A LINEAR-TIME ALGORITHM FOR TESTING THE TRUTH OF CERTAIN QUANTIFIED BOOLEAN FORMULAS *

Bengt ASPVALL, Michael F. PLASS and Robert Endre TARJAN
Computer Science Department, Stanford University, Stanford, CA 94305, U.S.A.

Received 22 August 1978, revised version received 16 October 1978

Quantified Boolean formula, strongly connected components, 2-CNF, 2-satisfiability

Let $F = Q_1x_1\, Q_2x_2 \cdots Q_nx_n\, C$ be a quantified Boolean formula with no free variables, where each Q_i is either $\exists$ or $\forall$ and C is in conjunctive normal form. That is, C is a conjunction of clauses, each clause is a disjunction of literals, and each literal is either a variable, x_i, or the negation of a variable, $\bar{x}_i$ $(1 \leqslant i \leqslant n)$. We shall use u_i to denote a literal equal to either x_i or $\bar{x}_i$. The *evaluation problem for quantified Boolean formulas* is to determine whether such a formula F is true.

The evaluation problem is complete in polynomial space [6], even if C is restricted to contain at most three literals per clause. The *satisfiability problem*, the special case in which all quantifiers are existential, is NP-complete [1,2,4] for formulas with three literals per clause. However, the satisfiability problem for formulas with only two literals per clause is solvable in polynomial time [1,2,4]; Even, Itai, and Shamir [3] outline a linear-time algorithm. Schaefer [5] claims a polynomial time bound for the evaluation problem with two literals per clause, although he gives no proof. In this note we present a simple constructive algorithm for the evaluation of formulas having two literals per clause, which runs in linear time on a random access machine.

* This work was supported by National Science Foundation Grants MCS-75-22870 and MCS-77-23738, by Office of Naval Research Contract N00014-76-C-0688, by a National Science Foundation Graduate Fellowship, and by a Guggenheim Fellowship. Reproduction in whole or in part is permitted for any purposes of the United States government.

Our algorithm uses properties of directed graphs. Suppose we are given a formula $F = Q_1x_1\, Q_2x_2 \cdots Q_nx_n\, C$ such that C is in conjunctive normal form with at most two literals per clause. We can assume without loss of generality that there are no clauses with only one literal since the clause u is equivalent to the clause $u \vee u$. We construct a directed graph $G(F)$ with $2n$ vertices and $2|C|$ edges (counting multiple edges) as follows:

1(i) For each variable x_i, we add two vertices named x_i and $\bar{x}_i$ to $G(F)$. We identify $\bar{\bar{x}}_i$ with x_i, and we call x_i and $\bar{x}_i$ *complements* of each other.

1(ii) For each clause $(u \vee v)$ of C, we add edges $\bar{u} \to v$ and $\bar{v} \to u$ to $G(F)$.

The graph $G(F)$ has the following *duality property*: $G(F)$ is isomorphic to the graph obtained from $G(F)$ by reversing the directions of all the edges and complementing the names of all the vertices. See Fig. 1.

Our algorithm relies upon identifying the strong components of $G(F)$. A graph is *strongly connected* if there is a path from any vertex to any other. The

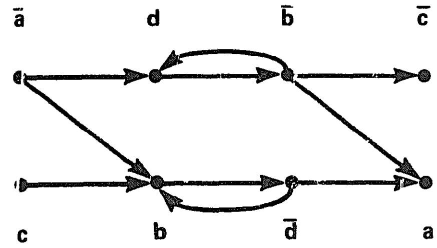

Fig. 1. Graph constructed for the set of clauses $C = \{a \vee b, b \vee \bar{c}, \bar{b} \vee \bar{d}, b \vee d, d \vee a\}$.

maximal strongly connected subgraphs of a graph are vertex-disjoint and are called its *strong components*. If S_1 and S_2 are strong components such that an edge leads from a vertex in S_1 to a vertex in S_2, then S_1 is a *predecessor* of S_2 and S_2 is a *successor* of S_1. The strong components of a directed graph can be found in linear time by an algorithm of Tarjan [1,4,7], which generates the components in reverse topological order; that is, in an order such that if S_1 is generated before S_2, then S_1 is not a predecessor of S_2.

By the duality property every strong component S of $G(F)$ has a dual component $\bar{S}$ consisting of the subgraph induced by the complements of the vertices in S. (It may be the case that $S = \bar{S}$.) If S_1 and S_2 are two strong components of $G(F)$ and S_1 is a predecessor of S_2, then $\bar{S}_1$ is a successor of $\bar{S}_2$ and vice versa. It follows that S_1 and S_2 are incomparable if and only if $\bar{S}_1$ and $\bar{S}_2$ are incomparable.

Suppose we assign truth values to the vertices of $G(F)$. Such an assignment corresponds to a set of truth values for the variables which makes C true if and only if:

2(i) For all i, vertices x_i and $\bar{x}_i$ receive complementary truth values.

2(ii) No edge $u \to v$ has u assigned true and v assigned false (equivalently, no path leads from a true vertex to a false vertex).

To clarify our approach, we first consider the satisfiability problem; thus let us assume that all the quantifiers in F are existential.

Theorem 1. *The expression C is satisfiable if and only if in $G(F)$ no vertex u_i is in the same strong component as its complement $\bar{u}_i$ (i.e., no strong component S is equal to its complement $\bar{S}$).*

Proof. In one direction the proof is easy: if some vertex u_i is in the same strong component as $\bar{u}_i$, then any truth assignment to the vertices of $G(F)$ must violate either 2(i) or 2(ii); hence C is unsatisfiable. To prove the converse we provide an algorithm which either detects the condition in Theorem 1 or marks the strong components of $G(F)$ in a way which implies the existence of a truth assignment satisfying C.

Two-satisfiability algorithm. Process the strong components S of $G(F)$ in reverse topological order as follows:

General Step. If S is marked, do nothing. Otherwise if $S = \bar{S}$ then stop: C is unsatisfiable. Otherwise mark S **true** and $\bar{S}$ **false**.

This algorithm stops prematurely only if some vertex is in the same strong component as its complement. By using the duality property and induction, it is easy to prove every component marked **true** has only true components as successors and every component marked **false** has only **false** components as predecessors. Thus, if the algorithm does not stop prematurely, it marks the components so that complementary components have complementary values and no path leads from a **true** component to a **false** component. If we assign to each vertex the truth value of the component containing it, we get a truth assignment satisfying 2(i) and 2(ii).

We now generalize this idea to solve the evaluation problem. We call a vertex *universal* if the corresponding variable is universally quantified and *existential* otherwise.

Theorem 2. *The formula F is true if and only if none of the following three conditions holds*:

3(i) *An existential vertex u is in the same strong component as its complement $\bar{u}$.*

3(ii) *A universal vertex u_i is in the same strong component as an existential vertex u_j such that $j < i$ (i.e., x_j is not quantified within the scope of Q_i).*

3(iii) *There is a path from a universal vertex u to another universal vertex v. (This condition includes the case that $v = \bar{u}$.)*

Proof. It is easy to show that if any 3(i)–3(iii) holds, then F is false. We prove the converse by giving an algorithm that either detects an occurrence of 3(i), 3(ii), or 3(iii), or else marks the strong components of $G(F)$ in a way that verifies the truth of F.

Two-CNF evaluation algorithm. Process the strong components S of $G(F)$ in reverse topological order as follows:

Step 1. If S is marked then go on to the next component. Otherwise if some successor of S is marked **false** or **contingent** go to *Step 2.* Otherwise go to *Step 3.*

Step 2. (S has a **false** or **contingent** successor.) If S contains one or more universal vertices, stop: condition 3(iii) holds. Otherwise, mark S **false** and go to *Step 5*.

Step 3. (All successors of S are **true**.) If S contains two or more universal vertices, stop: condition 3(iii) holds. Otherwise, if S contains one universal variable u_i, go to *Step 4*. Otherwise, mark S **true** and go to *Step 5*.

Step 4. (S contains a universal vertex u_i.) If S contains an existential vertex u_j with $j < i$, stop: condition 3(ii) holds. Otherwise mark S **contingent** and go to *Step 5*.

Step 5. (S is marked successfully.) If $S = \bar{S}$ stop: condition 3(i) or 3(iii) holds. Otherwise, go to *Step 6*.

Step 6. ($S \neq \bar{S}$.) If S is marked **contingent** or **false** and $\bar{S}$ is a predecessor of S, stop: condition 3(iii) holds. Otherwise, mark $\bar{S}$ **false** if S is **true, contingent** if S is **contingent**, and **true** if S is **false**; go on to the next component.

This algorithm marks each component processed either **true, false,** or **contingent**. Each component containing a universal vertex is marked **contingent**; each component containing only existential variables is marked either **true** or **false**. When a component is marked **false**, either all its successors are marked, at least one of them **contingent** or **false**, or all its predecessors are marked, all of them **false**. This follows immediately from Steps 2, 3, and 6 and the duality property. It follows by induction that if, during the operation of the algorithm, some component S_1 is marked **false** while it has an unmarked predecessor, then there is a path from S_1 to a component S_2 marked **contingent**. Similarly, when a component is marked **true**, either all its successors are marked, all **true**, or all its predecessors are marked, at least one **true** or **contingent**. Thus if some component S_2 is marked **true** while it has an unmarked successor, then there is a path from some **contingent** component S_1 to S_2.

It follows from these facts that if the algorithm stops in Step 2 or Step 6, then condition 3(iii) holds. If the algorithm stops in Step 3, Step 4, or Step 5, it is obvious that the indicated condition holds. Thus if the algorithm stops prematurely, at least one of the conditions 3(i)–3(iii) holds.

If the algorithm does not stop prematurely, every component is marked so that any **true** or **contingent** component has only **true** components as successors, and any **false** or **contingent** component has only **false** components as predecessors. This follows easily from the operation of the algorithm and the duality property. Furthermore every component and its complement receive complementary truth values, and every **contingent** component containing a universal vertex u_i contains as additional vertices only existential vertices u_j such that $i < j$. We can prove that F is true as follows: to each vertex in a **true** or **false** component we assign **true** or **false**, respectively. For any assignment of truth values to the universal variables, we assign to each vertex in a component containing a universal vertex u_i the truth value of x_i if $u_i = x_i$ and the complementary truth value if $u_i = \bar{x}_i$. Such an assignment satisfies 2(i) and 2(ii). Thus F is true.

Our algorithm requires $O(n + m)$ time, where m is the number of edges in $G(F)$ (twice the number of clauses in C). The algorithm processes the strong components in the same order as they are generated by the linear-time strong components algorithm; thus the strong components algorithm may be used with only minor modifications to solve this evaluation problem.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms (Addison-Wesley, Reading, MA, 1974).

[2] S.A. Cook, The complexity of theorem proving procedures, Proc. 3rd Ann. ACM Symp. Theory Comput. (1971) 151–158.

[3] S. Even, A. Itai and A. Shamir, On the complexity of timetable and multi-commodity flow problems, SIAM J. Comput. 5 (4) (1976) 691–703.

[4] E.M. Reingold, J. Nievergelt and N. Deo, Combinatorial Algorithms: Theory and Practice (Prentice-Hall, Englewood Cliffs, NJ, 1977).

[5] T.J. Schaefer, The complexity of satisfiability problems, Proc. 10th Ann. ACM Symp. Theory Comput. (1978) 216–226.

[6] L.J. Stockmeyer and A.R. Meyer, Word problems requiring exponential time, Proc. 5th Ann. ACM Symp. Theory Comput. (1973) 1–9.

[7] R.E. Tarjan, Depth first search and linear graph algorithms, SIAM J. Comput. 1 (2) (1972) 146–160.