MATHEMATICAL SYSTEMS THEORY **10**, 19–32 (1976)

Relativization of Questions About Log Space Computability

by

RICHARD E. LADNER*

Department of Computer Science
University of Washington
Seattle, Washington 98195

and

NANCY A. LYNCH**

Department of Mathematics
University of Southern California
Los Angeles, California 90007

ABSTRACT

A notion of log space Turing reducibility is introduced. It is used to define relative notions of log space, $\mathscr{L}^A$, and nondeterministic log space, $\mathscr{NL}^A$. These classes are compared with the classes $\mathscr{P}^A$ and $\mathscr{NP}^A$ which were originally defined by Baker, Gill, and Solovay [BGS]. It is shown that there exists a computable set A such that $\mathscr{NL}^A \subsetneq \mathscr{P}^A$. Furthermore, there exists a computable set A such that $\mathscr{NL}^A \not\subseteq \mathscr{P}^A$ and $\mathscr{P}^A \not\subseteq \mathscr{NL}^A$. Also a notion of log space truth table reducibility is defined and shown to be equivalent to the notion of log space Turing reducibility.

Introduction. Reducibility in polynomial time has received wide attention, in references [C2], [K], [Lal], [LLS], [BGS] and in many other places. There are several considerations which support a similar examination of reducibility in log space. First, unlike polynomial time reducibility, log space reducibility allows a meaningful classification of problems that are computable in polynomial time. Second, notions of space bounded reducibility allow us to state relativizations of open problems concerning both the relationship between deterministic and nondeterministic log space computability and the relationship between log space computability and polynomial time computability.

In Section 1 we generalize the definition of log space reducibility used in references [JL] and [SM] to permit Turing-type reductions. We also generalize reducibility to allow arbitrary space bounds and to allow nondeterminism.

*Research supported by NSF Grant No. GJ 43264.
**Research supported by NSF Grant No. DCR 92373.

In Section 2 we relativize certain complexity classes including $\mathcal{P}$ (sets computable in polynomial time) and $\mathcal{L}$ (sets computable in log space), $\mathcal{L}^k$ (sets computable in $\log^k$ space), and $\mathcal{NL}$ (sets computable in nondeterministic log space). For various sets A we compare $\mathcal{NL}^A$ *and* $\mathcal{P}^A$. By an argument found in reference [BGS] there are computable sets A such that $\mathcal{NL}^A = \mathcal{P}^A$. We show that there is a computable set A with $\mathcal{NL}^A \subsetneq \mathcal{P}^A$. On the other hand there is also a computable set A with $\mathcal{NL}^A \not\subseteq \mathcal{P}^A$ and $\mathcal{P}^A \not\subseteq \mathcal{NL}^A$. This latter result is somewhat surprising since it is well known that $\mathcal{NL} \subseteq \mathcal{P}$ [C1].

In Section 3 we try to explain why certain results in complexity theory uniformly relativize while others do not. Results that depend primarily on step-by-step simulations like the space hierarchy theorem of Stearns, Hartmanis, and Lewis [SHL] relativize uniformly. Results like $\mathcal{NL} \subseteq \mathcal{P}$ [C1] and $\mathcal{NL} \subseteq \mathcal{L}^2$ [Sa] do not relativize because they depend on indirect rather than step-by-step simulations.

In Section 4 we introduce a notion of log space truth table reducibility which is analogous to the notion of polynomial time truth table reducibility introduced by Ladner, Lynch and Selman [LLS]. Using the result of Lynch [Ly1], which establishes that a Boolean formula can be evaluated in log space, we argue that our definition is reasonable. We show the equivalence of log space Turing reducibility and log space truth table reducibility.

1. Preliminaries. We consider sets of words over the alphabet $\{0, 1\}$. Let $|x|$ be the length of a word x and let λ represent the empty word.

Our models of computation are variations of Turing machines (see [HU2]). A *Turing machine acceptor* is a Turing machine with a two-way read only input tape and a two-way read-write storage tape. A *Turing machine transducer* is a Turing machine with a two-way read only input, a two-way read-write storage tape and a one-way write only output tape. An *oracle Turing machine* is a Turing machine with a two-way read only input, a two-way read-write storage tape, and a one-way write only oracle tape. Each type of Turing machine may be *deterministic* or *nondeterministic*. All machines are deterministic unless otherwise specified.

A nondeterministic Turing machine T *runs in time* $t(n)$ if for all n and all x of length n, each computation path of T on input x halts within $t(n)$ moves. A nondeterministic Turing machine T *runs in space* $s(n)$ if for all n and all x of length n, each computation path halts with the storage tape head having visited no more than $s(n)$ distinct tape cells. The tape cells visited on the input tape, output tape, and oracle tape are not counted.

Turing machine acceptors have a special state *ACC*. A set $A \subseteq \{0, 1\}^*$ is *accepted* by a nondeterministic acceptor T if for all $x \in \{0, 1\}^*$, $x \in A$ if and only if there is some computation of T on input x which halts in the state *ACC*. Define $TIME(t(n))$ and $SPACE(s(n))$ to be the class of sets which are accepted by Turing machine acceptors which run respectively in time $t(n)$ and space $s(n)$. Define $NTIME(t(n))$ and $NSPACE(s(n))$ to be corresponding classes for nondeterministic Turing machine acceptors. Some special complexity classes we consider are defined:

$$\mathcal{P} = \bigcup_{k \geq 1} TIME(n^k),$$

$$\mathcal{NP} = \bigcup_{k \geq 1} NTIME(n^k),$$

$$\mathcal{L} = SPACE(\log n),$$

$$\mathcal{NL} = NSPACE(\log n),$$

$$\mathcal{L}^k = SPACE(\log^k n).$$

A function $f: \{0,1\}^* \to \{0,1\}^*$ *is computable in time* $t(n)$ (*space* $s(n)$) if there is a Turing machine transducer T that runs in time $t(n)$ (*space* $s(n)$) with the property that for all x, T halts with $f(x)$ written on the output tape. The most commonly used notions of polynomial time and log space reducibility are defined in terms of time and space bounded transducers.

We write $A \leq_m^{\mathcal{P}} B$ (*A is polynomial time many-one reducible to B*) if there is a function f computable in time n^k for some k such that $x \in A$ if and only if $f(x) \in B$.

We write $A \leq_m^{\mathcal{L}} B$ (*A is log space many-one reducible to B*) if there is a function f computable in space log n such that $x \in A$ if and only if $f(x) \in B$.

Oracle Turing machines have special states, *ACC*, *QUE*, *YES*, and *NO*. The state *ACC* is the accepting state while the state *QUE* is called the *query state*. In each state except *QUE* the machine may write a symbol onto the oracle tape. In state *QUE* the machine goes into state *YES* if the word written on the oracle tape is a member of the *oracle set*, otherwise it enters state *NO*. In moving from state *QUE* to *YES* or *NO* no other action is taken except to erase the oracle tape.

We write $A \leq_T^{\mathcal{P}} B$ (*A is polynomial time Turing reducible to B*) if there is an oracle Turing machine M that runs in time n^k for some k and M accepts A with oracle B.

We write $A \leq_T^{\mathcal{L}} B$ (*A is log space Turing reducible to B*) if there is an oracle Turing machine M that runs in space log n and M accepts A with oracle B.

It is straightforward to show that both $\leq_m^{\mathcal{P}}$ and $\leq_T^{\mathcal{P}}$ are transitive relations. Several authors including Jones [J] and Stockmeyer and Meyer [SM] have noted that $\leq_m^{\mathcal{L}}$ is transitive. By a similar argument $\leq_T^{\mathcal{L}}$ is also transitive [La1]. Also by a similar argument it can be shown that if $A \leq_T^{\mathcal{L}} B$ and $B \in \mathcal{L}^k$ then $A \in \mathcal{L}^k$. It is easy to see that $A \leq_m^{\mathcal{L}} B$ implies $A \leq_T^{\mathcal{L}} B$.

Two important classes of complete problems exist for log space reducibility. A set S *is log space complete in* $\mathcal{NL}$ if $S \in \mathcal{NL}$ and for all $A \in \mathcal{NL}$, $A \leq_m^{\mathcal{L}} S$. A set S *is log space complete in* $\mathcal{P}$ if $S \in \mathcal{P}$ and for all $A \in \mathcal{P}$, $A \leq_m^{\mathcal{L}} S$. The second definition could be extended to log space Turing reducibility. It appears that the 'threadable mazes' of Savitch is the first known example of a log space complete problem in $\mathcal{NL}$ [Sa]. The 'path systems' of Cook seem to be the first known example of a log space complete problem in $\mathcal{P}$ [C3]. Other examples can be found in references [J], [JL], [La2], and [Su].

These two classifications of problems are closely related to open problems in automata theory by the following lemmas.

LEMMA 1.1. *If S is log space complete in $\mathcal{NL}$ then $S \in \mathcal{L}$ if and only if $\mathcal{L} = \mathcal{NL}$.*

LEMMA 1.2. *If S is log space complete in $\mathcal{P}$ then for all k, $S \in \mathcal{L}^k$ if and only if $\mathcal{P} \subseteq \mathcal{L}^k$.*

These lemmas follow immediately from the facts in the preceding two paragraphs. Proofs may be found in references [J] and [JL].

One might question introducing log space Turing reducibility when in practice log space many-one reducibility is used. We do so because we believe that Turing reducibility represents the most general form of effective reduction of one problem to another. In particular, we believe that our definition of log space Turing reducibility represents a very general form of effective reduction with a log n space bound of one problem to another.

A more general notion is defined in terms of log space machines with multiple oracle tapes [Ly2]. This paper represents an initial attempt to understand log space Turing reducibility so that we shall restrict ourselves to Turing machines with a single oracle tape.

We note that the log space reducibilities as we define them are much less machine invariant than are the corresponding polynomial time reducibilities. For instance, we could not restrict the input head to be one-way rather than two-way. Certain variations are possible; for example, the class of log space computable functions does not depend on the direction of motion of the output tape head. In fact, we could even allow the output tape head to be two-way, with the ability to write and rewrite (but not read) [M]. The loss of a certain degree of machine invariance is a penalty extracted in exchange for a gain in fineness of classification.

Oracle Turing machines are used to relativize problems. We do it in the following way. Define $TIME^A(t(n))$ and $SPACE^A(s(n))$ to be the class of sets which are accepted by oracle Turing machines using the oracle A and running respectively in time $t(n)$ and space $s(n)$. We may analogously define $NTIME^A(t(n))$ and $NSPACE^A(s(n))$. Special classes are

$$\mathcal{P}^A = \bigcup_{k\geq 1} TIME^A(n^k),$$

$$\mathcal{NP}^A = \bigcup_{k\geq 1} NTIME^A(n^k),$$

$$\mathcal{L}^A = SPACE^A(\log n),$$

$$\mathcal{NL}^A = NSPACE^A(\log n),$$

$$(\mathcal{L}^k)^A = SPACE^A(\log^k n).$$

We repeat for emphasis that our definition of a machine running in time $t(n)$ or space $s(n)$ requires that *all* computation paths (for all inputs and oracle sets) eventually converge. Weakening this requirement leads to reasonable alternative definitions [Si]. All the above classes except $N\mathcal{L}^A$ remain unchanged under the weaker definitions; however, the weaker definition for $N\mathcal{L}^A$ leads to a set of results totally different from those in this paper.

At this point we define precisely several concepts concerning oracle Turing machines that will be used later. Let T be a nondeterministic oracle Turing machine which runs in space $s(n)$, has state set Q and storage tape alphabet Γ. Let x be an input. An *instantaneous description (i.d.) for x and T* has the form (q, i, j, γ) where $q \in Q$ indicates the state, $1 \leq i \leq n$ indicates the input head position, $1 \leq j \leq s(n)+1$ indicates the storage head position, $\gamma \in \Gamma^{s(n)}$ indicates the contents

of the storage tape. The *initial i.d.* is $(q_0, 1, 1, b^{s(n)})$ where q_0 is the start state of T. A *query i.d.* has the form (QUE, i, j, γ). A *yes i.d.* has the form (YES, i, j, γ) and a *no i.d.* has the form (NO, i, j, γ). An *accepting i.d.* has the form (ACC, i, j, γ). It is useful to lump the initial, yes and no i.d. together and call them *begin i.d.'s*. We sometimes say the query i.d. (QUE, i, j, γ) *corresponds* to the yes i.d. (YES, i, j, γ) and to the no i.d. (NO, i, j, γ).

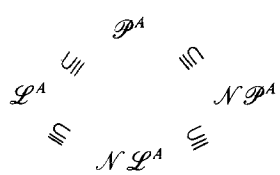
The *i.d. graph for x and T* is defined as follows. The nodes are all the i.d.'s. There is a directed edge from i.d. I to i.d. J if either (i) I is not a query i.d. and J follows in one move of T on input x from I or (ii) $I = (QUE, i, j, \gamma)$ for some i, j, γ and $J = (YES, i, j, \gamma)$ or $J = (NO, i, j, \gamma)$. A *simple path* is a path which does not pass through a query i.d. A *complete simple path* is a simple path from a begin i.d. to a query i.d. To each simple path we associate the *partial query generated by it*, the word written on the oracle tape during the sequence of moves indicated by the simple path. If the simple path is complete, then the partial query is simply called the *query*. Queries generated by complete simple paths in the i.d. graph of x and T are called the *queries generated by T on input x*.

Let $A \subseteq \{0, 1\}^*$ be any oracle set. The *query graph for x, T, and A* is defined as follows. Its nodes are all the begin i.d.'s together with all the accepting i.d.'s. There is a directed edge from i.d. I to i.d. J if either (i) I is a begin i.d. and J is a yes or no i.d., and there is a complete simple path from I to J' where J' is the query i.d. corresponding to J, and the query generated by this path is in A just in case J is a yes i.d., or (ii) J is an accepting i.d. and there is a simple path from I to J. A word y *supports* an edge (I, J) in the query graph if y is generated by a complete simple path from I to the query i.d. corresponding to J and either (i) $y \in A$ and J is a yes i.d. or (ii) $y \notin A$ and J is a no i.d. Queries that support edges in the query graph for x, T and A are called *queries generated by T on input x using oracle A*.

It should be clear that x is accepted by T with oracle A if and only if there is a path from the initial i.d. to an accepting i.d. in the query graph for x, T, and A.

2. Relativizations of $\mathcal{NL}$ and $\mathcal{P}$. It is well know that $\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP}$. It is as yet unknown whether any of the reverse inclusions hold. In this section we examine the possible relationships between the corresponding relativized classes, in the hope of shedding some light on the nonrelativized problems. The approach is similar to that used in the reference [BGS].

To begin with, given any oracle A the following diagram holds.



As we shall see, it is not always the case that $\mathcal{NL}^A \subseteq \mathcal{P}^A$ (Theorem 2.3).

THEOREM 2.1. *There is a computable set $A \subseteq \{0, 1\}^*$ such that $\mathcal{L}^A = \mathcal{NL}^A = \mathcal{P}^A = \mathcal{NP}^A$.*

Proof. The construction in [BGS, Theorem 1] will suffice. Also, if A is log

space complete in polynomial space then following [BGS, Theorem 2], A satisfies the Theorem.

We outline the argument of Baker, Gill and Solovay [BGS, Theorem 2]. There exists a set A which is log space complete in polynomial space [SM]; that is, A is computable in polynomial space and every set B also computable in polynomial space is log space many-one reducible to A.

Let $B \in \mathcal{NP}^A$. Since A is computable in polynomial space then B is computable in nondeterministic polynomial space. By appealing to Savitch's Theorem [Sa] B is computable in polynomial space. Hence $B \leq_m^{\mathcal{L}} A$. Thus $B \in \mathcal{L}^A$. ⊠

THEOREM 2.2. *There is a computable set* $A \subseteq \{0,1\}^*$ *such that* $\mathcal{NL}^A \subsetneq \mathcal{P}^A$.

Proof. Let g be the fast growing function defined by $g(0) = 1$ and $g(n+1) = 2^{g(n)}$. Define $G = \{0^{g(k)} : k \geq 0\}$. In what is to follow we use G as a set of diagonalization points. The set G has several nice properties including the property that it can be decided in space $\log n$ whether or not a string $x \in \{0,1\}^*$ is in G.

We construct sets A and B satisfying:

(i) $\mathcal{NL}^A \subseteq \mathcal{P}^A$,

(ii) $B \notin \mathcal{NL}^A$,

(iii) $B \in \mathcal{P}^A$.

The sets A and B will have the following properties which imply (i) and (iii).

(a) $B \subseteq G$,

(b) $A \subseteq \{0^{g(k)}1x : |x| \leq g(k) \,\&\, k \geq 0\}$,

(c) if $0^n1x \in A$ and y is a prefix of x then $0^n1y \in A$,

(d) if 0^n1x and $0^n1y \in A$ and $|x| = |y|$ then $x = y$,

(e) $0^n \notin B$ if and only if either $0^n \notin G$ or there is a y of length n such that $0^n1y \in A$.

We show later how to construct A and B. Using properties (a)—(e) we show (iii), how to compute B in polynomial time using the oracle A. The following algorithm decides B.

```
begin (Algorithm for B)
    read x ;
    if x ∉ G then REJECT else
    z ← x1 ;
    while |z| < 2|x|+1 do
    begin
        if z1 ∈ A then z ← z1 else
        if z0 ∈ A then z ← z0 else
        ACCEPT
    end ;
    REJECT;
```

end (Algorithm for B)

We leave it to the reader to verify that the algorithm runs in polynomial time.

We now proceed to show (i), $\mathcal{NL}^A \subseteq \mathcal{P}^A$. Let M be a nondeterministic oracle Turing machine that runs in space $\log n$. There is a polynomial q depending on M such that if $|x| = n$ then the number of i.d.'s for x and M is no more than $q(n)$. Furthermore, on input x no query of length greater than or equal to $q(n)$ is generated. We show how to decide in polynomial time whether x is accepted by M with oracle A by showing how to construct the query graph for x, M and A in polynomial time using the oracle A. Once the query graph is constructed then its transitive closure can be computed in polynomial time. From the transitive closure it can be decided immediately whether x is accepted by M with oracle A.

We proceed to construct the query graph in the following steps.

1. Using the oracle A compute the set $Y = \{y \in A : |y| \leq q(n)\}$. This can be done in polynomial time because A is so sparse and simple. The set Y has at most $2q(n)$ members.
2. Construct the sets $\mathfrak{T}$ and $\mathfrak{F}$ defined by:
   $\mathfrak{T} = \{(I, J, \delta) : I$ is a begin i.d., J is an i.d., $\delta \in \{0, 1\}^*$, there is a simple path in the i.d. graph for x and M from I to J which generates the partial query δ, and δ is a prefix of a member of $Y\}$
   $\mathfrak{F} = \{(I, J, \delta a) : I$ is a begin i.d., J is an i.d., $\delta \in \{0, 1\}^*$, $a \in \{0, 1\}$, there is a simple path from I to J which generates the partial query δa, δ is a prefix of a member of Y, and δa is not a prefix of a member of $Y\}$.
   Since Y has at most $2q(n)$ members each of length at most $q(n)$ and there are at most $q(n)$ i.d.'s then the cardinalities of $\mathfrak{T}$ and $\mathfrak{F}$ are bounded by $2(q(n))^4$. The sets $\mathfrak{T}$ and $\mathfrak{F}$ may be constructed in polynomial time by the following algorithm.

   begin (Construction of $\mathfrak{T}$ and $\mathfrak{F}$)
   $\mathfrak{T} \leftarrow \varnothing$; $\mathfrak{F} \leftarrow \varnothing$;
   $\mathfrak{T}' \leftarrow \{(I, I, \lambda) : I$ is a begin i.d.$\}$;
   while $\mathfrak{T} \neq \mathfrak{T}'$ **do**
   begin
   $\mathfrak{T} \leftarrow \mathfrak{T}'$;
   for all $(I, J, \delta) \in \mathfrak{T}$ and all J' which are not begin i.d.'s **do**
   if J' follows from J in one move and generates $a \in \{0, 1, \lambda\}$
   then if δa is a prefix of a member of Y
   then $\mathfrak{T}' \leftarrow \mathfrak{T}' \cup \{(I, J', \delta a)\}$
   else $\mathfrak{F} \leftarrow \mathfrak{F} \cup \{(I, J', \delta a)\}$;
   end
   end (Construction of $\mathfrak{T}$ and $\mathfrak{F}$)
3. Finally we can compute the query graph for A. There is a directed edge from I to J if I is a begin i.d., J' is the query i.d. corresponding to J if J is a yes or no i.d., and one of the following holds:
   (i) J is a yes i.d. and there is $\delta \in Y$ such that $(I, J', \delta) \in \mathfrak{T}$,
   (ii) J is a no i.d., there is a $\delta \in \{0, 1\}^*$ and an i.d. K such that $(I, K, \delta) \in \mathfrak{F}$ and there is a simple path from K to J'.
   (iii) J is an accepting i.d. and there is a simple path from I to J.

We now show how to construct A and B so that $B \notin \mathcal{NL}^A$ and A and B satisfy (a)—(e). As we mentioned earlier, we will use members of G as diagonalization points. That is, if T is an arbitrary nondeterministic oracle Turing machine that runs in space $\log n$ then some member 0^n of G will have the property that $0^n \in B$ *if* T does not accept 0^n with oracle A and $0^n \notin B$ if T does accept 0^n with oracle A. Before getting into the actual definition of A and B we need to prove a certain claim.

Let C be a finite set, let $n > |z|$ for all $z \in C$, and let $2^n > c^2$ where $c =$ the number of i.d.'s for 0^n and T. Define:

$$C'_y = C \cup \{0^n 1x : |x| < |y| \text{ and } x \text{ is a prefix of } y\}$$
$$C''_y = C \cup \{0^n 1x : x \text{ is a prefix of } y\}.$$

Claim. For some y of length n one of the following holds:

(1) 0^n is rejected by T with oracle C'_y,

(2) 0^n is accepted by T with oracle C''_y.

Proof of Claim. Assume (1) fails so that 0^n is accepted by T with oracle C'_y for each y of length n. Let G'_y be the query graph for 0^n, T, and C'_y and let G''_y be the query graph for 0^n, T, and C''_y. All such query graphs share the same nodes.

For each y of length n there is a path P_y in G'_y from the initial i.d. to an accepting i.d. If for some y, P_y is also a path in G''_y then (2) holds. So assume P_y is not a path in G''_y for any y. Since C''_y is obtained from C'_y by the addition of the one word $0^n 1y$ then $0^n 1y$ supports an edge e_y in G'_y which is not supported by any other member of $\overline{C'_y}$. Now, if $|x| = |y| = n$ and $x \neq y$ then $e_x \neq e_y$. For if $x \neq y$ and $e_x = e_y$ then e_y is supported by at least two members of $\overline{C'_y}$, namely $0^n 1x$ and $0^n 1y$, which is impossible. But there are at most c^2 possible edges in any query graph for 0^n and T and 2^n words of the form $0^n 1y$ where $|y| = n$. This is impossible because $2^n > c^2$. Hence (2) holds if (1) fails.

Using the claim we now give the construction of A and B. We let $T_1, T_2, \ldots$ be an effective enumeration of the nondeterministic oracle Turing machines that run in space $\log n$. There is a parameter t which indicates the 'stage' of construction.

```
begin (Construction of A and B)
  A←∅;
  B←∅;
  i←1;
  for s←0 until t do
  begin (stage s)
    n←g(s);
    c←the number of i.d.'s for 0^n and T_i;
    if 2^n ≤ c^2 then A←A ∪ {0^n 1 0^i : 0 ≤ i ≤ n} else
    begin (diagonalization of T_i)
     if 0^n is rejected by T_i with oracle A'_y for some y of length n then
        begin
          choose y of length n such that 0^n is rejected by T_i with oracle A'_y;
          B←B ∪ {0^n};
          A←A'_y
        end
      else
        begin
```

choose y of length n such that 0^n is accepted by T_i with oracle A''_y;
$A \leftarrow A''_y$
end;
$i \leftarrow i+1$
end (diagonalization of T_i)
end (stage s)
end (Construction of A and B)

To decide whether x is a member of A or of B run the construction of A and B with the parameter t where $g(t) > |x|$. On termination check the current values of A and B to determine if x is in the appropriate set.

The construction succeeds if we can show that each T_i is successfully diagonalized, that is, B is not accepted by T_i with oracle A. This can be shown by induction on i. Assume this is true for all $j < i_0$. There is a polynomial p such that the number of i.d.'s for each x and T_{i_0} *where* $|x| = n$ is at most $p(n)$. By the induction hypothesis there is a least number s_0 such that if the value of s is s_0 then the value of i is i_0. Since 2^n dominates $p^2(n)$ then there is an $s_1 \geq s_0$ when diagonalization begins on T_{i_0}. Let $n = g(s_1)$. By the claim and the fact that words that are added to A after stage s_1 are of length greater than or equal to 2^n, which is in turn greater than the length of any query generated by T_{i_0} on input 0^n, we can conclude that $0^n \in B$ if and only if 0^n is rejected by T_{i_0} with oracle A. ☒

THEOREM 2.3. *There is a computable set* $A \subseteq \{0,1\}^*$ *such that* $\mathcal{NL}^A \not\subseteq \mathcal{P}^A$ *and* $\mathcal{P}^A \not\subseteq \mathcal{NL}^A$.

Proof. We omit the details of the proof. The basic idea is to interlace the diagonalization of Theorem 2.2 with the following simple diagonalization (which is used by [BGS] in showing there is an A such that $\mathcal{P}^A \neq \mathcal{NP}^A$).

We construct A and C satisfying

(i) $C \notin \mathcal{P}^A$, and

(ii) $C \in \mathcal{NL}^A$.

To accomplish (ii) we force A and C to have the property that $x \in C$ if and only if $x \in G$ and there is a $y \in A$ of the same length as x.

To demonstrate a typical diagonalization let T be an arbitrary oracle Turing machine that runs in time $p(n)$ where p is a polynomial. Choose n and k such that $n = g(k)$ and $2^n > p(n)$. Choose y of length n such that y is not a query generated in the computation of T on input 0^n using the current oracle A. If T accepts 0^n then do not add anything to A or C. If T rejects 0^n then add 0^n to C and y to A. In either case restrain all other words of length less than 2^n from entering A subsequently.

The interlaced diagonalization will construct sets A, B, and C where $B \in \mathcal{P}^A$, $B \notin \mathcal{NL}^A$, $C \in \mathcal{NL}^A$, and $C \notin \mathcal{P}^A$. The interlacing will be done by doing one kind of diagonalization on points $0^{g(k)}$ where k is even and the other kind of diagonalization on points $0^{g(k)}$ where k is odd.

We should note that we must certainly lose the fact that $\mathcal{NL}^A \subseteq \mathcal{P}^A$ when we combine the constructions. What happens is that we can no longer compute the set $Y = \{y \in A : |y| \leq q(n)\}$ in polynomial time using the oracle A. ☒

The reader may perhaps find it surprising that the easier half of Theorem 2.3 is producing a set A with $\mathcal{NL}^A \not\subseteq \mathcal{P}^A$, in view of the fact that $\mathcal{NL} \subseteq \mathcal{P}$.

One interesting problem that remains open is whether or not there is a set A with $\mathscr{P}^A \subsetneq \mathscr{NL}^A$.

3. Relativizations of Other Problems. As we saw in Section 2 the fact that $\mathscr{NL} \subseteq \mathscr{P}$ does not relativize to arbitrary oracles. There are computable sets A with $\mathscr{NL}^A \not\subseteq \mathscr{P}^A$. Results that do relativize uniformly seem to be those that depend primarily on step-by-step simulations. An example of such a result is the space hierarchy theorem of Stearns, Hartmanis, and Lewis [SHL].

THEOREM 3.1. *Let A be any subset of $\{0, 1\}^*$ and let s and r be natural number functions with s uniformly tape constructable,* $\liminf_n(s(n)/\log n) > 0$ *and* $\liminf_n(r(n)/s(n)) = 0$. *Then* $SPACE^A(s(n)) - SPACE^A(r(n)) \neq \varnothing$.

(A function s is *uniformly tape constructable* if there is a Turing machine acceptor T with the property that for all n and all x of length n, on input x, T scans exactly $s(n)$ storage tape cells. This notion is a somewhat stronger notion of tape constructability than was used by Stearns, Hartmanis and Lewis.)

Proof. We omit the details of the proof, since it is essentially the same as that in [SHL] with some minor modifications outlined below.

A set $B \subseteq \{0, 1\}^*$ is constructed with $B \in SPACE^A(s(n)) - SPACE^A(r(n))$. If $x \in \{0, 1\}^*$ then x codes up an oracle Turing machine description in the initial nonzero portion of x; that is, if $x = d10^m$ then d describes an oracle Turing machine.

To determine if $x = d10^m$ is in B in space $s(|x|)$ using the oracle A, we simulate d on the input x, always bounding the space used in the simulation to $s(|x|)$ and the time to $2^{s(|x|)}$. The query generated by the simulation of d is put onto the oracle tape which acts as an oracle tape to d.

Should d accept the input in the allocated space and time, then x is rejected, otherwise x is accepted.

It follows that $B \in SPACE^A(s(n)) - SPACE^A(r(n))$. ⊠

Other results that relativize uniformly include: (i) the characterization of $\mathscr{NP}$ by polynomial length bounded quantifiers over relations in $\mathscr{P}$ [C2]; and (ii) equivalence of two-way multihead finite automata and Turing machines that run in space $\log n$ [H] [HY]. The former fact was pointed out to us by A. Selman.

There are a wide variety of results in automata theory that depend on indirect rather than step-by-step simulations. Among them are $\mathscr{NL} \subseteq \mathscr{P}$ [C1], $\mathscr{NL} \subseteq \mathscr{L}^2$ [Sa], $\mathscr{P}$ is equal to the class of languages accepted by nondeterministic log space bounded auxiliary pushdown store machines [C1], and $NSPACE(n^2)$ is equal to the class of languages accepted by nonerasing stack automata [HU1]. These kinds of results in general do not relativize uniformly. As a paradigm we offer the following theorem.

THEOREM 3.2. *Let p be any polynomial. There is a computable set $A \subseteq \{0, 1\}^*$ with the property that $\mathscr{NL}^A \not\subseteq SPACE^A(p(n))$.*

Proof. This is a diagonalization similar to that of Theorem 2.3. We outline the proof. Let $k =$ the degree of $p(n)$. We construct A and B so that $B \in \mathscr{NL}^A - SPACE^A(p(n))$. Define the fast growing function h by $h(0) = 1$ and $h(n+1) = 2^{(h(n))^{k+1}}$. Further define $H = \{0^{h(n)} : n \geq 0\}$. We use the set H as a set of

diagonalization points. It should be noted that H can be decided in space log n.

We achieve $B \in \mathcal{NL}^A$ by defining $x \in B$ if and only if $x \in H$ and there is a word of length $|x|^{k+1}$ in A.

We diagonalize in the following way. Let T be an arbitrary oracle Turing machine that runs in space $p(n)$. Assume T has s states and t storage tape symbols. Choose $0^n \in H$ such that $2^{n^{k+1}} > snp(n)t^{p(n)}$ so that $2^{n^{k+1}}$ is greater than the total number of i.d.'s for 0^n and T. Choose a y of length n^{k+1} which is not a query generated by T on input 0^n using the current oracle A. Such a y exists because T must make less than $2^{n^{k+1}}$ moves on input 0^n. If 0^n is accepted by T then do nothing to A and B. If 0^n is rejected by T then add 0^n to B and y to A. In either case restrain all other words of length less than $2^{n^{k+1}}$ from entering A subsequently. ⊠

COROLLARY 3.3. *There is a computable set* $A \subseteq \{0, 1\}^*$ *such that* $\mathcal{NL}^A \not\subseteq (\mathcal{L}^2)^A$.

4. Log Space Truth Table Reducibility. The motivation for studying log space truth table reducibility comes from the investigation of polynomial time truth table reducibility in [LLS]. The intuitive idea behind truth table reducibility is the following. A set A is truth table reducible to a set B if given x we can generate (independent of B) queries $y_1, y_2, \ldots, y_m$ and a Boolean function α such that $x \in A$ if and only if $\alpha(B(y_1), \ldots, B(y_m)) = 1$ (where $B(y) = 1$ if $y \in B$ and $B(y) = 0$ if $y \notin B$). In [LLS] this notion is restricted to be polynomial time bounded, and it is shown that polynomial time truth table reducibility and polynomial time Turing reducibility are distinct notions.

Our definition of log space truth table reducibility is analogous to the definition of polynomial time truth table reducibility in [LLS] with a slight modification.

Let $\Delta = \{a, b\}$. A *tt-condition* is a member of $(\Delta^* c \{0, 1\}^* c)^* \Delta^*$. A *tt-condition generator* is a computable function mapping $\{0, 1\}^*$ into the set of tt-conditions. A *tt-condition evaluator* is a computable mapping of $(\Delta^* \{0, 1\})^* \Delta^*$ into $\{0, 1\}$. Let e be a tt-condition evaluator; a tt-condition $\alpha_1 c y_1 c \alpha_2 c y_2 c \ldots \alpha_k c y_k c \alpha_{k+1}$ (with $\alpha_i \in \Delta^*$ and $y_i \in \{0, 1\}^*$) is *e-satisfied by* $B \subseteq \{0, 1\}^*$ if $e(\alpha_1 B(y_1) \alpha_2 B(y_2) \ldots \alpha_k B(y_k) \alpha_{k+1}) = 1$.

Define $A \leq_{tt}^{\mathcal{L}} B$ (*A is log space truth table reducible to B*) if there exist a log space computable tt-condition generator g and a log space tt-condition evaluator e such that $x \in A$ if and only if $g(x)$ is e-satisfied by B. We may also define $A \leq_{tt}^{\mathcal{P}} B$ (*A is polynomial time truth table reducible to B*) if the generator and evaluator are computable in polynomial time. This definition is equivalent to the definition of $\leq_{tt}^{\mathcal{P}}$ in [LLS].

If our abstract definition of $\leq_{tt}^{\mathcal{L}}$ is to be reasonable it should include as special cases some of the common representations of Boolean functions. We list the three basic representations of Boolean functions in increasing order of efficiency of size: (i) truth tables, (ii) Boolean formulas in all binary and unary operations, (iii) Boolean circuits using all possible binary and unary gates. It turns out that truth tables and Boolean formulas can be used as truth table conditions, while it seems in general that Boolean circuits cannot. The trouble with Boolean circuits is that

the problem of evaluating them is log space complete in $\mathscr{P}$ [La2]. Hence they can be evaluated in log space if and only if $\mathscr{P} \subseteq \mathscr{L}$.

At this point we give an example of a log space truth table reduction procedure. Let $A, B \subseteq \{0, 1\}^*$. The sets A and B can be coded into one set $A \oplus B = \{x0 : x \in A\} \cup \{x1 : x \in B\}$. It can be shown using techniques of [LLS] that there are computable sets A and B with $A \cup B \not\leq_m^{\mathscr{L}} A \oplus B$. On the other hand, it is quite easy to show that $A \cup B \leq_{tt}^{\mathscr{L}} A \oplus B$. Consider the following generator and evaluator. Let:

$$g(x) = cx0c \vee cx1c$$

$$e(\sigma \vee \tau) = \begin{cases} 0 & \text{if } \sigma = \tau = 0 \\ 1 & \text{otherwise} \end{cases}$$

(Technically the symbol $\vee$ is coded in the alphabet Δ.)
Clearly, $x \in A \cup B$ if and only if $g(x)$ is e-satisfied by $A \oplus B$.

Define a *general Boolean formula* (*gBf*) inductively as either: (i) a member of $c\{0, 1\}^*c$ or (ii) (P^*Q) or $(\sim P)$ where $^* \in \{\wedge, \vee, \oplus, \ldots\}$ = all binary Boolean operation symbols and P and Q are *gBf*'s. Define a *Boolean formula* in the same way as a *gBf* except replace the first condition with "a member of $\{0, 1\}$". If P is a Boolean formula then define $v(P)$ to be the value of P in the usual way. If P is a general Boolean formula and $B \subseteq \{0, 1\}^*$ then we know what it means for P to be v-satisfied by B. Define $A \leq_{Bf}^{\mathscr{L}} B$ (*A is log space Boolean formula reducible to B*) if there is a log space computable general Boolean formula generator g such that $x \in A$ if and only if $g(x)$ is v-satisfied by B. We could also analogously define what it means for A to be *polynomial time Boolean formula reducible to B*.

THEOREM 4.1. *For all $A, B \subseteq \{0, 1\}^*$, if $A \leq_{Bf}^{\mathscr{L}} B$ then $A \leq_{tt}^{\mathscr{L}} B$.*

Proof. The alphabet of Boolean formulas could be coded easily into a two letter alphabet like Δ. By Lynch, Boolean formulas can be evaluated in space $\log n$ [Ly1]. Hence the function v is computable in space $\log n$. ⊠

We do not know whether or not $\leq_{Bf}^{\mathscr{L}}$ and $\leq_{tt}^{\mathscr{L}}$ are equivalent notions. Another closely related problem is whether or not $\leq_{Bf}^{\mathscr{P}}$ and $\leq_{tt}^{\mathscr{P}}$ are equivalent. Both problems are closely related to the problem of whether or not there is a polynomial p such that given any Boolean circuit P there is an equivalent Boolean formula Q such that $SIZE(Q) \leq p(SIZE(P))$.

We now show the equivalence of $\leq_{tt}^{\mathscr{L}}$ and $\leq_{T}^{\mathscr{L}}$. As we mentioned earlier, this is in contrast to the polynomial time analogue where $\leq_{tt}^{\mathscr{P}}$ is properly stronger than $\leq_{T}^{\mathscr{P}}$.

THEOREM 4.2. *For all A and $B \subseteq \{0, 1\}^*$, $A \leq_{tt}^{\mathscr{L}} B$ if and only if $A \leq_{T}^{\mathscr{L}} B$.*

Proof. Assume $A \leq_{tt}^{\mathscr{L}} B$ via a generator g and evaluator e. We outline the action of an oracle Turing machine T that runs in space $\log n$ such that T accepts A with B as its oracle. Let G and E be the log space transducers that compute g and e respectively.

Let x be an input of length n and let $g(x) = a_1 c y_1 c a_2 c y_2 c \ldots c a_k c y_k c a_{k+1}$ where $a_i \in \Delta^*$ and $y_i \in \{0, 1\}^*$. The Turing machine T on input x will simulate E on input

$w = a_1 B(y_1) a_2 B(y_2) a_3 \ldots a_k B(y_k) a_{k+1}$. Of course T cannot write w in log space, but because g is computable in space log n then the length of w is bounded by a polynomial. So T simply keeps a count c_E of where the read head on w is in the simulation of E. Because the count c_E is bounded by a polynomial the count c_E can be stored in log n storage tape cells.

To discover the c_E-th letter of w, T simulates G on input x in the following way. A count c_G, which is initially equal to c_E, is maintained. Each time an output symbol in Δ is generated and each odd time a c is generated the count c_G is decremented by one. The count is *not* decremented when a member of $\{0, 1, \lambda\}$ is generated as an output symbol. When $c_G = 0$ then stop. If the last symbol generated is in Δ then that symbol is the c_E-th letter of w. If the last letter is a c then a 'query' is about to be generated by E, so continue simulating E, entering the output of E onto the oracle tape of T, until a c again is output. Now, T enters the state *QUE*. Should T enter state *YES* then the c_E-th letter of w is 1 and should T enter state *NO* then the c_E-th letter of w is 0. The details of T are left to the reader.

Now, assume $A \leq_T^{\mathscr{L}} B$. Let T accept A with oracle B in space log n. The important thing to notice is that given x the only potential queries by T are generated by complete simple paths in the i.d. graph for x and T. Because T is deterministic the number of complete simple paths is less than or equal to the number of begin i.d.'s for x and T.

The generator g is defined by $g(x) = a_1 c y_1 c a_2 c \ldots c a_k c y_k c \beta$ where $a_1, \ldots, a_k$ are the begin i.d.'s that initialize the complete simple paths, y_i is the query generated by the complete simple path initialized by a_i and β is a list of the begin i.d.'s that lead by a simple path to accepting i.d.'s, followed by the input x itself. Of course the a_i's and β are coded into the alphabet Δ. The function g can be computed in space log n, by cycling through all the i.d.'s for x and T and making output as required by the definition of g.

The evaluator e is a simulator of T. Let a typical input to e be $a_1 \sigma_1 a_2 \sigma_2 \ldots a_k \sigma_k \beta$, where $a_1, \ldots, a_k$ are begin i.d.'s, β is a list of begin i.d.'s followed by the input x, and $\sigma_i \in \{0, 1\}$.
The Turing machine that computes e behaves as follows.

```
begin
  I←the initial i.d. for x and T;
  while I is in the list of i.d.'s a_1, ..., a_k do
    begin
      let I = a_i ;
      simulate T from i.d. I until a query i.d. J is reached ;
      if σ_i = 1 then I←the yes i.d. corresponding to J
      else I←the no i.d. corresponding to J ;
    end ;
  if I is in the list of i.d.'s in β then write 1 else write 0
end
```

It should be fairly clear that e can be computed in space log n and that $x \in A$ if and only if $g(x)$ is e-satisfied by B. ⊠

COROLLARY 4.3. *The reducibility* $\leq_T^{\mathscr{L}}$ *is properly stronger than the reducibility* $\leq_T^{\mathscr{P}}$.

Proof. It is clear that if $A \leq_T^{\mathscr{L}} B$ then $A \leq_T^{\mathscr{P}} B$. In [LLS] it is shown that there are sets A and B with $A \leq_T^{\mathscr{P}} B$ and $A \not\leq_{tt}^{\mathscr{P}} B$. These same two sets have the property that $A \leq_T^{\mathscr{P}} B$ and $A \not\leq_T^{\mathscr{L}} B$. ⊠

Two problems related to this Corollary remain open: (i) are $\leq_{tt}^{\mathscr{L}}$ and $\leq_{tt}^{\mathscr{P}}$ distinct notions and (ii) are $\leq_m^{\mathscr{L}}$ and $\leq_m^{\mathscr{P}}$ distinct notions. Both problems are closely related to the open question, whether or not the class of functions computable in polynomial time is different from the class of functions computable in log space.

REFERENCES

[BGS] T. BAKER, J. GILL, and R. SOLOVAY, Relativizations of the $\mathscr{P}=\mathscr{NP}$ question. To appear in *SIAM Journal on Computing.*

[C1] S. A. COOK, Characterizations of pushdown machines in terms of time-bounded computers, *Journal of the ACM,* **18** (1971) 4–18.

[C2] S. A. COOK, The complexity of theorem proving procedures, *Proc. Third Annual ACM Symposium on Theory of Computing,* 1971 pp. 151–158.

[C3] S. A. COOK, An observation on time-storage trade off, *Proc.of Fifth Annual ACM Symposium on Theory of Computing,* 1973 pp. 29–33.

[H] J. HARTMANIS, On nondeterminacy of simple computing devices, *Acta Informatica,* **1** (1972) 336–344.

[HU1] J. E. HOPCROFT and J. D. ULLMAN, Nonerasing stack automata, *Journal of Computer and System Sciences,* **1** (1967) 166–186.

[HU2] J. E. HOPCROFT and J. D. ULLMAN, *Formal Languages and Their Relation to Automata,* Addison-Wesley, Reading, Ma. 1969.

[HY] P. HSIA and R. YEH, Finite automata with marks, in *Automata, Languages and Programming,* M. Nivat, editor, American Elsevier, New York 1973.

[J] N. D. JONES, Space-bounded reducibility among combinatorial problems, *Journal of Computer and System Sciences,* **11** (1975) 68–85.

[JL] N. D. JONES and W. T. LAASER, Complete problems for deterministic polynomial time, *Proc. of Sixth Annual ACM Symposium on Theory of Computing,* pp. 40–46 1974.

[K] R. M. KARP, Reducibility among combinatorial problems, in *Complexity of Computer Computations,* R. Miller and J. Thatcher, editors, Plenum Press, New York 1972.

[La1] R. E. LADNER, On the structure of polynomial time reducibility, *Journal of the ACM,* **22** (1975) 155–171.

[La2] R. E. LADNER, The circuit value problem is log space complete for $\mathscr{P}$, *SIGACT News,* **7** (1975) 18–20.

[LLS] R. E. LADNER, N. A. LYNCH, and A. L. SELMAN, A comparison of polynomial time reducibilities. To appear in *Theoretical Computer Science.*

[Ly1] N. A. LYNCH, Log space recognition and translation of parenthesis languages, manuscript.

[Ly2] N. A. LYNCH, Log space machines with multiple oracle tapes, manuscript.

[M] A. R. MEYER, private communication.

[Sa] W. J. SAVITCH, Relationship between nondeterministic and deterministic tape complexities, *Journal of Computer and System Sciences,* **4** (1970) 177–192.

[Si] SIMON, ISTVAN. Private communication (Ph.D. Thesis in preparation. Stanford).

[SHL] R. E. STEARNS, J. HARTMANIS, and P. M. LEWIS II, Hierarchies of memory limited computations, *IEEE Conf. Record on Switching Circuit Theory and Logical Design,* (1965) 191–202

[SM] L. STOCKMEYER and A. R. MEYER, Word problems requiring exponential time *Proc. of Fifth Annual ACM Symposium on Theory of Computing,* (1973) pp. 1–9.

[Su] I. H. SUDBOROUGH, On tape-bounded complexity classes and multihead automata, *Journal of Computer and System Sciences,* Vol, **10** (1975) 62–76.

(Received August 6, 1975 and in revised form February 10, 1976)