. Thus, all literals in X must be assigned True and their negations assigned False in any satisfying assignment of ϕ. This fact is captured by introducing the implications $(\top \rightarrow l)$ and $(\neg l \rightarrow \perp)$ for each l in X. Thus, in the graph, we introduce the nodes $\top$ and $\perp$ (representing the logical formulae $\top$ and $\perp$ respectively), and insert edges from $\top$ to every $l \in X$, and from every $l \in X$ to $\perp$. Note that if we regard the node $\perp$ as $\neg\top$, the reversible negated paths property of G_ϕ is preserved even after adding these edges.

The variable TrueNode represents a literal (or the formula $\top$ in the very first iteration of the **while** loop) that can be assigned True to get a satisfying assignment of ϕ.

The step that sets all literals reachable from TrueNode to True essentially captures the fact that for an implication to be satisfied, whenever the antecedent is True, the consequent must be True as well.

So how do we show that there is never an edge from a literal assigned True to a literal assigned False? We show this by contradiction.

As the algorithm executes, suppose an edge of the above type appears *for the first time* when TrueNode is some literal l_i. Let this edge be from literal l_j to l_k, where l_j has been set to True and l_k to False. From the reversible negated paths property of G_ϕ, there must also be an edge from $\neg l_k$ to $\neg l_j$ as well, where $\neg l_k$ is set to True and $\neg l_j$ to False.

It is clear from the above algorithm that the only way that a literal can be set to True is if there is a path to the literal from l_i (the current TrueNode) or from a literal (or the formula $\top$) that was TrueNode prior to l_i. Since this is the first time that an edge from True to False appears in the graph, there must be a path from l_i (current TrueNode) to at least one of l_j and $\neg l_k$. Without loss of generality, suppose there is a path from l_i to l_j, and a path from l_h to $\neg l_k$, where l_h is either l_i or some literal (or $\top$) that was TrueNode prior to l_i (this is the only way that a literal can be assigned True by the above algorithm). By the reversible negated paths property of G_ϕ, we can now infer that there exists a path from $\neg l_j$ to $\neg l_i$, and a path from l_k to $\neg l_h$. It follows immediately that G_ϕ has a path from l_i to $\neg l_h$ and a path from l_h to $\neg l_i$.

Recall that both l_h and l_i are TrueNodes at some time during the execution of the algorithm. We now consider the different possibilities of l_h and l_i.

- Suppose $l_i \neq \top$. If $l_h = l_i$, then the existence of a path from l_i to $\neg l_h = \neg l_i$ must put $\neg l_i$ in the set X at the start of the algorithm. Therefore, l_i can never become a TrueNode, contradicting our assumption that l_i is the current TrueNode. If $l_h \neq l_i$, then since l_i is the current TrueNode, l_h must have been TrueNode prior to l_i. But then the existence of a path from l_h to $\neg l_i$ would have set $\neg l_i$ to True and hence l_i to False, when considering literals reachable from l_h. In this case too, l_i could not have been the current TrueNode.

- Suppose $l_i = \top$. Since $\top$ is the very first TrueNode in the above algorithm, we must have $l_h = l_i = \top$. The paths from l_i to $\neg l_h$ and from l_h to $\neg l_i$ are therefore paths in G_ϕ from $\top$ to $\perp$. Let the first edge in the path from l_i to $\neg l_h$ be from $\top$ to l_r, and the last edge in this path be from l_s to $\perp$. Therefore, the first edge in the path from l_h to $\neg l_i$ must be from $\top$ to $\neg l_s$, and the last edge in this path must be from $\neg l_r$ to $\perp$. From the way edges are added to and from $\top$ and $\perp$ at the beginning of the algorithm, we can now infer that there must be a path from $\neg l_r$ to l_r and similarly, there must be a path from l_s to $\neg l_s$. However, this gives rise to a path from l_r to l_s, continuing through $\neg l_s$ and $\neg l_r$, back to l_r. This is an inconsistent cycle, since it contains both l_r and $\neg l_r$. This violates our assumption that there are no inconsistent cycles in G_ϕ.

This establishes the validity of the theorem we mentioned earlier. Note that the above algorithm is required only to find a satisfying assignment of ϕ in case there are no inconsistent cycles. However, the satisfiability question can be answered simply by checking for inconsistent cycles.
Given a graph, the set of cycles (strongly connected components) in the graph can be computed in $O(n^2)$ time using Tarjan's algorithm, where n is the no. of propositions (giving rise to $2n$ nodes in the graph). Once these are computed, all we need to do is to check whether any cycle contains both a literal and its complement.