THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

On the Unique Satisfiability Problem

Andreas Blass and Yuri Gurevich

CRL-TR-1-83

JANUARY 1983

Computing Research Laboratory
Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1]All correspondence should be sent to Professor Yuri Gurevich. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

engm

UMR0978

On the Unique Satisfiability Problem

Andreas Blass, Department of Mathematics, and

Yuri Gurevich, Department of Computer and Communication Sciences

The University of Michigan
Ann Arbor, Michigan 48109

JANUARY 1983

ABSTRACT

UNIQUE SAT is the problem of deciding whether a given Boolean formula has exactly one satisfying truth assignment. We address the question whether UNIQUE SAT is complete for the class $DIF^P = \{L_1 - L_2 : L_1, L_2 \in NP\}$. We construct an oracle relative to which UNIQUE SAT is not complete for DIF^P, and another oracle relative to which UNIQUE SAT is complete for DIF^P whereas $NP \neq co-NP$. (We care about $NP \neq co-NP$ because the equality $NP = co-NP$ easily implies that UNIQUE SAT is complete for DIF^P.)

1. Introduction

UNIQUE SAT is the following well-known problem. Given a Boolean formula, is it true that it has exactly one satesfying truth assignment? We are interested in the complexity of UNIQUE SAT relative to many-one polynomial time reductions. (As it often happens, the reductions constructed are in fact log-space computable.)

UNIQUE SAT is easily seen to be co-NP-hard. Indeed, the set of unsatisfiable Boolean formulas, which is well-known to be co-NP-complete, is reducible to UNIQUE SAT by assigning to each Boolean formula $\varphi(x_1, \ldots, x_n)$ the formula

$$(x_0 \wedge x_1 \wedge \cdots \wedge x_n) \vee (\sim x_0 \wedge \varphi(x_1, \ldots, x_n)) .$$

It is also easy to see that UNIQUE SAT belongs to the class Δ_2^P of problems solvable in polynomial time by an algorithm with an NP oracle. The oracle is used to learn whether the given formula has at least one satisfying truth assignment and whether it has at least two.

Papadimitriou and Yannakakis [2] introduced and studied the class

$$\begin{aligned} DIF^P &= \{L_1 - L_2 : L_1, L_2 \in NP\} \\ &= \{L_1 \cap L_2 : L_1 \in NP \text{ and } L_2 \in co-NP\} . \end{aligned}$$

They observe that UNIQUE SAT, considered as a set of formulas, is obviously in DIF^P. It is SAT minus $\{\ \varphi : \varphi$ has two distinct satisfying assignments$\}$. They write:

> We note here that the precise complexity of UNIQUE SAT is a persistent open question. Our observation casts doubt on the recurring conjecture that it is complete for Δ_2^P.

We address the question whether UNIQUE SAT is complete for DIF^P. It will be useful to consider this question in a somewhat more general context. Let UNIQUE SOLUTION, or simply US, be the class of sets $L \subseteq \sum^*$ that can be represented in the form

$$L = \{x \in \textstyle\sum^* | \exists ! y \in \sum_1^* R(x,y)\}$$

where R is a polynomially bounded polynomial-time computable relation. Here $\sum$ and $\sum_1$ are finite alphabets and "polynomially bounded" means that there is apolynomial p such that length(y) $\leq p$(length(x)) whenever $R(x,y)$. It is clear that UNIQUE SAT belongs to the class US, since we can take $R(x,y)$ to be "x is a Boolean formula and y is a truth assignment satisfying it." Furthermore, the same technique, of coding computations as truth assignments, that is used to prove the NP-completeness of SAT, can also be used to prove that UNIQUE SAT is complete for the class US.

The preceding remarks, along with the observation that US is closed under polynomial-time many-one reducibility, easily imply that $US \subseteq DIF^P$ and that the following three questions are equivalent.

(1) Is UNIQUE SAT complete for DIF^P ?

(2) Is DIF^P included in US?

(3) Is DIF^P equal to US?

They also imply that the following two questions are equivalent to each other.

(4) Is UNIQUE SAT NP-hard?

(5) Is NP included in US?

In fact, all five of these questions are equivalent. Comparing (2) and (5), we see that it suffices to show that, if $NP \subseteq US$ then $DIF^P \subseteq US$. Using the second formulation of the definition of DIF^P and the fact that $co-NP \subseteq US$ (because UNIQUE SAT is co-NP-hard), we see that it suffices to show that US is closed under intersection. But this is easy;

$$\{x | \exists ! y \; R_1(x,y)\} \cap \{x | z \; R_2(x,z)\} = \left\{x | \exists !(y,z) \left[R_1(x,y) \wedge R_2(x,z)\right]\right\}.$$

We shall work mainly with formulation (5) of the original question (1). This is partly because NP is easier to work with than DIF^P and partly because the class US is easier to relativize to an oracle than the specific problem UNIQUE SAT. The appropriate relativization of UNIQUE SAT, like that of SAT, involves asking about truth assignments subject to certain constraints relating the truth values to the oracle. This is more awkward than the relativization of the class US, which involves merely allowing the use of the oracle in the computation of R in the definition of US. It is, of course, possible to work instead with other US complete problems that are easier to relativize; one such is the unique halting problem, defined as follows. An instance consists of a (standard code for a) nondeterministic Turing machine M and a natural number k in unary notation. The question about the instance (M,k) is whether M, with empty input, has exactly one halting computation of length at most k. This problem is easily seen to be US complete, and it can be relativized by simply taking M to be equipped with an oracle.

It is conceivable that our question has an affirmative answer for trivial reasons. Specifically, it is conceivable that $NP = co-NP$, and, in this situation, the fact that UNIQUE SAT is co-NP-hard immediately answers (4). Therefore, to avoid such trivialities, we are interested only in what happens under the assumption that $NP \neq co-NP$.

We shall show that, in a sense, anything can happen. In § 2 we construct an oracle relative to which $NP \not\subseteq US$ and in § 3 we constrct an oracle relative to which $NP \subseteq US$ but $NP \neq co-NP$. In view of the equivalence of (1) through (5), our results imply that UNIQUE SAT and the unique halting problem, relativized to oracles as indicated above, can be complete or incomplete for DIF^P, depending on the oracle. Of course, if they are incomplete for DIF^P then they are *a fortiori* incomplete for the larger class Δ_2^P, so the recurring conjecture quoted by Papadimitriou and Yannakakis is false relative to some oracles.

2. An oracle making NP $\not\subseteq$ US

Theorem 1. There is an oracle $A \subseteq \{0,1\}^*$ *such that* $NP^A \not\subseteq US^A$.

Proof. We construct A in stages; initially it is empty, and at each stage a finite number (possibly zero) of words will be added to it. Also, at each stage, we will "freeze" A up to some length l ; this means that we decide not to add words of length l or less at any later stage.

We work with a fixed enumeration, $M_1, M_2, \ldots,$ of all nondeterministic query machines (Turing machines equipped to interact with an as yet unspecified oracle) with polynomial clock bounds $p_1, p_2, \cdots$. The n^{th} stage of the construction of A will be devoted to ensuring that the NP^A set

$$\left\{x \mid \exists y \left[y \in A \text{ and length}(x) = \text{length}(y)\right]\right\} \tag{6}$$

is different from the set

$$\{x \mid M_n^A \text{ has exactly one halting computation on input } x\}. \tag{7}$$

This will suffice to prove the theorem, because every set in US^A , say

$$\{x \mid \exists! y \; R(x,y)\}$$

(with R polynomially bounded and polynomial-time computable in A), is of the form (7) for the machine M_n which, with input x , guesses y and checks $R(x,y)$.

We now describe stage n of the construction. To simplify notation, we write M and p for M_n and p_n . We write A for the set of words put into A at previous stages (the current A) and l for the length up to which A has been frozen in previous stages. We begin by fixing a natural number d , which will be the length of all the words (if any) to be added to A at the present stage. This d is to be chosen larger than l (so that the present stage will not violate the previous freeze), larger than the lengths of the present members of A , and so large that $p(d) < 2^{d-1}$. We consider three cases.

Case 1: M^A , on input 0^d , has exactly one halting computation. Then add nothing to A , but increase l to be larger than d and $p(d)$. This increase in l guarantees that the case hypothesis will remain true despite any future additions to A because all the queries are about words shorter than $p(d)$; thus 0^d belongs to the set in (7). But 0^d does not belong to the set in (6) since we have not added, and will not add, any words of length d to A . Thus we have achieved our goal in this case.

Case 2: M^A , on input 0^d , has two or more halting computations. Choose two such computations and find an $x \in \{0,1\}^d$ such that neither of these two computations involves a query about x ; this is possible because, by the clock bound, each computation uses at most $p(d)$ queries and $2p(d) < 2^d$. Add x to A and increase l to be larger than d and larger than $p(d)$. The choice of x and this increase in l guarantee that, even after all future additions to A , the same two halting computations still exist and show that 0^d will not belong to the set in (7). But, it will belong to the set in (6) because we have added x to A . So we have achieved our goal in this case.

Case 3: Neither of the previous cases applies and there is a nonempty subset B of $\{0,1\}^d$ such that $M^{A \cup B}$ does not have exactly one halting computation on input 0^d . Then choose such a B and add all its members to A . Also increase l to be larger than d and larger than $p(d)$. This increase in l again guarantees that the case hypothesis will remain true despite all future additions to A . Therefore, for the final A , 0^d does not belong to the set in (7), but it does belong to the set in (6) because B is nonempty. So we have achieved our goal in this case as well.

To complete the proof, we derive a contradiction from the assumption that none of the three cases occurs. This assumption means that M^A has no halting computation on 0^d but, for every nonempty $B \subseteq \{0,1\}^d$, $M^{A \cup B}$ has exactly one halting computation on 0^d. We define an equivalence relation on the collection of all these nonempty sets B by calling B and B' equivalent if and only if the unique halting computations of $M^{A \cup B}$ and $M^{A \cup B'}$ on 0^d are identical. Thus, if we define the relevant part of B to be

$$Rel(B) = \left\{ y \in \{0,1\}^d \mid \text{The unique halting computation of } M^{A \cup B} \text{ on } 0^d \text{ involves a query about } y \right\},$$

then the equivalence class of B consists of those B' such that $B' \cap Rel(B) = B \cap Rel(B)$. This equivalence class has 2^m elements, where $m = 2^d - |Rel(B)|$. Since $|Rel(B)| \leq p(d) < 2^d$, every equivalence class has an even number of members. But these equivalence classes partition the collection of all nonempty subsets of $\{0,1\}^d$, and the number of such subsets is $2^{2^d} - 1$, which is odd. This contradiction proves that one of the three cases must occur, so the construction of A and the proof of Theorem 1 are complete.

3. An oracle making NP $\subseteq$ US but NP $\neq$ co-NP

Throughout this section, we use " M " to denote a nondeterministic Turing machine with a polynomial clock bound. Because of the clock bound, every computation will terminate; we use "halt" to mean "terminate in an accepting state." We let Halt(M) be the set of halting computations of M on empty input. We code computations by binary words in such a way that the machine itself is easily reconstructible from any of its computations.

By a *tagged computation* of M , we mean a binary word of even length whose first half is in Halt(M); the second half, called the tag, is arbitrary.

Lemma. Suppose that L is a polynomial-time computable set that contains exactly one tagged computation of M whenever Halt(M) *is not empty. Then $NP \subseteq US$.*

Proof. For any M and any input x , let M_x be the machine which, on empty input, prints x on the input tape and then behaves like M . Then the set of words accepted by M is

$$\Big\{x \mid \exists! y \,\Big[y \in L, \text{ and length}(y) \text{ is even}, \text{ and the first half of } y \text{ is in Halt}(M_x)\Big]\Big\} . \tag{8}$$

The predicate after the unique existential quantifier in (8) is clearly computable in polynomial time and it is polynomially bounded because of the clock bound on M . Thus, the NP set accepted by M is in US .

Theorem 2. There is an oracle $C \subseteq \{0,1\}^*$ such that $NP^C \subseteq US^C$ and $NP^C \neq co-NP^C$.

Proof. We shall construct C so that its "even part"

$$A = \{x \in C \mid \text{length}(x) \text{ is even}\}$$

contains exactly one tagged computation for each M^C such that Halt(M^C) is nonempty; it will follow, by the lemma relativized to C, that $NP^C \subseteq US^C$. The "odd part" of C,

$$B = \{x \in C \mid \text{length}(x) \text{ is odd }\}$$

will be constructed so that the NP^C set

$$\text{Lengths}(C) = \{0^l \mid C \text{ contains a word of length } l\}$$

is not in co-NP^C; this part of the construction is essentially as in [1]. The subtlety here is to ensure that the two parts of the oracle do not obstruct each other's intended purpose.

The construction of A and B proceeds by stages. In addition we construct also auxiliary sets A' and B' of words explicitly forbidden to be put into A and B respectively. We describe below a stage n of the construction distinguishing between the cases when n is even or odd. The current finite approximations to A, A', B, B' are called simply A, A', B, B'. On the other hand, C always refers to the final value. From time to time we update these sets by putting in additional words. At each stage, A will be ***reasonable*** in the sense that, for each M and Z, at most one tagged computation of M^Z belongs to A. After stage n, $A \cup A' \cup B \cup B'$ will contain all words of length at most n, as well as possibly some longer words. The construction will have the property that words of an odd length n are added to B only at stage n; words of an even length n may be added to A at stage n and possibly at one earlier odd stage.

Stage n, for n even: Inspect, in lexicographic order, all binary words of length n. Whenever you find one that (i) is not in A' and (ii) is a tagged computation for some M^C that has no tagged computation already in A, put that word into A. All words of length n not in A or A' when this procedure is finished are to be put into A'. Observe that, in deciding whether a word x of length n is a tagged computation for some M^C, we can (easily) determine the relevant M from x, and we can check the correctness of all the oracle's answers in the alleged computation because the queries are shorter than x and $A \cup A' \cup B \cup B'$ already contains all words of length less than n. Thus, the construction at even stages is well defined. It clearly preserves the reasonableness of A, since we add a tagged computation for M^C only if none was previously present. If we can verify that, for every M^C with Halt(M^C) nonempty, some tagged computation is eventually put into A, then the lemma will imply, since A is clearly polynomial-time computable from C, that $NP^C \subseteq US^C$. This verification, however, must be postponed, since it depends on what happens at odd stages.

Stage n, for n odd. At odd stages, we seek to "defeat" each machine M by ensuring that the set accepted by M^C is not the complement of Lengths(C); if we do this for every M, we shall have $NP^C \neq co-NP^C$ as desired. At stage n, consider the first (in a standard enumeration) machine M not defeated at any earlier odd stage, and let p be its polynomial clock bound. We assume, without loss of generality, that $p(n) \geq n$ for all n. Unless both

$$p(n) < 2^{n/2} \tag{9}$$

and

$$\text{all words in } A \cup A' \cup B \cup B' \text{ have length less than } n\ , \tag{10}$$

we simply add to B' all words of length n and go to stage $n+1$, leaving M undefeated and thus to be considered again at stage $n+2$. Recall that the construction is to have the property that words of an odd length n are added to B only at stage n; this ensures that no word of length n is already in B, so it is permissible to add them all to B'. If both (9) and (10) hold, we shall defeat M. Before proceeding, however, we observe that every M will eventually be defeated. For, suppose not, and let M be the first machine (in the enumeration) that is not defeated. Then, at all sufficiently late odd stages, the machine under consideration is M and (9) holds. Furthermore, at these stages and the intervening even stages, no word longer than the stage number is added to $A \cup A' \cup B \cup B'$. It follows that (10) will eventually hold, and M will be defeated, contrary to our assumption.

The procedure for defeating M, when (9) and (10) hold, is as follows. Consider all sets X that

(i) consist of words of even lengths $\leq p(n)$,

(ii) include A and are disjoint from A', and

(iii) are reasonable (in the sense indicated above for A).

If there is no such X for which $M^{X \cup B}$ accepts 0^n, then we add to B' all words of odd lengths between n and $p(n)$ inclusive. (None of these are in B, so this is permissible.) Thus, when the construction is complete, 0^n will be in the complement of Lengths(C). We assert that, when the construction is complete, M^C will not accept 0^n, so M is defeated. To prove this assertion, suppose M^C had an accepting computation with input 0^n. It involves queries only of lengths $\leq p(n)$, so it is also an accepting computation relative to any oracle that agrees with C on words of lengths $\leq p(n)$, in particular relative to the oracle $X \cup B$, where

$$X = \left\{ x \in C \,|\, \text{length}(x) \text{ is even and } \leq p(n) \right\}$$

and where B is, as before, the *present* value of B ; it is important here that we added to B' words of length up to $p(n)$, so that the present B agrees with the eventual C for these lengths. Since X satisfies (i), (ii), and (iii), $M^{X \cup B}$ has no accepting computation on input 0^n , so we have a contradiction. Thus, M is defeated.

There remains the case that there is an X satisfying (i), (ii), and (iii) such that $M^{X \cup B}$ has an accepting computation on input 0^n . In this case, choose such an X and choose such a computation. Add to A all elements of X that occur as queries in this computation, add to A' all words of even length that occur as queries but are not in X , and add to B' all words of odd length that occur as queries but are not in B . These additions guarantee that, when the construction is finished, M^C will accept 0^n by virtue of exactly the same computation. Then add to B all words of length n that were not added to B' . Notice that at most $p(n)$ words were added to B' , so, by (9), some words were added to B . Therefore 0^n will be in Lengths(C) , and M is defeated.

This completes the construction, but we must still see that the even stages achieve the desired effect, that every M with Halt(M^C) nonempty has a tagged computation in A . So suppose u , of length l , is the lexicographically first among the shortest words in Halt(M^C) . Suppose that no tagged computation for M^C was put into A before stage $n = 2l$. Then at this stage we encounter the 2^l tagged computations uv , where v ranges over all binary words of length l . If any of these are not in A' when stage n begins, then the lexicographically first will be added to A as desired. We complete the proof by establishing that, when stage n begins, A' has fewer than 2^l elements of length n .

At even stages k prior to stage n, A' acquires only words of length k, so any word of length n that is in A' when stage n begins must have been added to A' at an earlier *odd* stage k at which (9) and (10) hold (for otherwise only B' acquires members at stage k). Furthermore, all such words must have been added at the same stage k, for if one is added at stage k then (10) fails for all k' with $k < k' < n$. So we need only show that, for any odd $k < n$, fewer than 2^l words are added to A' at stage k. But the words added to A' at stage k are those that occur as queries in a certain computation of length $p(k)$, where p is the polynomial clock bound used at stage k. Since (9) holds at stage k, the number of such queries is at most

$$p(k) < 2^{k/2} < 2^{n/2} = 2^l ,$$

as desired. This completes the proof of Theorem 2.

Remark: By slightly altering the definition of "reasonable" we can arrange that, whenever a tagged computation for M^C occurs in C, its first half is the lexicographically first among the shortest members of Halt(M^C).