TWO RESULTS ON POLYNOMIAL TIME TURING REDUCTIONS TO SPARSE SETS

by

Esko Ukkonen

Memorandum No. UCB/ERL M81/68

11 September 1981

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Two results on polynomial time Turing reductions to sparse sets*

Esko Ukkonen
Department of Computer Science, University of Helsinki
and
Computer Science Division, University of California, Berkeley

ABSTRACT

Inspired by the recent solution of the Berman-Hartmanis conjecture that NP cannot have a sparse complete set for many-one reductions unless $P=NP$, we analyze the implications of NP or $PSPACE$ having sparse hard or complete sets for Turing reductions. Three special cases of Turing reductions are considered. First we show that if NP ($PSPACE$) has a tally $\leq_{k-T}^{P}$-hard set then $P=NP$ ($P=PSPACE$). Here $\leq_{k-T}^{P}$ denotes a subcase of polynomial time Turing reducibility in which, for some constant k, the reducing Turing machine is allowed to make at most k queries to the oracle. We also show that if $co-NP$ ($PSPACE$) has a sparse hard set for conjunctive polynomial time reductions then $P=NP$ ($P=PSPACE$).

Key words: Turing reduction, polynomial time, hard set, sparsity.

1. Introduction

This paper is one in a series of research initially stimulated by a conjecture of L. Berman and J. Hartmanis [1] that all sets $\leq_m^P$-complete for NP are polynomial time isomorphic; i.e. that between any two such sets there is a polynomial time bijective reduction with polynomial time inverse. Because the conjecture implies that $P \neq NP$, it understandably is still open.

Another implication of the conjecture is that all $\leq_m^P$-complete sets have similar density. Since all the known complete sets have exponential density [1],

*This work was carried out when I was visiting the Computer Science Division of the University of California at Berkeley. I am indebted to Professor Michael A. Harrison for providing me with this opportunity. The work was supported by the Academy of Finland and by the Finnish Cultural Foundation. Partial support was provided by the National Science Foundation Grant MCS 79-15763 (Univ. of California).

Berman and Hartmanis additionally conjectured that no sparse set (a set with polynomial density) could be $\leq_m^P$-complete for NP unless $P=NP$. Recently, this conjecture was proved by Mahaney [8]. However, the first significant step towards the solution was made already by P. Berman [2] who proved that if NP has an $\leq_m^P$-complete set in 1^* (a *tally* complete set), then $P=NP$. Fortune [3], and later Meyer and Paterson [9], improved Berman's result by showing that if NP has a co-sparse $\leq_m^P$-complete set, then $P=NP$.

With the Berman-Hartmanis conjecture solved for $\leq_m^P$-complete sets, it is natural to consider implications of NP having sparse hard or complete sets with respect to reductions more general than $\leq_m^P$. Particularly interesting is the polynomial time Turing reduction $\leq_T^P$ since, as noted by Meyer (see [1]), NP has polynomial size circuits if and only if NP has a sparse $\leq_T^P$-hard set. For Turing reductions it is known [5] that if NP has a sparse $\leq_T^P$-hard set (or equivalently, polynomial size circuits) then the polynomial hierarchy collapses to $\Sigma_2^P \cap \Pi_2^P$. Mahaney [8] establishes a related result by showing that if the sparse $\leq_T^P$-hard set for NP is actually in NP, then the polynomial hierarchy collapses to Δ_2^P. Finally Long [7] proves a companion result that the polynomial hierarchy equals Δ_2^P if NP has a co-sparse $\leq_T^P$-complete set.

Thus the question whether the existence of a sparse (or co-sparse) $\leq_T^P$-hard set for NP implies $P=NP$ is still open. Instead of the $\leq_T^P$-reducibility, which is the most general form of polynomial time reducibilities, we consider in this paper some more restricted reductions which still are properly more general than $\leq_m^P$. First we consider polynomial time Turing reductions restricted to machines which may make at most a constant number of queries to the oracle. Such reducibility is denoted by $\leq_{k-T}^P$ where k is the integer constant limiting the number of queries. We show, generalizing an original result of P. Berman [2], that if NP has a $\leq_{k-T}^P$-hard tally set (i.e., a set over one symbol alphabet), then $P=NP$. We also consider two other subcases of Turing reductions, the polynomial time conjunctive and disjunctive reductions $\leq_c^P$ and $\leq_d^P$. It is shown, generalizing a result of Fortune [3], that if $co-NP$ has an $\leq_c^P$-hard sparse set (or, equivalently, NP has an $\leq_d^P$-hard co-sparse set), then $P=NP$. These proofs are easily modified to show that if $PSPACE$ has a tally $\leq_{k-T}^P$-hard set or sparse $\leq_c^P$-hard set (or co-sparse $\leq_d^P$-hard set) then $P=PSPACE$.

2. Polynomial time reducibilities

We assume familiarity with classes P, NP, $co-NP$, $PSPACE$ and with the concepts of hardness and completeness for a class of languages with respect to a given form of reductions between languages [4]. Unless specified otherwise, all sets are languages over a fixed finite alphabet Σ of at least two elements. In particular, sets in 1^*, i.e. sets over one symbol alphabet, are called *tally languages*. For $A \subset \Sigma^*$, $\bar{A}$ denotes the complement of A in Σ^*. For a string x, $|x|$ denotes the length of x.

An oracle Turing machine is a deterministic multitape Turing machine acceptor with a distinguished oracle tape and three special states Q, YES, NO. By an acceptor we mean a machine where final states are divided to accept and reject states. When the oracle machine enters state Q, the next state is YES or NO depending on whether or not the word currently on the oracle tape belongs to the oracle set. In this way, the machine with an oracle set B receives an answer to a test of the form "$x \in B$?" in one step. The set accepted by the oracle Turing machine M with B as its oracle set will be denoted by $L(M,B)$.

A set A is Turing reducible to a set B in polynomial time ($A \leq_T^P B$), if $A = L(M,B)$ for an oracle Turing machine M that runs in polynomial time.

Set A is many-one reducible to set B in polynomial time ($A \leq_m^P B$) if there is a function $f: \Sigma^* \rightarrow \Sigma^*$, computable in polynomial time, such that for all $x \in \Sigma^*$, $x \in A$ if and only if $f(x) \in B$.

The computations of an oracle Turing machine M, which operates with input x, can be described by a binary computation tree $T = T(M,x)$. The root of T is labeled x. Every leaf is labeled either "accept" or "reject". Every internal node is labeled with a query. Every right branch is labeled "yes" and every left branch is labeled "no" corresponding to whether the state following the query state is YES or NO. A path from the root to a leaf is called an *accept path* (*reject path*) if the leaf is labeled "accept" ("reject").

In what follows we analyze three restricted forms of the Turing reductions. The first one is simply the $\leq_T^P$-reduction limited to oracle machines that independently of the input and the oracle may ask only a constant number of questions. In more detail, let $A \leq_T^P B$ via a machine M so that, for some fixed integer k, the height of the computation tree $T(M,x)$ is at most k for every input x. Then we write $A \leq_{k-T}^{P} B$ and say that A is *k-question Turing reducible to B in polynomial time*. Thus independently of the current oracle, any computation of M contains at most k queries.

To give the two remaining reductions we say that a set A is *disjunctive reducible to a set B in polynomial time*, $A \leq_{d}^{P} B$, if $A \leq_{T}^{P} B$ via an oracle Turing machine whose accept states contain only the YES state. Furthermore, a set A is *conjunctive reducible to a set B in polynomial time*, $A \leq_{c}^{P} B$, if $A \leq_{T}^{P} B$ via a machine whose reject states contain only the NO state. These definitions are more convenient to work with than the equivalent definitions in the literature (e.g. [6]). Note that if $A \leq_{d}^{P} B$ via a machine M then in every computation tree of M, the path leading to the leftmost leaf is the only reject path. Similarly, if $A \leq_{c}^{P} B$ via M then the only accept path is the path leading to the rightmost leaf.

For completeness, we will recall from the literature some additional forms of reducibility between languages and compare them with those given above. First, an oracle Turing machine is called *positive* if whenever an oracle set B is a subset of another oracle set B', then $L(M,B) \subseteq L(M,B')$. Set A is *positive reducible to a set B in polynomial time*, $A \leq_{p}^{P} B$, if $A \leq_{T}^{P} B$ via a positive Turing machine [11]. Clearly, both $\leq_{c}^{P}$ and $\leq_{d}^{P}$ are positive reductions.

Next recall that according to the original definition [10], a set A is *truth–table reducible* to a set B ($A \leq_{tt} B$) if there is a recursive function f that on input x computes a list of queries $q_1, \ldots, q_k$ and a boolean function α such that $x \in A$ if and only if $\alpha(C_B(q_1), \ldots, C_B(q_k)) = 1$ where C_B is the characteristic function of the set B.

The following characterization of $\leq_{tt}$ is useful from the complexity theoretic point of view [11]. Let c be a symbol not in $\{0,1\}$. Then $A \leq_{tt} B$ if and only if there is an oracle Turing machine M such that $A \leq_{T} B$ via M and a recursive function $f: \{0,1\}^* \rightarrow (c\{0,1\}^*)^*$ such that, for each input x to M, M only makes queries to B from the list $f(x)$. If here M operates in polynomial time and f is polynomial time computable, then we say that A is *truth-table reducible to B in polynomial time* ($A \leq_{tt}^{P} B$). In addition, if M is positive, then A is *positive truth-table reducible to B in polynomial time* ($A \leq_{ptt}^{P} B$), and if $f: \{0,1\}^* \rightarrow (c\{0,1\}^*)^k$ for some fixed integer k (thus the length of $f(x)$ is always k), then A is *k-question truth-table reducible to B in polynomial time* ($A \leq^P_{k\text{-}tt} B$).

The basic relations between these reducibilities are now outlined. From [6] we quote:

Theorem 1 [6].

$$\begin{tikzcd}
 & A \leq^P_c B \arrow[dr, Rightarrow] & & \\
A \leq^P_m B \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & & A \leq^P_{ptt} B \arrow[r, Rightarrow] & A \leq^P_{tt} B. \quad \blacksquare \\
 & A \leq^P_d B \arrow[ur, Rightarrow] & &
\end{tikzcd}$$

We also have:

Theorem 2. $A \leq^P_m B \Rightarrow A \leq^P_{k\text{-}tt} B \Rightarrow A \leq^P_{k\text{-}T} B \Rightarrow A \leq^P_{(2^k-1)\text{-}tt} B \Rightarrow A \leq^P_{tt} B \Rightarrow A \leq^P_T B.$

Proof. All the implications in the theorem are almost trivial. We show here only the second and the third one.

Let $A \leq^P_{k\text{-}tt} B$ via a machine M. For each input the list of allowed queries is of length k. Hence M may ask the oracle only k different questions. This does not necessarily mean that the number of queries in every computation is $\leq k$ because the same question may occur several times. However, by providing M with an extra tape for bookkeeping queries and answers we obtain a polynomial time machine which needs to ask each different query at most once. This shows that $A \leq^P_{k\text{-}T} B$.

If $A \leq^P_{k\text{-}T} B$ via a machine M, then every computation tree $T(M,x)$ contains at most $2^k - 1$ queries. Let $f(x)$ denote a list of such queries. Then $f(x)$ can be computed in polynomial time $O(2^k p(|x|)$ by simulating each of the at most 2^k paths from the root to a leaf in $T(M,x)$. Hence $A \leq^P_{(2^k-1)\text{-}tt} B$. $\blacksquare$

3. k-question Turing reducibility and tally oracles

Now we are ready to generalize the original result of P. Berman [2] to the $\leq^P_{k-T}$-reducibility. Technically we follow, when appropriate, the exposition of [8]. The proof is based, besides on properties of $\leq^P_{k-T}$, on the following *self-reducibility structure* of satisfiable boolean formulas: the problem of deciding the satisfiability of a formula F reduces to problems of whether either of F_t and F_f are satisfiable, where F_t (F_f) is the result of setting the first variable in F to true (false) and simplifying. It is important that then $|F_t| \leq |F|$ and $|F_f| \leq |F|$.

Theorem 3. *If NP has a tally $\leq^P_{k-T}$-hard set, then $P=NP$.*

Proof. Let SAT be the set of satisfiable boolean formulas. Since SAT is $\leq^P_m$-complete for NP, it suffices to prove that if SAT$\leq^P_{k-T}B$ for some $B \subseteq 1^*$, then SAT$\in P$.

Suppose that $B \subseteq 1^*$ and SAT$\leq^P_{k-T}B$ via a machine M. Let F be a boolean formula whose satisfiability is to be decided. The self-reductions of F form a binary tree with F as the root and F_t and F_f, as defined above, as the left and right sons of the root; and so on. The leaves will simply be true or false. If F has m variables, then the tree will have $2^{m+1}-1$ nodes.

We perform a depth-first search of this tree and use properties of M and B to prune the search so much that the time requirement is only polynomial in $|F|$. The search will either find a satisfying asssignment or determine that none exists.

At every node G encountered during the search we simulate all the computations presented by the tree $T(M,G)$. During the simulation we form a list $l=(l_1,l_2, \ldots, l_p)$ such that the list has an element l_i for each computation found in $T(M,G)$. Since $T(M,G)$ contains at most 2^k computation paths from the root to acceptance or rejection, we have $p \leq 2^k$. Each l_i is of the form $l_i=((q_1,a_1),(q_2,a_2), \ldots, (q_r,a_r))$. Here $q_j \in 1^*$ is the jth query in the computation represented by l_i, and $a_j=1$ or 0 depending on whether the computation takes after the jth query the YES branch or the NO branch. Since M is a k-query machine, we have $r \leq k$. We label node G with $l=l(G)$. All this can be accomplished in time $O(2^k p(|G|))$ where p is the polynomial bounding the time requirement of M. Since $|G| \leq |F|$ for every formula G in the self-reducibility tree of F, we obtain $O(2^k p(|G|)) \leq O(2^k p(|F|))$. Thus the search needs a polynomial time at each node.

For a node G, its label $l(G)$ is called a *reject label*, if we know that G is not in SAT. During the search we can infer that certain labels are reject labels as follows:

(1) l(false) is clearly a reject label;

(2) if $l(G_f)$ and $l(G_t)$ are reject labels, then $l(G)$ is also a reject label since then G_f as well as G_t are not in SAT which means that G cannot be in SAT.

The search is pruned by not searching below a node whose label is already known to be a reject label. Observe that if $l(G)$ is a reject label by rule 1 or 2 then all G' such that $l(G')=l(G)$ have a reject label, of course. This conclusion is correct only if we can now infer that every such G' is not in SAT. Because G is not in SAT, machine M with oracle B must follow some reject path of $T(M,G)$. Let $((q_1,a_1),(q_2,a_2), \ldots ,(q_r,a_r))$ be the encoding of this path in the label $l(G)$. Thus each $q_i \in B$ if and only if $a_i=1$. Since $l(G)=l(G')$, a reject path with the same encoding occurs in $T(M,G')$. With orcle B, machine M must follow this path and will reject G'. Thus G' is not in SAT.

The search stops when either a leaf with formula "true" is found or when $l(F)$ is found to be a reject label. In the former case the path from the root to the "true" leaf indicates a satisfying assignment. In the latter case F cannot be satisfiable.

To complete the proof we must show that the outlined searching algorithm runs in polynomial time. The following lemma establishes this.

Lemma 4. *Let F be a formula with m variables and let p be a polynomial bound of the running time of M. Then the algorithm above visits at most $m+m\times(2(p(|F|)+1))^{k\cdot 2^k}$ interior nodes of the self-reducibility tree for F and therefore runs in polynomial time.*

Proof. If G and G' are two unsatisfiable formulas with the same label (i.e.

$l(G)=l(G')$) occurring in the interior of the pruned search tree, then they must be on the same branch from the root. Otherwise, one of the formulas, say G, would be searched first, and its label $l(G)$ would be determined to be a reject label. But then the depth-first search would not go below G', contradicting the assumption that G' is not a leaf.

Thus the number of distinct paths from the root to unsatisfiable interior nodes is bounded by the number of distinct reject labels. This number surely is at most equal to the number of all possible labels. This in turn equals $(2(p(|F|)+1))^{k \cdot 2^k}$ because each label $l(G)$ is of the form $(l_1, \ldots, l_p)$, where $p \leq 2^k$ and each l_i is of the form $((q_1,a_1),(q_2,a_2), \ldots, (q_r,a_r))$ where $r \leq k$ and each $q_j \in 1^*$ is of length at most $p(|G|) \leq p(|F|)$ and each a_j equals 0 or 1. Since the tree has height m, there are at most $m \times (2(p(|F|)+1))^{k \cdot 2^k}$ interior nodes with reject labels. A satisfying assignment visits at most another m nodes. ▪

The method presented in the proof of Theorem 3 above for deciding the satisfiability of a boolean formula can easily be adapted for deciding in polynomial time the validity of a quantified boolean formula, c.f. [3]. Since the set of valid quantified boolean formulas is $\leq_m^P$-complete for *PSPACE*, we obtain:

Theorem 5. *If PSPACE has a tally* $\leq_{k-T}^P$*-hard set, then P=PSPACE.* ▪

Moreover, the generalization of Fortune's [3] results given by Meyer and Paterson [9] applies also to our proof of Theorem 3. Thus every language which has self-reducibility property in the precise sense defined in [9] and which $\leq_{k-T}^P$-reduces to a tally language, can be recognized in polynomial time.

4. Conjunctive reducibility and sparse oracles

In conjunctive and disjunctive reductions a Turing machine uses its oracle in a very limited way. Therefore for these reductions we may obtain a result similar to Theorem 3 without a restriction to k-question machines. Also the restriction to tally oracles can be relaxed. It suffices to assume that the oracle is *sparse*, that is, there is a polynomial q such that the number of elements in the oracle of length at most n is at most $q(n)$.

Theorem 6. *If co-NP has a sparse $\leq_c^P$-hard set (or equivalently, NP has a co-sparse $\leq_d^P$-hard set), then P=NP.*

Proof. We first note that because $A \leq_c^P B$ if and only if $\bar{A} \leq_d^P \bar{B}$, the two alternative premisses of the theorem really are equivalent.

The set of non-tautological boolean formulas, $\overline{\text{SAT}}$, is $\leq_m^P$-complete for $co-NP$. Hence it suffices to prove that if $\overline{\text{SAT}} \leq_c^P B$, where $B \subset \Sigma^*$ is sparse, then $\overline{\text{SAT}} \in P$. Then also SAT$\in P$ which means $P=NP$, as required.

Let $\overline{\text{SAT}} \leq_c^P B$ via a Turing machine M whose only reject state is the NO state and whose running time is bounded by a polynomial p. Let F be a boolean formula whose satisfiability is to be decided. We again perform a similar depth-first search in the tree of self-reductions of F as in the proof of Theorem 3. The labeling function l must be modified as follows. The label of a node corresponding to a formula G, $l(G)$, equals the list of all queries occuring in the tree $T(M,G)$. The list is easy to form because it is the sequence of queries on the (rightmost) path on which each query to the oracle is answered "yes". So $l(G)$ is of the form $l(G)=(q_1, \ldots, q_r)$ where each $q_i \in \Sigma^{p(|F|)}$ and $r \leq p(|G|) \leq p(|F|)$. Hence $l(G)$ can be computed in time $O(p(|F|))$.

As in the proof of Theorem 3, a label $l(G)$ is called a reject label if we know that $G \in \overline{\text{SAT}}$. (Actually, machine M will accept such a formula G, but we prefer to stick to the old terminology.) Now we have three possibilities (1)-(3) below to infer that certain labels are reject labels ((1) and (2) are as in the proof of Theorem 3):

(1) l(false) is a reject label;

(2) if $l(G_f)$ and $l(G_t)$ are reject labels, then $l(G)$ is also a reject label;

(3) if each element q_i in the label $l(G)=(q_1, \ldots, q_r)$ occurs in some reject label, then $l(G)$ is also a reject label.

If $l(G')$ is a reject label then M accepts G', which is possible only if M obtains the answer "yes" to every query it makes for input G'. So every element of $l(G')$ must be in the oracle set B. This implies that also the new rule (3) is correct. Suppose, namely, that $l(G)$ is to be a reject label by rule (3), i.e. that each element of $l(G)$ occurs in some other reject label found so far. Then we know (formally, by induction hypothesis) that every element of $l(G)$ must be in B. This implies that M must accept G, and hence $G \in \overline{\mathrm{SAT}}$.

Now the search procedure can be completed as in the proof of Theorem 3: the search is pruned by not searching below a node whose label is already known to be a reject label. In this way, either a satisfying assignment is found or $l(F)$ is found to be a reject label, in which case $F \in \overline{\mathrm{SAT}}$.

In the following lemma we show that the running time of the pruned search is polynomially bounded.

Lemma 7. *Let F be a formula with m variables. Let p be a polynomial bounding the running time of M and q a polynomial bounding the density of the sparse oracle set B. Then the algorithm above visits at most $m+m \times q(p(|F|))$ interior nodes of the self-reducibility tree for F and therefore runs in polynomial time.*

Proof. We will again show that the number of distinct paths from the root of the pruned search tree to an interior node is polynomially bounded. Because now the number of different labels of nodes does not necessarily have a polynomial bound, the proof of Lemma 5 must be modified.

Let t be a path in the pruned tree from the root to a leaf corresponding to an unsatisfiable formula. Suppose moreover that the interior nodes of t are not properly contained in the interior nodes of any other path. Let G be the last interior node of t. Then all sons of G must be leaves.

Consider the moment when the search reaches a node G. Denote by A the set of those strings in Σ^* which occur in some label known to be a reject label. Since G is an interior node, at least one element q of $l(G)$ must be outside A. Otherwise we could infer by rule (3) that $l(G)$ is a reject label and the depth-first search would not go below G, contradicting the assumption that G is not a leaf.

Hence the search goes below G. When returning back to G, $l(G)$ is determined to be a reject label. The current set A therefore will contain also q. So each path t must increase the size of A at least by one. On the other hand, A must always be a subset of the sparse oracle B. The length of each element of A is at most $p(|F|)$. There are at most $q(p(|F|))$ such elements in B. Thus

$q(p(|F|))$ is an upper bound for the number of distinct paths t. Then, since the tree has height m, there can be at most $m \times q(p(|F|))$ interior nodes with reject labels. A satisfying assignment again visits at most another m nodes. ▪

As for Theorem 3, we also in this case have the following related result:

Theorem 8. *If PSPACE has a sparse $\leq_c^P$-hard (or co-sparse $\leq_d^P$-hard) set, then P=PSPACE.* ▪

Moreover, the generalization in [9] applies to the proof of Theorem 6. Thus every language which has the self-reducibility property of [9] and the complement of which $\leq_c^P$-reduces to a sparse language, can be recognized in polynomial time.

References

[1] Berman,L. and J.Hartmanis, On isomorphisms and density of *NP* and other complete sets. SIAM J. Comp. 6,2 (1977), 305-322.

[2] Berman,P., Relationship between density and deterministic complexity of *NP*-complete languages. In: Automata, Languages and Programming (Fifth Int. Coll., Udine, Italy, July 1978), Lecture Notes in Computer Science, Vol. 62, pp. 63-71, Springer-Verlag, Berlin-Heidelberg-New York 1978.

[3] Fortune,S., A note on sparse complete sets. SIAM J. Comp. 8,3 (1979), 431-433.

[4] Garey,M.R. and D.S.Johnson, *Computers and Intractability.* Freeman, San Francisco 1979.

[5] Karp,R.M. and R.J.Lipton, Some connections between nonuniform and uniform complexity classes. Proc. 12th ACM Symp. on Theory of Computing (1980), 302-309.

[6] Ladner,R.E., N.A.Lynch and A.L.Selman, A comparison of polynomial time reducibilities. Theoretical Computer Science 1 (1975), 103-123.

[7] Long,T.J., A note on co-sparse polynomial time Turing complete sets for *NP*. Manuscript, Dept. of Computer and Information Science, The Ohio State University (November 1980).

[8] Mahaney,S.S., Sparse complete sets for *NP*: Solution of a conjecture of Berman and Hartmanis. Proc. 21st Ann. IEEE Symp. on Foundations of Computer Science (1980), 54-60.

[9] Meyer,A.R. and M.S.Paterson, With what frequency are apparently intractable problems difficult? MIT Technical Report, MIT/LCS/TM-126 (February 1979).

[10] Post,E.L., Recursively enumerable sets of integers and their decision problems. Bull. Amer. Math. Soc. 50 (1944), 284-316.

[11] Selman,A.L., Analogues of semirecursive sets and effective reducibilities to the study of *NP* complexity. Manuscript, Comp. Sci. Dept., Iowa State University (1981).