The Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

Throughout this paper, a *set of strings* means a set of strings on some fixed, large, finite alphabet Σ. This alphabet is large enough to include symbols for all sets described here. All Turing machines are deterministic recognition devices, unless the contrary is explicitly stated.

1. *Tautologies and Polynomial Re-Reducibility*.

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ. Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary notation to distinguish that symbol. Thus a formula of length n can only have about $n/\log n$ distinct function and predicate symbols. The logical connectives are & (and), $\vee$ (or), and $\neg$(not).

The set of tautologies (denoted by {tautologies}) is a certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By *reduced* we mean, roughly speaking, *that* if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time. In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles in [1].

A *query machine* is a multitape Turing machine with a distinguished tape called the *query tape*, and three distinguished states called the *query state*, *yes state*, and *no state*, respectively. If M is a query machine and T is a set of strings, then a *T-computation* of M is a computation of M in which initially M is in the initial state and has an input string w on its input tape, and each time M assumes the query state there is a string u on the query tape, and the next state M assumes is the yes state if $u \in T$ and the no state if $u \notin T$. We think of an "oracle", which knows T, placing M in the yes state or no state.

Definition

A set S of strings is *P-reducible* (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial $Q(n)$ such that for each input string w, the T-computation of M with input w halts within $Q(|w|)$ steps ($|w|$ is the length of w), and ends in an accepting state iff $w \in S$.

It is not hard to see that P-reducibility is a transitive relation. Thus the relation E on sets of strings, given by $(S,T)\epsilon E$ iff each of S and T is P-reducible to the other, is an equivalence relation. The equivalence class containing a set S will be denoted by deg (S) (the polynomial degree of difficulty of S).

Definition: We will denote deg ({0}) by $\mathcal{L}_*$, where 0 denotes the zero function.

Thus $\mathcal{L}_*$ is the class of sets recognizable in polynomial time. $\mathcal{L}_*$ was discussed in [2], p. 5, and is the string analog of Cabham's class of functions [3].

We now define the following special sets of strings.

1) The subgraph problem is the problem given two finite undirected graphs, determine whether the first is isomorphic to a subgraph of the second. A graph G can be represented by a string $\bar{G}$ on the alphabet $\{0,1,*\}$ by listing the successive rows of its adjacency matrix, separated by *s. We let {subgraph pairs} denote the set of strings $\bar{G}_1{**}\bar{G}_2$ such that G_1 is isomorphic to a subgraph of G_2.

2) The graph isomorphism problem will be represented by the set, denoted by {isomorphic graphpairs}, of all strings $\bar{G}_1{**}\bar{G}_2$ such that G_1 is isomorphic to G_2.

3) The set {Primes} is the set of all binary notations for prime numbers.

4) The set {DNF tautologies} is the set of strings representing tautologies in disjunctive normal form.

5) The set D_3 consists of those tautologies in disjunctive normal form in which each disjunct has at most three conjuncts (each of which is an atom or negation of an atom).

Theorem 1: If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to {DNF tautologies}.

Corollary: Each of the sets in definitions 1)-5) is P-reducible to {DNF tautologies}.

This is because each set, or its complement, is accepted in polynomial time by some nondeterministic Turing machine.

Proof of the theorem: Suppose a nondeterministic Turing machine M accepts a set S of strings within time $Q(n)$, where $Q(n)$ is a polynomial. Given an input w for M, we will construct a proposition formula $A(w)$ in conjunctive normal form such that $A(w)$ is satisfiable iff M accepts w. Thus $\neg A(w)$ is easily put in disjunctive normal form (using De Morgan's laws), and $\neg A(w)$ is a tautology if and only if $w\not\epsilon S$. Since the whole construction can be carried out in time bounded by a polynomial in $|w|$ (the length of w), the theorem will be proved.

We may as well assume the Turing machine M has only one tape, which is infinite to the right but has a left-most square. Let us number the squares from left to right 1, 2, ---. Let us fix an input w to M of length n, and suppose $w\epsilon S$. Then there is a computation of M with input w that ends in an accepting state within $T = Q(n)$ steps. The formula $A(w)$ will be built from many different proposition symbols, whose intended meanings, listed below, refer to such a computation.

Suppose the tape alphabet for M is $\{\sigma_1, \ldots, \sigma_\ell\}$, and the set of states is $\{q_1, \ldots, q_s\}$. Notice that since the computation has at most $T = Q(n)$ steps, no tape square beyond number T is scanned.

Proposition symbols:

$P^i_{s,t}$ for $1\le i\le\ell$, $1\le s,t\le T$. $P^i_{s,t}$ is true iff tape square number s at step t contains the symbol σ_i.

Q^i_t for $1\le i\le r$, $1\le t\le T$. Q^i_t is true iff at step t the machine is in state q_i.

$S_{s,t}$ for $1\le s,t\le T$ is true iff at time t square number s is scanned by the tape head.

The formula $A(w)$ is a conjunction $B\&C\&D\&E\&F\&G\&H\&I$ formed as follows. Notice $A(w)$ is in conjunctive normal form.

B will assert that at each step t, one and only one square is scanned. B is a conjunction $B_1 \& B_2 \& \ldots \& B_T$, where B_t asserts that at time t one and only one square is scanned:

$$B_t = (S_{1,t} \vee S_{2,t} \vee \ldots \vee S_{T,t}) \&$$

$$[\underset{1\le i<j\le T}{\&} (\neg S_{i,t} \vee \neg S_{j,t})]$$

For $1\le s\le T$ and $1\le t\le T_j$ $C_{s,t}$ asserts that at square s and time t there is one and only one symbol. C is the conjunction of all the $C_{s,t}$.

D asserts that for each t there is one and only one state.

E asserts the initial conditions are satisfied:

$$E = Q_1^\circ \& S_{1,1} \& P^{i_1}_{1,1} \& P^{i_2}_{2,1} \& \ldots \& P^{i_n}_{n,1}$$

$$\& P^1_{n+1,1} \& \ldots \& P^1_{T,1}$$

where $w = \sigma_{i_1} \ldots \sigma_{i_n}$, q_o is the initial state and σ_1 is the blank symbol.

F, G, and H assert that for each time t the values of the P's, Q's and S's are updated properly. For example, G is the conjuction over all t, i, j of $G^t_{i,j}$, where $G^t_{i,j}$ asserts that if at time t the machine is in state q_i scanning symbol σ_j, then at time t + 1 the machine is in state q_k, where q_k is the state given by the transition function for M.

$$G^t_{i,j} = \underset{s=1}{\overset{T}{\&}} (\neg Q^i_t \vee \neg S_{s,t} \vee \neg P^j_{s,t} \vee Q^k_{t+1})$$

Finally, the formula I asserts that the machine reaches an accepting state at some time. The machine M should be modified so that it continues to compute in some trival fashion after reaching an accepting state, so that A(w) will be satisfied.

It is now straightforward to verify that A(w) has all the properties asserted in the first paragraph of the proof.

Theorem 2: The following sets are P-reducible to each other in pairs (and hence each has the same polynomial degree of difficulty): {tautologies}, {DNF tautologies}, D_3, {subgraph pairs}.

Remark: We have not been able to add either {primes} or {isomorphic graph pairs} to the above list. To show {tautologies} is P – reducible to {primes} would seem to require some deep results in number theory, while showing {tautologies} is P-reducible to {isomorphic graph pairs} would probably upset a conjecture of Corneil's [4] from which he deduces that the graph isomorphism problem can be solved in polynomial time.

Incidently, it not hard to see from the Davis-Putnam procedure [5] that the set D_2 consisting of all DNF tautologies with at most two conjuncts per disjunct, is in $\mathcal{L}*$. Hence D_2 cannot be added to the list in theorem 2 (unless all sets in the list are in $\mathcal{L}*$).

Proof of theorem 2: By the corollary to theorem 1, each of the sets is P-reducible to {DNF tautologies}. Since obviously {DNF tautologies} is P-reducible to {tautologies}, it remains to show {DNF tautologies} is P-reducible to D_3 and D_3 is P-reducible to {subgraph pairs}.

To show {DNF tautologies} is P-reducible to D_3, let A be a proposition formula in disjunctive normal form. Say $A = B_1 \vee B_2 \vee \ldots \vee B_k$, where $B_1 = R_1 \& \ldots \& R_s$, and each R_i is an atom or negation of an atom, and $s > 3$. Then A is a tautology if and only if A' is a tautology where

$$A' = P\&R_3\&\ldots\&R_s \vee \neg P\&R_1\&R_2 \vee B_2 \vee \ldots \vee B_k,$$

where P is a new atom. Since we have reduced the number of conjuncts in B_1, this process may be repeated until eventually a formula is found with at most three conjuncts per disjunct. Clearly the entire process is bounded in time by a polynomial in the length of A.

It remains to show D_3 is P-reducible to {subgraph pairs}. Suppose A is a formula in disjunctive normal form with three conjuncts per disjunct. Thus $A = C_1 \vee \ldots \vee C_k$, where

$C_i = R_{i1}$ & R_{i2} & R_{i3}, and each R_{ij} is an atom or a negation of an atom. Now let G_1 be the complete graph with vertices $\{v_1, v_2, \ldots, v_k\}$, and let G_2 be the graph with vertices $\{u_{ij}\}$, $1 \le i \le k$, $1 \le j \le 3$, such that u_{ij} is connected by an edge to u_{rs} if and only if $i \neq r$ and the two literals (R_{ij}, R_{rs}) do not form an opposite pair (that is they are neither of the form $(P, \neg P)$ nor of the form $(\neg P, P)$). Thus there is a falsifying truth assignment to the formula A iff there is a graph homomorphism $\phi : G_1 \to G_2$ such that for each i, $\phi(v_i) = u_{ij}$ for some j. (The homomorphism tells for each i which of R_{i1}, R_{i2}, R_{i3} should be falsified, and the selective lack of edges in G_2 guarantees that the resulting truth assignment is consistently specified).

In order to guarantee that a one-one homomorphism $\phi : G_1 \to G_2$ has the property that for each i, $\phi(v_i) = u_{ij}$ for some j, we modify G_1 and G_2 as follows. We select graphs $H_1, H_2, \ldots, H_k$ which are sufficiently distinct from each other that if G_1' is formed from G_1 by attaching H_i to v_i, $1 \le i \le k$, and G_2' is formed from G_2 by attaching H_i to each of u_{i1} and u_{i2} and u_{i3}, $1 \le i \le k$, then every one-one homomorphism $\phi : G_1' \to G_2'$ has the property just stated. It is not hard to see such a construction can be carried out in polynomial time. Then G_1' can be embedded in G_2' if and only if $A \notin D_3$. This completes the proof of theorem 2.

2. Discussion

Theorem 1 and its corollary give strong evidence that it is not easy to determine whether a given proposition formula is a tautology, even if the formula is in normal disjunctive form. Theorems 1 and 2 together suggest that it is fruitless to search for a polynomial decision procedure for the subgraph problem, since success would bring polynomial decision procedures to many other apparently intractible problems. Of course the same remark applies to any combinatorial problem to which {tautologies} is P-reducible.

Furthermore, the theorems suggest that {tautologies} is a good candidate for an interesting set not in $\mathcal{L}*$, and I feel it is worth spending considerable effort trying to prove this conjecture. Such a proof would be a major breakthrough in complexity theory.

In view of the apparent complexity of {DNF tautologies}, it is interesting to examine the Davis-Putnam procedure [5]. This procedure was designed to determine whether a given formula in conjunctive normal form is satisfiable, but of course the "dual" procedure determines whether a given formula in disjunctive normal form is a tautology. I have not yet been able to find a series of examples showing the procedure (treated sympathetically to avoid certain pitfalls) must require more than polynomial time. Nor have I found an interesting upper bound for the time required.

If we let strings represent natural numbers, (or k-tuples of natural numbers) using m-adic or other suitable notation, then the notions in the preceeding sections can be made to apply to sets of numbers (or k-place relations on numbers). It is not hard to see that the set of relations accepted in polynomial time by some nondeterministic Turing machine is precisely the set $\mathcal{L}^+$ of relations of the form

(1) $(\exists y \le g_k(\bar{x}))\ R(\bar{x}, y)$

where $g_k(\bar{x}) = 2^{(\ell(\max \bar{x}))^k}$, $\ell(z)$ is the dyadic length of z, and $R(\bar{x},y)$ is an $\mathcal{L}^*$ relation, ($\mathcal{L}^+$ is the class of extended positive rudimentary relations of Bennett [6]). If we remove the bound on the quantifier in formula (1), the class $\mathcal{L}^+$ would become the class of recursively enumerable sets. Thus if $\mathcal{L}^+$ is the analog of the class of r.e. sets, then determining tautologyhood is the analog of the halting problem; since, according to theorem 1, {tautologies} has the complete $\mathcal{L}^+$ degree just as the halting problem has the complete r.e. degree. Unfortunately, the diagonal argument which shows the halting problem is not recursive apparently cannot be adapted to show {tautologies} is not in $\mathcal{L}^*$.

3. The Predicate Calculus

Formulas in the predicate calculus are represented by strings in a manner similar to the propositional calculus. In addition to the symbols for the latter, we need the quantifier symbols $\forall$ and $\exists$, and symbols for forming an infinite list of individual variables, and infinite lists of function and predicate symbols of each order (of course the underlying alphabet Σ is still finite).

Suppose Q is a procedure which operates on the above formulas and which terminates on a given input formula A iff A is unsatisfiable. Since there is no decision procedure for satisfiability in the predicate calculus, it follows that there is no recursive function T such that if A is unsatisfiable, then Q will terminate within T(n) steps, where n is the length of A. How then does one appraise the efficiency of the procedure?

We will take the following approach. Most automatic theorem provers depend on the Herbrand theorem, which states briefly that a formula A is unsatisfiable if and only if some conjunction of substitution instances of the functional form fn(A) of A is truth functionally inconsistent. Suppose we order the terms in the Herbrand universe of fn(A) according to rank, and then order in a natural way the substitution instances of fn(A) from the Herbrand universe. The ordering should be such that in general substitution instances which use terms with greater rank follow substitution instances which use terms of lesser rank. Let $A_1, A_2, \ldots$ be these substitution instances in order.

Definition: If A is unsatisfiable, then $\phi(A)$ is the least k such that $A_1 \& A_2 \& \ldots \& A_k$ is truth-functionally inconsistent. If A is satisfiable, then $\phi(A)$ is undefined.

Now let Q be the procedure which, given A, computes the sequence $A_1, A_2, \ldots$ and for each i, tests whether $A_1 \& \ldots \& A_i$ is truth-functionally consistent. If the answer is ever no, the procedure terminates successfully. Then clearly there is a recursive T(k) such that for all k and all formulas A, if the length of $A \leq k$ and $\phi(A) \leq k$, then Q will terminate within T(k) steps. We suggest that the function T(k) is a measure of the efficiency of Q.

For convenience, all procedures in this section will be realized on single tape Turing machines, which we shall call simply machines.

Definition: Given a machine M_Q and recursive function $T_Q(k)$, we will say M_Q is of type Q and runs within time $T_Q(k)$ provided that when M_Q starts with a predicate formula A written on its tape, then M_Q halts if and only if A is unsatisfiable, and for all k, if $\phi(A) \leq k$ and $|A| \leq \log_2 k$, then M_Q halts within $T_Q(k)$ steps. In this case we will also say that $T_Q(k)$ is of type Q. Here $|A|$ is the length of A.

The reason for the condition $|A| \leq \log_2 k$ instead of $|A| \leq k$, is that with the latter condition, finding a lower bound for $T_Q(k)$ would be nearly equivalent to finding a lower bound for the decision problem for the propositional calculus. In particular, theorem 3A would become obvious and trivial.

Theorem 3: A) For any $T_Q(k)$ of type Q,

$$(2) \qquad \frac{T_Q(k)}{\sqrt{k}\,/(\log k)^2} \quad \text{is unbounded.}$$

B) There is a $T_Q(k)$ of type Q such that

$$T_Q(k) \le k\; 2^{\,k(\log k)^2}$$

Outline of proof: A). Given any machine M, one can construct a predicate formula A(M) which is satisfiable if and only if M never halts when starting on a blank tape. This is done along the lines described in Wang [7] in the proof which reduces the halting problem to the decision problem for the predicate calculus. Further, if M halts in s steps, then $\phi(A(M)) \le s^2$. Thus, if, contrary to (2), $T_Q(k) = 0(\sqrt{k}/\log^2 k)$, then a modification of M_Q could verify in only

$$0(\sqrt{s^2}/\log^2 s^2) = 0(s/\log^2 s)$$

steps that M halted in s steps (provided $m \le \log s^2$, where m is the length of A(M)). A diagonal argument (see [8] p. 153) shows that this is impossible in general.

B) The machine M_Q operates in time T_Q by following the procedure outlined at the beginning of this section. Note that the formula A_1 & A_2 & ... & A_k has length $0(k \log^2 k)$, since we can assume $|A| \le \log k$.

Theorem 4: If the set S of strings is accepted by a nondeterministic machine within time $T(n) = 2^n$, and if $T_Q(k)$ is an honest (i.e. real-time countable) function of type Q, then there is a constant K so S can be recognized by a deterministic machine within time $T_Q(K8^n)$.

Proof: Suppose M_1 is a nondeterministic machine which accepts S in time 2^n. Let M_2 be a nondeterministic machine which simulates M_1 for exactly 2^n steps and then halts, unless M_1 accepts the input, in which case M_2 computes forever. Thus for all strings w, if $w \in S$ then there is a computation for which M_2 with input w fails to halt, and if $w \notin S$, then M_2 with input w halts within 4^n steps for all computations. Now given w of length n, we may construct a formula A(w) of length 0(n) such that A(w) is satisfiable if and only if M_1 accepts w. (A(w) is constructed in a way similar to A(M) in the proof of 1A). Further, if M_2 halts within 4^n steps for all possible computations, then $\phi(A(w)) \le K(4^n)^2 = K8^n$. Thus, a deterministic machine M can be constructed to determine whether $w \in S$ by presenting M_Q with input A(w). If no result appears within $T_Q(K8^n)$ steps, then $w \in S$, and otherwise $w \notin S$.

4. More Discussion:

There is a large gap between the lower bound of $\sqrt{k}/(\log k)^2$ for time functions $T_Q(k)$ given in theorem 3A and a possible

$$T_Q(k) = k2^{k(\log k)^2}$$

given in 3B. However, there are reasons for the gap. For example, if we could improve the result in 3B and find a $T_Q(k)$ bounded by a polynomial in k, then by theorem 4 we could simulate a nondeterministic 2^n time bounded machine deterministically in time $p(2^n)$ for some polynomial p. This is contrary to experience which indicates deterministic simulation of a nondeterministic T(n) time bounded machine requires time $k^{T(n)}$ in general.

On the other hand, if we could push up the lower bound given in theorem 3A and show

$$\frac{T_Q(k)}{2^k}$$

is unbounded, then we could conclude $\{\text{Tautologies}\} \notin \mathcal{L}*$, since otherwise the general Herbrand proof procedure would provide a $T_Q(k)$ smaller than 2^k. Thus such an improvement in 3A would require a major breakthrough in complexity theory.

The field of mechanical theorem proving badly needs a basis for comparing and evaluating the dozens of procedures which appear in the literature. Performance of a procedure on examples by computer is a good criterion, but not sufficient (unless the procedure proves useful in some practical way). A theoretical complexity criterion is needed which will bring out fundamental limitations and suggest new goals to pursue. The criterion suggested here (the function $T_Q(k)$) is probably too crude. For example, it might be better to make $T_Q(k)$ a function of several variables, of which one is $\phi(A)$, and another might be the minimum number of substitution instances of fn(A) needed to form a contradiction (note that in general not all of $A_1, A_2, \ldots, A_{\phi(A)}$ are needed.)

$T_Q(k)$ may be a crude measure, but it does provide a basis for discussion, and, I hope, will stimulate progress toward finding better complexity measures for theorem provers.

REFERENCES

1. D. L. Kreider and R. W. Ritchie: Predictably Computable Functionals and Definitions by Recursion. Zeitschrift für math. Logik und Grundlagen der Math., Vol. 10, 65-80 (1964).

2. S. A. Cook: Characterizations of Pushdown Machines in terms of Time-Bounded Computers. J. Assoc. Computing Machinery, Vol. 18, No. 1, Jan. 1971, pp 4-18.

3. Cobham, Alan: The intrinsic computational difficulty of functions. Proc. of the 1964 International Congress for Logic, Methodology, and the Philosophy of Science, North Holland Publishing Co., Amsterdam, pp. 24-30.

4. D. G. Corneil and C. C. Gotlieb: An Efficient Algorithm for Graph Isomorphism. J. Assoc Computing Machinery Vol. 17, No. 1, Jan. 1970, pp 51-64.

5. M. Davis and H. Putnam: A Computing Procedure for Quantification Theory. J. Assoc. Computing Machinery, 1960, pp. 201-215.

6. J. H. Bennett: On Spectra. Doctoral Dissertation, Princeton University, 1962.

7. Hao Wang: Dominoes and the AEA case of the decision problems. Proc. of the Symposium on Mathematical Theory of Automata, at Polytechnic Institute of Brooklyn, 1962. pp. 23-55.

8. John Hopcroft and Jeffrey Ullman: Formal Languages and their Relation to Automata. Addison-Wesley, 1969.