JOURNAL OF COMPUTER AND SYSTEM SCIENCES 36, 336–350 (1988)

Isomorphisms and 1-L Reductions*

ERIC W. ALLENDER

Department of Computer Science, Rutgers University, New Brunswick, New Jersey 08903

Received August 6, 1986; revised March 26, 1987

All sets complete for NP under 1-L reductions are complete under length-increasing, invertible, and "almost one–one" $\leq_m^p$ reductions. All sets complete for PSPACE under 1-L reductions are *p*-isomorphic. © 1988 Academic Press, Inc.

1. INTRODUCTION

Berman and Hartmanis showed in [5] that all of the most familiar NP-complete problems are *p*-isomorphic, and thus from the standpoint of complexity theory they can be thought of as simple reencodings of the same set. It was conjectured in [5] that in fact *all* NP-complete problems are *p*-isomorphic. In the intervening decade, the Berman–Hartmanis conjecture has provided the motivation for many investigations into the structure of complexity classes, including numerous papers dealing with sparse sets (e.g., [19]) and studies of *p*-isomorphism degrees [20, 23].

Joseph and Young reconsidered the Berman–Hartmanis conjecture in [16], and they presented heuristic evidence that the conjecture fails. More specifically, it was conjectured in [16] that there are NP-complete sets which are *not* *p*-isomorphic, assuming that one-way functions exist.

The Joseph–Young conjecture motivated an investigation of the class of sets complete for EXPTIME, since all sets complete for EXPTIME *are* *p*-isomorphic if one-way functions do not exist [4, 6, 25]. Interesting results in this setting have been reported in [25, 17, 18, 7].

None of the papers which have appeared since [5] have dealt with the problem of taking an NP-complete set **A** and *building* a *p*-isomorphism between **A** and SAT. If one could do this in general, then it would follow that the Berman–Hartmanis conjecture were true and hence that $P \neq NP$; thus such a program is likely to be difficult to carry out. However, if we assume that the NP-complete set **A** is complete under a *restricted* class of reductions, we may have a better chance of building the isomorphism. That is the problem considered in this paper.

* Portions of this research were carried out while the author was supported by NSF Grant MCS 81-03608.

0022-0000/88 $3.00

1-L reductions were defined in [8] and were considered again in [9, 13–15, 21, 22, 12]. Basically, a 1-L reduction is a function computed by a logspace-bounded Turing machine which has a one-way input tape; more complete definitions will be given in Section 2. The major reason for introducing such weak reductions is that finer distinctions can be made using 1-L reductions than, for instance, logspace reductions. Although 1-L reductions are not very powerful, it turns out that they are powerful enough for most practical applications; in [8] it was shown that most NP-complete problems which have appeared in the literature are complete for NP under 1-L reductions. Note, however, that it is easy to *construct* a set p-isomorphic to SAT which is not complete under 1-L reductions [8]. In [9] it is shown that no set complete under 1-L reductions can be sparse. We greatly improve on that result; we show that all such sets are "almost" p-isomorphic.

Independantly, Huynh [12] has shown that sets complete under 1-L reductions have exponential density. That result follows as a corollary of the main result presented here. (Huynh actually shows a somewhat stronger result, using "nonuniform" 1-L reductions.)

Let a function f be *poly–one* if, for all y in the range of f, $|f^{-1}(y)| = |y|^{O(1)}$. Note that a function is poly–one if it is "almost" one–one, in the sense that it does not map very many strings to any given string. Poly–one functions which have hard inverses are considered in [3]; here, we shall consider poly–one functions with easy inverses.

A function f is *strongly invertible* if there is a polynomial-time function which, on input y in the range of f, prints out all of the elements of $f^{-1}(y)$. Note that a strongly invertible function is necessarily poly–one.

THEOREM 3.1. *Let A be complete for NP (or DLOG, NLOG, P, NEXPTIME, etc.) under 1-L reductions. Then A is complete under length-increasing, strongly invertible $\leqslant_m^p$ reductions.*

One result of [5] states that any set which is complete for NP under length-increasing, *one–one*, invertible $\leqslant_m^p$ reductions is p-isomorphic to SAT. Thus Theorem 3.1 says that sets complete for NP under 1-L reductions are "almost" p-isomorphic.

Given a set L which is complete under *one–one* 1-L reductions (which are not known to be honest), Theorem 3.1 shows that L is complete under length-increasing, invertible functions which are not necessarily one–one. The next result eliminates this defect.

THEOREM 3.2. *Let A be complete for NP (or DLOG, NLOG, etc.) under one–one 1-L reductions. Then A is complete under one–one, length-increasing, invertible $\leqslant_m^p$ reductions.*

For deterministic complexity classes containing PSPACE or EXPTIME, the result of Theorem 3.2, together with the techniques of [4, 6, 25] yields a stronger result.

THEOREM 3.3. *All sets complete for PSPACE (EXPTIME, DTIME($2^{n^{O(1)}}$), etc.) under 1-L reductions are p-isomorphic.*

2. PRELIMINARIES

We assume familiarity with the usual notions of Turing machine and complexity classes such as DLOG, NLOG, P, and PSPACE. EXPTIME and NEXPTIME refer to DTIME($2^{O(n)}$) and NTIME($2^{O(n)}$), respectively. For background and definitions, consult [11].

The notation $|S|$ denotes the cardinality of the set S; $|w|$ denotes the length of the string w. The empty string is denoted by ε. We consider only strings over the alphabet $\Sigma = \{0, 1\}$. Occasionally, we may refer to strings in $\{0, 1, \#\}^*$; this should be viewed only as a notational convenience. Such strings should be viewed as being in $\{00, 01, 11\}^*$. We assume a standard lexicographical ordering on strings; we say $x \leqslant y$ if x comes before y in this ordering.

A $\leqslant_m^p$ reduction is a function f which is computable in time polynomial in the length of the input. We say $A \leqslant_m^p B$ via f if for all x, $x \in A \Leftrightarrow f(x) \in B$. A *logspace reduction* is a $\leqslant_m^p$ reduction which is computable in logspace. A *log-lin* reduction [24] is a logspace reduction f such that for all x, $|f(x)| = O(|x|)$. A reduction f is *honest* if there is a polynomial p such that for all x, $p(|f(x)|) > |x|$; f is *length-increasing* iff $|f(x)| > |x|$ for all x. We will call a $\leqslant_m^p$ reduction *invertible* if it is one–one and strongly invertible. A function f is a $\leqslant_{1,i}^{li}$ reduction if it is a length-increasing, invertible $\leqslant_m^p$ reduction. Two sets A and B are *p-isomorphic* if $A \leqslant_m^p B$ via an invertible bijection f.

The Berman–Hartmanis conjecture states that all NP-complete sets are p-isomorphic [5]. The following facts are frequently useful in work relating to p-isomorphism.

FACT 1 [5]. If $A \leqslant_{1,i}^{li} B$ and $B \leqslant_{1,i}^{li} A$, then A and B are p-isomorphic.

FACT 2 [5]. An NP-complete set A is p-isomorphic to SAT iff $A \times \Sigma^* \leqslant_{1,i}^{li} A$.

Sets A such that $A \times \Sigma^* \leqslant_{1,i}^{li} A$ are called p-cylinders in [6, 20] in analogy to a related notion in recursive function theory. A function p such that $A \times \Sigma^* \leqslant_{1,i}^{li} A$ via p is called a *padding function for A*. A nice formulation of results on p-isomorphism in terms of p-cylinders may found in [20].

A set S is *sparse* iff there is a polynomial p such that $p(n) \geqslant |\{x \in S \mid n \geqslant |x|\}|$. S is *p-printable* if the function which maps 1^n to an encoding of $\{x \in S \mid n \geqslant |x|\}$ is computable in time polynomial in n. Clearly, all p-printable sets are sparse. P-printable sets were first defined and studied in [10].

Theorems 3.1, 3.2, and 3.3 apply to a large number of common complexity classes. Let us say that a class of languages $\mathscr{C}$ is *suitable* if

(1) $\mathscr{C}$ is closed under log-lin reductions.

(2) If $L \in \mathscr{C}$, then $\{x^{2^{|x|}} \mid x \in L\} \in \mathscr{C}$.

(3) There is a p-cylinder which is complete for $\mathscr{C}$ under $\leqslant_m^p$ reductions.

It is easy to verify that all of the classes in the list (DLOG, NLOG, P, NP, PSPACE, EXPTIME, NEXPTIME, DTIME($2^{n^{O(1)}}$), ...) are suitable.

The definition of 1-L reduction is somewhat controversial. As defined in [8, 9], a 1-L reduction is a function computed by a Turing machine with a one-way read-only input tape with an endmarker, a one-way write-only output tape, and a read-write worktape with endmarkers, such that on inputs of length n, the worktape has $\lceil \log n \rceil$ cells between the endmarkers. Such a Turing machine is called a 1-L machine. Given a 1-L machine M, a *configuration* of M is a string encoding the worktape contents, the positions of the input, output, and worktape heads, and the finite state of M at a given point in a computation. The results in this paper are proved using this definition of 1-L reduction. The problem with the definition is pointed out by the following theorem, which contradicts Proposition 1.3 in [8].

THEOREM 2.1. *The class of* 1-*L reductions is not closed under composition.*

Proof. Let $f(x) = 1x$ if $\lceil \log |x| \rceil$ is even, and $f(z) = 0x$ otherwise, and let $g(x) = x1^{|x|}$ if the last bit of x is 1, and $g(x) = x$ otherwise. It is easy to see that f and g are both 1-L reductions. However, we now claim that $h(x) = f(g(x))$ is *not* a 1-L reduction.

Assume that M is a 1-L machine computing h. First note that M cannot produce any output until it has scanned the entire input, since for all strings x, the first bit of $h(x0)$ is different from the first bit of $h(x1)$. (If $\lceil \log |x0| \rceil$ is odd, then $h(x0) = 0x0$ and $h(x1) = 1x11^{|x1|}$. If $\lceil \log |x0| \rceil$ is even, then $h(x0) = 1x0$ and $h(x1) = 0x11^{|x1|}$.)

M has only polynomially many worktape configurations; thus there must be some length n such that the number of worktape configurations of M on strings of length n is less than 2^n. Thus if we associate with each string z of length n the configuration C_z which M enters when it first scans the final symbol of x, it follows that there are two strings u and v of length n such that $C_u = C_v$, and thus by the observations in the previous paragraph, M produces the same output on input $u0$ as on input $v0$. Thus M does not compute h, contrary to assumption. ∎

It should be noted that an earlier version of this paper, [2], contained a proof of Theorem 3.3 which relied on closure of 1-L reductions under composition. The proof presented in the current paper does not suffer from that defect.

Since it is desirable for a class of reductions to be closed under composition, one is tempted to modify the definition in such a way that closure under composition holds. This is easily done. Define a 1-L′ reduction to be a function computed by a Turing machine with a one-way read-only input tape with endmarker, a one-way write-only output tape, and a read-write worktape which is initially set to all blanks, such that for all inputs x, at most $\lceil \log |x| \rceil$ cells of the worktape are ever

visited. Note that this corresponds to the usual notion of on-line space-bounded computation. The following proposition is easily proved.

PROPOSITION 2.2. *The class of 1-L′ reductions is closed under composition.*

It is also easily seen that SAT and many other NP-complete sets are complete under 1-L′ reductions. The same is true for the standard sets complete for PSPACE. Unfortunately, much of the motivation for 1-L reductions comes from small complexity classes such as DLOG and NLOG; for instance, the graph accessibility problem GAP is complete for NLOG under 1-L reductions, and the so-called deterministic graph accessibility problem GAP1 is complete for DLOG under 1-L reductions [8]. It is not at all clear that there is a nice encoding for these problems such that they would be complete for the appropriate class under 1-L reductions.

Although we feel that it is worthwhile to resolve the difficulties with the definition of 1-L reduction, for the purposes of this paper it will suffice to use the definition which makes 1-L reductions as powerful as possible, since that has the result of strengthening the theorems proved here.

3. MAIN RESULTS

THEOREM 3.1. *Let $\mathscr{C}$ be any suitable class of languages. If A is complete for $\mathscr{C}$ under 1-L reductions, then A is complete for $\mathscr{C}$ under length-increasing, strongly-invertible $\leqslant_m^p$ reductions.*

In particular, $\mathscr{C}$ can be any of the natural classes DLOG, NLOG, P, NP, etc. For clarity, we will assume that $\mathscr{C} = \mathrm{NP}$ throughout the proof. The proof consists of a series of lemmas.

LEMMA 3.1. *There is a set SAT′ p-isomorphic to SAT such that $w \in SAT' \Rightarrow |w|$ is a power of 2 and $|w| > 1$.*

Proof. Let $\mathrm{SAT}' = \{x10^{r-1} \mid r = 2^{\lceil \log |x| \rceil + 1} - |x| \text{ and } x \in \mathrm{SAT}\}$. Clearly, $\mathrm{SAT} \leqslant_{1,i}^{li} \mathrm{SAT}'$, and thus SAT and SAT′ are p-isomorphic. ∎

In what follows, let A be a given set which is complete for NP under 1-L reductions, let $B = \{x^{2^{|x|}} \mid x \in \mathrm{SAT}'\}$, and let f be a 1-L reduction computed by a machine M, where f reduces B to A.

LEMMA 3.2. *There is a p-printable set S such that if $x \in SAT'$ and $x \notin S$, then M on input $x^{2^{|x|}}$ produces at least one bit of output while reading each copy of x. So, in particular, S contains all $x \in SAT'$ such that $|f(x^{2^{|x|}})| \leqslant |x|$.*

Proof. On input n, the following routine prints a list containing all $x \in \mathrm{SAT}'$ of length at most n such that M, on input $x^{2^{|x|}}$, fails to produce any output while processing some copy of x.

begin
for $m := 1$ to $\lceil \log n \rceil$ //Print all such x with $|x| = 2^m$.
(1) Create a labelled digraph $G = (V, E)$ with V being the set of all configurations of M of size $1 + 2m$ $(= \log |x^{2|x|}|$, if $|x| = 2^m)$, and E containing an edge labelled $a \in \Sigma \cup \{\varepsilon\}$ from C_i to C_j iff M has a move $C_i \vdash C_j$ which consumes input a and produces no output. (The label l_p of a path p in G is the concatenation of the labels of its edges.)
(2) **for** each configuration $C_i \in V$
Make a copy G_i of G
By doing a breadth-first search of G_i starting at C_i, find and mark those edges which can be traversed by a path p from C_i, where $|l_p| \leqslant 2^m$.
Delete all unmarked edges and all vertices which are not connected to C_i by a marked path.
for each C_j in G_i
if there are two paths from C_i to C_j, mark C_j
Delete all marked vertices.
//G_i is now a tree, since for every C_j in G_i, there is exactly one path from C_i to C_j.
for each C_j in G_i
if the path p from C_i to C_j has $|l_p| = 2^m$, put l_p in TEMP
for each $x \in$ TEMP
output x.
end

To see that the routine is correct, let x be any word in SAT′ such that there exists some $r < 2|x|$ such that after reading x^r, M produces no output while reading the $r+1$th x. We need to show that the routine outputs x on input $n \geqslant |x|$.

Let C_i be the configuration M enters after consuming x^r, and let C_j be the configuration M enters after consuming x^{r+1}.

Let us assume that the routine does not output x. Then there must be some path p in G_i from C_i to C_j, with $l_p = w \neq x$. Note that without loss of generality, $|w| \leqslant |x|$, since all edges corresponding to words of length $> |x|$ are deleted before we check for duplicate paths.

Consider M's computation on input $x^r w x^{2|x|-r-1}$. In order to determine M's initial configuration, we must calculate $\lceil \log |x^r w x^{2|x|-r-1}| \rceil$. Since $x \in$ SAT′, $|x| = 2^m$ for some $m > 0$. Thus

$$\begin{aligned} 2m+1 = \lceil \log 2^m(2^{m+1}-1) \rceil &= \lceil \log 2^{2m+1} - 2^m \rceil \\ &= \lceil \log 2|x|^2 - |x| \rceil \\ &\leqslant \lceil \log 2|x|^2 - |x| + |w| \rceil \\ &\leqslant \lceil \log 2|x|^2 \rceil = 2m+1. \end{aligned}$$

That is, M's initial configuration is the same on input $x^r w x^{2|x|-r-1}$ as on input

$x^{2|x|}$. Thus on input $x^r w x^{2|x|-r-1}$, M enters configuration C_i after reading x^r, enters C_j after reading w without producing any output while reading w, and then finishes the computation, reading $x^{2|x|-r-1}$. Thus M produces the same output on input $x^r w x^{2|x|-r-1}$ as on input $x^{2|x|}$; thus $f(x^r w x^{2|x|-r-1}) = f(x^{2|x|})$ and $x^r w x^{2|x|-r-1} \in B$, since $x^{2|x|} \in B$. Thus $x^r w x^{2|x|-r-1} = y^{2|y|}$ for some $y \in \mathrm{SAT}'$. Since $|w| \leqslant |x|$, we must have $|y| \leqslant |x|$. However, $|y| = |x|$ implies $y = z = w$, which contradicts $x \neq w$. Thus $|y| < |x|$. However, $|y| < |x|$ implies $|y| \leqslant 2^{m-1}$, which implies $|y^{2|y|}| \leqslant 2^{2m-1} < 2^{2m+1} - 2^m \leqslant |x^r w x^{2|x|-r-1}|$, which contradicts $x^r w x^{2|x|-r-1} = y^{2|y|}$. ∎

LEMMA 3.3. *There is a polynomial q_1 such that $y \in A \Rightarrow |f^{-1}(y) \cap \Sigma^n| \leqslant q_1(n)$ for all n.*

Proof. Note that since $y \in A$, we have that $f^{-1}(y) \subseteq B$. We thus have $|f^{-1}(y) \cap \Sigma^n| = 0 \leqslant q_1(n)$ unless $n = 2m^2$, where m is some power of two. If $n = 2m^2$, then we may write $f^{-1}((y)) = \{x_1^{2m}, x_2^{2m}, \ldots, x_r^{2m}\}$ for some $r \geqslant 0$.

M, in its computation on x_1^{2m}, reaches a point when it has consumed the first x_1 of the input, has output some prefix y' of y, and is in some configuration C_1.

Since $f(x_i^{2m}) = y$ for all i, $1 \leqslant i \leqslant r$, M reaches a point in its computation on each x_i^{2m} when it has scanned some prefix x_i' of x_i^{2m}, has output y', and is in some configuration C_i. If $C_i = C_j$, then

(1) $|x_i'| = |x_j'|$, since a configuration of M records the position of the input head.

(2) M outputs y on input $x_i' x_j''$, where $x_j^{2m} = x_j' x_j''$. Note that $|x_i' x_j''| = n$.

(3) $x_i' x_j'' \in B$, and thus $x_i' x_j'' = x_k^{2m}$ for some k, $1 \leqslant k \leqslant r$.

Since $|x_k| = m \leqslant |x_k^{2m}|/2 = |x_i' x_j''|/2$, x_k is either a prefix of x_i', a suffix of x_j'', or both. Thus $i = k = j$. That is, $C_i = C_j \Rightarrow i = j$.

Thus r is less than or equal to the number of configurations of M on inputs of length n. Since M is a logspace-bounded machine, this number can be bounded by some polynomial q_1. ∎

LEMMA 3.4. *Let* $\mathrm{preimage}(y) = \{x \mid f(x) = y$ *and* $|x| \leqslant |y|\}$. *Then* $|\mathrm{preimage}(y)|$ *can be computed in time polynomial in* $|y|$, *and* $\mathrm{preimage}(y)$ *can be computed in time polynomial in* $(|y| + |\mathrm{preimage}(y)|)$.

Proof. Let M' be a logspace-bounded nondeterministic Turing machine which, on input y, will guess a string x of length no greater than $|y|$ and check that $f(x) = y$. (Since f is a 1-L reduction, the bits of x can be guessed one at a time, and thus it is not necessary to store all of x; thus logspace is sufficient.) Clearly, such a machine M' can be constructed such that there is a one-to-one correspondence between accepting computations of M' and elements of $\mathrm{preimage}(y)$.

In order to compute $|\mathrm{preimage}(y)|$, first build the digraph $G = (V, E)$ with configurations of M' as vertices, and edges representing the $\vdash$ relation. Without loss of generality, assume that G contains no cycles (i.e., assume that M' keeps track of the number of steps it has executed). The following simple routine com-

putes, for each C in V, the number of accepting computations of M' starting from C:

begin
  repeat
    for all C in V
      if C is a rejecting configuration
        then count(C) := 0
      if C is an accepting configuration
        then count(C) := 1
      if C is not a halting configuration
          and count(D) is defined for all successors of C
        then count(C) := sum({count(D) | D is a successor of C})
  until count(C_{init}) is computed, where C_{init} is the initial configuration
end

It is clear that count(C_{init}) is |preimage(y)|, and that the computation can be performed in polynomial time.

In order to enumerate preimage(y), it suffices to enumerate all paths in G from C_{init} to accepting configurations. Each such path p corresponds to an element $x_p \in$ preimage(y). Computing x_p given p is straightforward. The entire computation can be done in time polynomial in ($|y| + |\text{preimage}(y)|$). ∎

Remark. Nowhere in the proof of Lemma 3.4 is use made of the fact that f is a reduction of B to A. Thus *any* poly-one length-increasing 1-L reduction is strongly invertible. See also [1] for more general results.

By Lemma 3.3, there is a polynomial q such that if $x \in \text{SAT}'$, then $|\text{preimage}(f(x^{2^{|x|}}))| \leqslant q(|x|)$.

We are now ready to define a procedure which computes a function g which we claim is a strongly invertible, length-increasing, $\leqslant_m^p$ reduction of SAT′ to A.

Since SAT′ is p-isomorphic to SAT, there is a one–one, length-increasing, invertible padding function p such that $p(x, y) \in \text{SAT}'$ iff $x \in \text{SAT}'$. Let T be some fixed element of SAT′. Let $a(x)$ be the string which differs from $x^{2^{|x|}}$ only in the rightmost bit. Let rejectable(x) and trash(x) be defined by the following procedures.

rejectable(x)
begin
  if $|x|$ is not a power of two greater than 1 **or**
    $x \notin S$ and $|f(x^{2^{|x|}})| \leqslant |x|$ **or**
    $|\text{preimage}(f(x^{2^{|x|}}))| > q(|x|)$ **or**
    [$|\text{preimage}(f(x^{2^{|x|}}))| \leqslant q(|x|)$ and
      there is some element of preimage($f(x^{2^{|x|}})$) which is not of the form $y^{2^{|y|}}$]
    then return true
    else return false
end

```
trash(x)
begin
  let y be the least such that p(T, x # y) ∉ S
  return f(a(p(T, x # y)))
end
```

The function g is computed by the following routine.

```
g(x)
begin
  if rejectable(x)
    then
      g(x) = trash(x)
    else
      if x ∉ S
        then
          g(x) = f(x^{2^{|x|}})
        else
          let y be the last such that p(x, y) ∉ S
          if rejectable(p(x, y))
            then
              g(x) = trash(x)
            else
              g(x) = f(p(x, y)^{2^{|p(x,y)|}})
end
```

LEMMA 3.5. *The routine presented above computes g in polynomial time.*

Proof. This is immediate from Lemma 3.4, except for verifying that the operation "let y be the least such that $p(x, y) \notin S$" can be computed in polynomial time. Since S is sparse, at most $|x|^{O(1)}$ elements of $p(x, \Sigma^*)$ need to be examined in order to find some y such that $p(x, y) \notin S$. ∎

LEMMA 3.6. *g is a length-increasing reduction of SAT′ to A.*

Proof. If x is in SAT′, then rejectable(x) is false, and $x^{2^{|x|}} \in B$, and $\{f(x^{2^{|x|}}), f(p(x, y)^{2^{|p(x,y)|}}) \mid y \in \Sigma^*\} \subseteq A$. If, in addition, $x \notin S$, then $|g(x)| = |f(x^{2^{|x|}})| > |x|$. If $x \in S$ and y is the least such that $p(x, y) \notin S$ then $|g(x)| = |f(p(x, y)^{2^{|p(x,y)|}})| > |p(x, y)| > |x|$.

If x is not in SAT′ and rejectable(x) is false, then since x is not in SAT′, $x^{2^{|x|}} \notin B$, and $p(x, y)^{2^{|p(x,y)|}} \notin B$ for all y. That g is length increasing and $g(x) \notin A$ follows by an argument similar to that in the preceding paragraph.

If x is not in SAT′, then if rejectable(x) is true, $g(x) = f(a(p(T, x \# y)))$, which is not in A since $a(p(T, x \# y))$ is not of the form $z^{2^{|z|}}$, and hence is not in B. Since $p(T, x \# y)^{2^{|p(T,x\#y)|}} \in B$, and $p(T, x \# y) \notin S$, M outputs at least one bit while

processing each copy of $p(T, x \# y)$ on input $p(T, x \# y)^{2^{|p(T,x\#y)|}}$. Since $a(p(T, x \# y))$ differs from $p(T, x \# y)^{2^{|p(T,x\#y)|}}$ only in the rightmost bit, it follows that $|g(x)| = |f(a(p(T, x \# y)))| > |p(T, x \# y)| \geq |x|$. ∎

LEMMA 3.7. *g is strongly invertible.*

Proof. The following procedure computes the strong inverse of g.

```
inverse(y)
begin
  if |preimage(y)| ≤ q(y)
    then
      compute preimage(y)
      for each w in preimage(y)
        if w is of the form x^(2^|x|) and g(x) = y
          then put x in LIST
        if w is of the form p(x, u)^(2^|p(x,u)|) and g(x) = y
          then put x in LIST
        if w is of the form a(p(T, x # u)) and g(x) = y
          then put x in LIST
    else
      for each prefix y' of y
        for each configuration C of M of length ≤⌈log |y|⌉
          for each s ∈ {0, 1}
            let y_{C,s} be the string which M outputs when in configuration C with s
            remaining on the input tape
            if |preimage(y'y_{C,s})| ≤ q(|y'y_{C,s}|)
              then
                compute preimage(y'y_{C,s})
                for each w in preimage(y'y_{C,s})
                  if w is of the form p(T, x # u)^(2^|p(T,x#u)|) and g(x) = y
                    then put x in LIST
  output LIST
end
```

To see that the routine is correct, note that if $g(x) = y$, then either $[g(x) = f(x^{2^{|x|}})$ and $|\text{preimage}(f(x^{2^{|x|}}))| \leq q(|x|) \leq q(|y|)]$ or $[g(x) = f(p(x, u)^{2^{|p(x,u)|}})$ and $|\text{preimage}(g(x))| \leq q(|x|) \leq q(|y|)]$ or $[g(x) = f(a(p(T, x \# u)))]$. Clearly, the only difficulty arises in the case $g(x) = f(a(p(T, x \# u)))$.

If $g(x) = f(a(p(T, x \# u)))$, then $g(x) = y'z$, where M outputs y' before reading the final input character. The routine tries all prefices y' of y and generates all strings z which M could possibly affix to y', and then checks to see if $p(T, x \# u)^{2^{|p(T,x\#u)|}} \in \text{preimage}(y'z)$ for any u.

It is clear that the running time is polynomial. ∎

Proof (of Theorem 3.1). Let A be any set complete for NP under 1-L reductions, and let C be any set in NP; we need to show that there is a strongly invertible, length-increasing $\leqslant_m^p$ reduction from C to A. Since SAT′ is complete for NP under $\leqslant_{1,i}^{li}$ reductions, there is a $\leqslant_{1,i}^{li}$ reduction h from C to SAT′. Lemmas 3.1 through 3.7 show that there is a strongly invertible, length-increasing $\leqslant_m^p$ reduction g which reduces SAT′ to A. The function $g \circ h$ is the desired reduction from C to A. ∎

The next result shows that if f is one–one, then we can transform it into a $\leqslant_{1,i}^{li}$ reduction.

THEOREM 3.2. *Let $\mathscr{C}$ be as in the proof of Theorem* 3.1. *If A is complete for $\mathscr{C}$ under one–one* 1-*L reductions, then A is complete for $\mathscr{C}$ under $\leqslant_{1,i}^{li}$ reductions. Hence all sets complete for $\mathscr{C}$ under one–one* 1-*L reductions are p-isomorphic.*

Proof. The function g constructed in the proof of Theorem 3.1 fails to be one–one, since for certain strings x and y we can have $g(x) = g(p(x, y)) = f(p(x, y)^{2^{|p(x,y)|}})$. The function g' computed by the following routine avoids such behavior:

```
g′(x)
begin
  if rejectable p(x, 0)
    then
      g′(x) = trash(x)
    else
      if p(x, 0) ∉ S
        then
          g′(x) = f(p(x, 0)^{2^{|p(x,0)|}})
        else
          let y be the least such that p(p(x, y), 1) ∉ S
          if rejectable(p(p(x, y), 1))
            then
              g′(x) = trash(x)
            else
              g′(x) = f(p(p(x, y), 1)^{2^{|p(p(x,y),1)|}})
end
```

For all strings x, either $g'(x) = f(a(p(T, x \# y)))$, $g'(x) = f(p(x, 0)^{2^{|p(x,0)|}})$, or $g'(x) = f(p(p(x, y), 1)^{2^{|p(p(x,y),1)|}})$ for some y. Since f is one–one, and since for all $z \neq x$ and all y_1, y_2, y_3, and y_4, $\{a(p(T, x \# y_1)), p(x, 0)^{2^{|p(x,0)|}}, p(p(x, y_2), 1)^{2^{|p(p(x,y_2),1)|}}\} \cap \{a(p(T, z \# y_3)), p(z, 0)^{2^{|p(z,0)|}}, p(p(z, y_4), 1)^{2^{|p(p(z,y_4),1)|}}\} = \varnothing$, g' is one–one. The proof that g' is length-increasing and invertible is the same as in the proof of Theorem 3.1. ∎

For larger complexity classes, the techniques of [4] (see also [6, 25]), yield a stronger result.

THEOREM 3.3. *Let $\mathscr{C}$ be any suitable deterministic time- or space-complexity class which contains PSPACE or EXPTIME. All sets complete for $\mathscr{C}$ under 1-L reductions are p-isomorphic.*

Proof. We prove only that all sets complete for PSPACE under 1-L reductions are p-isomorphic. The results for other deterministic classes can be proved similarly. This proof is based on the techniques of [25].

Let $M_1, M_2, \ldots$ be an indexing of all 1-L machines, and let A be any set complete for PSPACE under 1-L reductions. Let QBF be the set of all true quantified Boolean formulae; QBF is complete for PSPACE under one–one 1-L reductions [8].

Consider the set S accepted by a Turing machine M which performs the following computation:

begin
On input z of length n, mark off n^2 space.
if z is not of the form $i \# x \# y10^l$, halt and reject.
Run M_i on input z. (Do not store the output of M_i, but do record how much output M_i produces on input z.) If more than n^2 space is required, halt and reject.
if $|M_i(z)| \leqslant |z|$
  then
    accept z iff $M_i(z) \notin A$
    (Note that this step requires only polynomial space, since it involves checking if a string of length less than $|z|$ is in A.)
else for all $u \# v$ such that $u \# v \leqslant x \# y$
  Run M_i on input $i \# u \# v10^l$. If more than n^2 space is required, halt and reject. Compare the output of M_i on input $i \# u \# v10^l$ with the output of M_i on input $i \# x \# y10^l$. (This comparison can be done bit-by-bit, so that the entire output does not need to be stored all at one time.)
  if the output of M_i on input $i \# u \# v10^l =$ the output of M_i on input $i \# x \# y10^l$,
    then
      Halt and accept iff $u \notin$ QBF
endfor
(If the computation reaches this point, there is no $u \# v \leqslant x \# y$ such that the output of M_i on input $i \# u \# v10^l =$ the output of M_i on input $i \# x \# y10^l$.)
Halt and accept iff $x \in$ QBF
end

Clearly, $S \in$ PSPACE. Thus there is some 1-L reduction computed by some machine M_i reducing S to A. There is some constant r such that for all strings x

and y, M can carry out the simulation of M_i on input $i \# x \# y \# 10^r$ in n^2 space. Note that for all z of the form $i \# x \# y \# 10^r$, it must be that $|M_i(z)| > |z|$, since otherwise the diagonalization assures that M_i does not compute a reduction of S to A. If $w =$ the output of M_i on input $i \# x \# y \# 10^r =$ the output of M_i on input $i \# u \# v \# 10^r$ for some $u \# v \neq x \# y$, then let $u \# v \leqslant x \# y$ be the lexicographically smallest two strings which map to w in that way. Then $w \in A$ iff $i \# u \# v \# 10^r \in S$ iff $u \in$ QBF, and $w \in A$ iff $i \# x \# y \# 10^r \in S$ iff $u \notin$ QBF. This is a contradiction. Thus the function f which takes $x \# y$ to the output of M_i on input $i \# x \# y \# 10^r$ is a one–one, length-increasing function. Also, f is computable by a 1-L machine. Furthermore, $i \# x \# y \# 10^r \in S$ iff $x \in$ QBF, and thus $f(x \# y) \in A$ iff $i \# x \# y \# 10^r \in S$ iff $x \in$ QBF. That is, f is a one–one, length-increasing 1-L reduction from QBF $\# \Sigma^*$ to A. Since by the remarks after Lemma 3.4, all one–one length-increasing 1-L reductions are easy to invert, it follows that f is a $\leqslant_{1,i}^{li}$ reduction from QBF $\# \Sigma^*$ to A, and thus A is p-isomorphic to QBF. ∎

4. Observations and Questions

Let us say that the Berman–Hartmanis conjecture is *very false* if there are sets complete for NP under 1-L reductions which are not p-isomorphic. It is known that if the Berman–Hartmanis conjecture is true, then $P \neq NP$ (since if $P = NP$ then finite sets are NP-complete). It follows from Theorem 3.3 that if the Berman–Hartmanis conjecture is very false, then $NP \neq PSPACE$. We refrain from conjecturing that the Berman–Hartmanis conjecture is very false.

In [16], Joseph and Young posed the question of whether the Berman–Hartmanis conjecture is true iff one–one one-way functions do not exist. A possible first step toward answering this question would be to show that all sets complete for NP under length-increasing strongly invertible reductions are p-isomorphic. We have been unable to show that this is true, even with the additional assumption that the reductions under consideration never map more than two different strings to the same output.

The techniques used in this paper are very specific to 1-L reductions. For instance, consider sets which are complete for DTIME($2^{O(n)}$) under two-way DFA transductions. (A number of such sets are presented in [24].) It follows from the results of, e.g., [25], that all such sets are p-isomorphic. However, nothing is known about sets complete for NTIME($2^{O(n)}$) under two-way DFA reductions; it is not even known if such reductions can be replaced by one–one or length-increasing reductions, although it follows from results in [1] that all such reductions are easy to invert.

ACKNOWLEDGMENTS

Stimulating discussions with the following people influenced the direction in which this work developed: my thesis advisor: Kim King, Juris Hartmanis, Deborah Joseph, Steve Mahaney, Osamu Watanabe, and Paul Young. This paper also benefited from the careful scrutiny and helpful comments of an anonymous referee.

REFERENCES

1. E. W. ALLENDER, "Invertible Functions," Ph.D. dissertation, Georgia Institute of Technology, 1985.
2. E. W. ALLENDER, Isomorphisms and 1-L reductions, *in* "Proceedings, Structure in Complexity Theory Conference," Lecture Notes in Computer Science Vol. 223, pp. 12–22, Springer-Verlag, Berlin/New York, 1986.
3. E. W. ALLENDER AND R. RUBINSTEIN, P-printable sets, submitted for publication. See also E. W. Allender, The complexity of sparse sets in P, *op. cit.* , pp. 1–11.
4. L. BERMAN, "Polynomial Reducibilities and Complete Sets," Ph.D. dissertation, Cornell University, 1977.
5. L. BERMAN AND J. HARTMANIS, On isomorphisms and density of NP and other complete sets, *SIAM J. Comput.* **6** (1977), 305–323.
6. M. DOWD, "Isomorphism of Complete Sets," Technical Report LCSR-TR-34, Rutgers University, 1982.
7. J. GOLDSMITH AND D. JOSEPH, Three results on the polynomial isomorphism of complete sets, *in* "Proceedings, 27th IEEE Symposium on Foundations of Computer Science, 1986," pp. 390–397.
8. J. HARTMANIS, N. IMMERMAN, AND S. MAHANEY, One-way log-tape reductions, *in* "Proceedings, 19th IEEE Symposium on Foundations of Computer Science, 1978," pp. 65–72.
9. J. HARTMANIS AND S. MAHANEY, Languages simultaneously complete for one-way and two-way log-tape automata, *SIAM J. Comput.* **10** (1981), 383–390.
10. J. HARTMANIS AND Y. YESHA, Computation times of NP sets of different densities, *Theoret. Comput. Sci.* **34** (1984), 17–32.
11. J. E. HOPCROFT AND J. D. ULLMAN, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, MA, 1979.
12. D. T. HUYNH, Non-uniform complexity and the randomness of certain complete languages, Technical Report TR 85-34, Computer Science Department, Iowa State University, 1985.
13. N. IMMERMAN, Number of quantifiers is better than number of tape cells, *J. Comput. System Sci.* **22** (1981), 384–406.
14. N. IMMERMAN, Upper and lower bounds for first order expressibility, *J. Comput. System Sci.* **25** (1982), 76–98.
15. N. IMMERMAN, Languages with capture complexity classes, *SIAM J. Comput.* **16** (1987), 760–778.
16. D. JOSEPH AND P. YOUNG, Some remarks on witness functions for non-polynomial and non-complete sets in NP, *Theoret. Comput. Sci.* **39** (1985), 225–237.
17. K.-I. KO, T. J. LONG, AND D.-Z. DU, A note on one-way functions and polynomial-time isomorphisms, *in* "Proceedings, 18th Annual ACM Symposium on Theory of Computing, 1986," pp. 295–303.
18. S. A. KURTZ, S. R. MAHANEY, AND J. S. ROYER, Collapsing degrees, *in* "Proceedings, 27th IEEE Symposium on Foundations of Computer Science, 1986," pp. 380–389.
19. S. MAHANEY, Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis, *J. Comput. System Sci.* **25** (1982), 130–143.
20. S. MAHANEY AND P. YOUNG, Reductions among polynomial isomorphism types, *Theoret. Comput. Sci.* **39** (1985), 207–224.
21. B. MONIEN AND I. H. SUDBOROUGH, On eliminating nondeterminism from Turing machines which use less than logarithm worktape space, *Theoret. Comput. Sci.* **21** (1982), 237–253.

22. D. A. PLAISTED, Complete problems in the first-order predicate calculus, *J. Comput. System Sci.* **29** (1984), 8–35.
23. K. W. REGAN, On diagonalization methods and the structure of language classes, *in* "Proceedings, 4th International Conference on Foundations of Computation Theory," Lecture Notes in Computer Science Vol. 158, pp. 368–380, Springer-Verlag, Berlin/New York, 1983.
24. L. J. STOCKMEYER, "The Complexity of Decision Problems in Automata Theory and Logic," Ph.D. dissertation, Massachusetts Institute of Technology, 1974.
25. O. WATANABE, On one–one polynomial time equivalence relations, *Theoret. Comput. Sci.* **38** (1985), 157–165.