Theoretical Computer Science 6 (1978) 25–39.

LOG SPACE MACHINES WITH MULTIPLE ORACLE TAPES

Nancy LYNCH*

School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332, U.S.A.

Communicated by Albert Meyer
Received November 1975
Revised April 1977

Abstract. As an alternative to previously studied models for space-bounded relative computation, an oracle Turing machine with a space bound on its worktape and an arbitrary number of oracle tapes is considered. Basic properties of the resulting reducibilities are examined.

1. Introduction

Complexity-bounded reducibilities between problems have been widely studied in recent years ([1, 2, 3, 5–8, 11, 12, 17, 19] and many others). There appear to be two major motivations for studies of this type.

First, complexity-bounded reducibilities are an extremely powerful tool for the classification of problems by their time and space complexities. There are many examples of problems whose "absolute" complexities are unknown, but which can be shown to be related to each other by very restrictive complexity-bounded reducibilities ([3, 5, 6]). Classes of open problems are thus reduced to single open problems. Even if the absolute complexity of two problems were known, their relative complexity classification might still be interesting from the point of view of algorithm design; for example, complexity-bounded reducibilities could be used to carry out the complexity analysis of an algorithm in a modular way. Finally, complexity-bounded reducibilities have provided the framework for proofs of large lower bounds on the complexity of natural problems ([12] and many others).

For classification purposes, it is generally important that the reducibility $\leq$ have some type of transitivity, and that properties roughly expressible in the form "$A \leq B$, $B \in \mathscr{C} \Rightarrow A \in \mathscr{C}$" hold (where $\mathscr{C}$ is a complexity class of problems). The reducibility should be defined using a reasonably natural model for relative computation, and ideally, should have some invariance over such models.

* Ideas for this work originated while the author was visiting IBM Research, Yorktown Heights, New York. Part of the work was supported by N.S.F. grant DCR75-02373.

There is a second and entirely different motivation for some of the relative complexity results in [1, 2, 7], however. There, the reducibilities are chosen to correspond as closely as possible to Turing reducibility in recursive function theory [14]. The reason is that many simulation constructions of recursive function theory "relativize" using Turing reducibility. Since proofs of equality or inequality of complexity classes of problems might be expected to involve simulation constructions, use of the proper reducibilities and examination of the relativizations of such problems ought to yield insight into the unrelativized problems. For example, it is shown in [1] that the relativized version of the $\mathcal{P} =^? \mathcal{NP}$ question can be answered in either way, depending on the choice of oracle set. Baker, Gill and Solovay's interpretation of this phenomenon is that ordinary simulation constructions (including ordinary diagonalization as a special case) are unlikely to solve the basic problem, because such constructions would relativize. Their conclusion is that study of specific natural decision problems is more likely to succeed than is an attempt to provide constructions of a general nature.

In [7], space-bounded reducibilities were studied, motivated both by classification and by relativization considerations. Various definitions presented in that paper seemed somewhat useful for classification purposes. Also, more interestingly, work involving a space-bounded reducibility similar to Turing reducibility led us to question the informal conclusions of [1]. Namely, we showed that the relativized versions of other questions, such as $\mathcal{NL} \subseteq^? \mathcal{P}$ or $\mathcal{NL} \subseteq^? \mathcal{L}^2$ [16] (where $\mathcal{NL}$ and $\mathcal{L}^2$ represent the collections of languages recognizable in nondeterministic log space and in deterministic $\log^2$ space respectively) can similarly be answered in either way, depending on the oracle set. Since the answers to these basic questions are known, and in fact use general simulations in their proofs, we cannot conclude that general diagonalization and simulation constructions are doomed to failure in the study of problems such as $\mathcal{P} =^? \mathcal{NP}$. The key factor seems to us to be that the kind of simulation involved is not a direct, step-by-step simulation, but rather a more "global" or "graph-theoretic" simulation involving state accessibility. Such simulations also occur in [13] and [4]; these results also appear not to relativize directly (although techniques of the type used in [1] and [7] do not seem to be fine enough to show that the questions could be answered in both ways). Our point is simply that general constructions should not be discarded as possible techniques for solving difficult complexity-class problems.

One objection that has been raised to the conclusions in [7] is that the log space Turing reducibility used in that paper does not satisfy all the criteria desirable for classification reducibilities; namely, it is not invariant over several minor modifications in the machine model (nonerasure of oracle tapes, counting space on the oracle tape, not requiring termination on all paths, and allowing larger numbers of oracle tapes, for instance). This paper represents consideration of the last of these variations. This study was originally undertaken because a question left open in [7] would have had a pleasing answer using a many-tape model; it was hoped that

the many-tape model would prove more natural than the one-tape model. In [17] a model which does not require termination on all computation paths is considered. A model which counts space on the oracle tape is considered in [18]; this model has the unpleasant property that it fails to generalize the very basic many-one reducibility used in [5] and [15]. The case of nonerasure, as far as we know, has not been studied.

Study of these many variations, each with some natural properties, has forced us to the conclusion that space complexity does not have a single natural relativization, but that different reducibilities may prove useful in different classification situations. Another conclusion is that the whole concept of relativization does not seem to fit into complexity theory in the clean way it fits into recursion theory. The restrictions of complexity theory do not seem to be of the type that admit introduction of an oracle set in any uniform way.

But we note that for *each* reasonable Turing-type definition we have suggested (both weak and strong), *known* simulation constructions fail to relativize. This fact seems to indicate that the reason some constructions fail to relativize is not only that space does not relativize properly, but also that the simulations are of an "indirect" type that does not readily adapt to introduction of an oracle in any way.

Basic properties of the many-tape model are examined in this paper; it is shown that it has all the properties wanted for classification purposes, except for invariance. Some of the technical proofs may be interesting because they involve some limitations on the kinds of computations that can be performed in log space.

In Section 2, we give our definitions and show that allowing several oracle tapes does indeed provide us with a more general log space reducibility than allowing only one.

In Section 3, we consider relationships between polynomial time reducibility and the new log space reducibilities. A convenient property of various types of log space computability is that computations which are performed in log space generally are performed in time bounded above by a polynomial. This seems natural; the transition from space to time classification for a problem should probably involve at most an exponential increase in complexity. Our new definitions seem to allow considerable liberty for "log space" reducibilities; a log space bound is required of the worktape only, while oracle tapes may use space polynomial in the length of the input. If such liberty caused "log space" computations to require exponential time, it would probably be unreasonable. However, we show that polynomial time does suffice to carry out all our log space computations; thus, our generalization seems like an acceptable one.

We then show that the inclusion of our new reducibilities in polynomial time reducibility is proper. It is hoped that the elaboration of techniques such as those used in proofs of this type may eventually prove useful in showing that certain (non-relativized) problems cannot be solved in log space.

In Section 4, we examine relationships among the new log space reducibilities. We first show a kind of transitivity result. Then we show that adding successively higher numbers of oracle tapes yields a hierarchy of reducibilities. The simulation techniques used for the former result originate in [15] and [9], while the diagonalization techniques used for the latter result are very similar to those in Section 3.

Finally, in Section 5, we mention related results and questions, and briefly discuss relativization using the new definitions.

2. Notation, definitions and basic results

We consider sets of strings over the alphabet $\{0, 1\}$. Let $|x|$ represent the length of string x, and let λ represent the empty string. An ordering of strings by length, and lexicographically within sets of equal length strings, will be used.

A k-oracle Turing machine is a Turing machine with a two-way read-write worktape and k (≥ 0) one-way write-only oracle tapes. The machine may be deterministic or nondeterministic; it is deterministic unless otherwise specified. k-oracle Turing machines have special states START, ACCEPT, REJECT, YES, NO, AND Que_i, $1 \leq i \leq k$. The latter are called *query states*. From each state which is not a query state, the machine may write a symbol, 0 or 1, onto any single oracle tape. From state Que_i, the machine enters state YES if the string written on the i^{th} oracle tape is in the *oracle set*; otherwise, it enters state NO. In moving from state Que_i to YES or NO, no other action is taken except to erase the oracle tape.

A k-oracle Turing machine *M runs in time t* if for all n, all x of length n and all oracle sets B, each computation of M on input x and oracle set B halts (i.e. enters state ACCEPT or REJECT) within $t(n)$ moves. A k-oracle Turing machine *M runs in space s* if for all n, x and B as above, each computation of M on input x and oracle set B halts with no more than $s(n)$ distinct tape squares visited by the worktape head. Note that the tape cells visited on the input and oracle tapes are not counted.

An instantaneous description (*i.d.*) for a k-oracle Turing machine M and input x is a quadruple (q, i, j, y), where q is the state, i ($0 \leq i \leq |x|+1$) is the input head position, j (≥ 0) is the worktape head position and y is the contents of the worktape. A *tape contents* for a k-oracle Turing machine is an element of $(\{0, 1\}*)^k$. Here, the i^{th} component denotes the contents of the i^{th} oracle tape ($1 \leq i \leq k$).

$A \leq^{\mathcal{L}(k)} B$ if there is a k-oracle Turing machine that runs in log space and with oracle set B accepts exactly the members of A.

$A \leq^{\mathcal{L}(\omega)} B$ if $A \leq^{\mathcal{L}(k)} B$ for some k.

$A \leq^{\mathcal{P}(k)} B$ if there is a k-oracle Turing machine that runs in time p for some polynomial p, and with oracle set B accepts exactly the members of A.

$A \leq^{\mathcal{P}(\cdot)} B$ may be defined analogously. By allowing nondeterminism in the oracle machines, we may also define $A \leq^{\mathcal{NL}(k)} B$, $A \leq^{\mathcal{NL}(\omega)} B$, $A \leq^{\mathcal{NP}(k)} B$ and $A \leq^{\mathcal{NP}(\omega)} B$. However, it is easy to see that:

Remark 2.1. For any sets A and B, $A \leqslant^{\mathcal{P}(\omega)} B$ iff $A \leqslant^{\mathcal{P}(1)} B$, and $A \leqslant^{\mathcal{NP}(\omega)} B$ iff $A \leqslant^{\mathcal{NP}(1)} B$.

Thus, for polynomial time-bounded relative computation, our extension of the usual definitions is insignificant. (This is generally the case for reasonable variations on such definitions for time-bounded reducibilities.) The index is thus omitted in these cases, and $A \leqslant^{\mathcal{P}} B$ and $A \leqslant^{\mathcal{NP}} B$ are written. The situation is different for log space relative computation.

Theorem 2.2. *There exist recursive sets A and B for which $A \leqslant^{\mathcal{L}(2)} B$ but $A \not\leqslant^{\mathcal{L}(1)} B$.*

Proof. B will be defined in stages; A will then be determined by

$$x \in A \quad \text{iff} \quad xy_1 \cdots y_{|x|} \in B,$$

where for all i with $1 \leqslant i \leqslant |x|$, $y_i \in \{0, 1\}$ and $[y_i = 1$ iff $x1^{i-1} \in B]$. Clearly, this relationship ensures that $A \leqslant^{\mathcal{L}(2)} B$.

Let $\{M_n\}$ be an effective enumeration of a class of 1-oracle log space Turing machines sufficient to compute all (relative) functions computable by such machines. Before stage n, B will be defined on an initial segment B_n of all binary strings under the assumed ordering; during stage n, the definition of B will be extended to ensure that M_n with oracle B does not accept exactly the members of A.

Before stage 0, define $\lambda \in B$.

Begin *stage n.*

Choose p to be a polynomial such that for every string x, there is a set Q of at most $p(|x|)$ distinct strings, containing all strings about which M_n queries *any* oracle set, on input x. (The existence of such a polynomial results from the bounded number of i.d.'s for M_n and x.) Choose $x \notin B_n$ with $2^{|x|} > p(|x|)$, and fix Q as above, for this particular value of x.

Choose y so that $|y| = |x|$ and $xy \notin Q$ (xy denotes the concatenation of x and y). Write $y = y_1 \cdots y_{|x|}$, where $y_i \in \{0, 1\}$ for all i. Extend the definition of B so that for all i with $1 \leqslant i \leqslant |x|$, $[x1^{i-1} \in B$ iff $y_i = 1]$, and so that B is defined on all members of Q. Put xy into $\bar{B}$ if M_n on input x and oracle B accepts; otherwise put xy into B. Further extend the definition of B so that B is again defined on an initial segment.

End

At stage n, it is ensured that there is an x such that $x \in \bar{A}$ iff M_n on input x and oracle B accepts; thus $A \not\leqslant^{\mathcal{L}(1)} B$. □

We write $\mathcal{L}(k)^B$ for $\{A \mid A \leqslant^{\mathcal{L}(k)} B\}$. Analogous definitions are used for $\mathcal{L}(\omega)^B$, $\mathcal{P}^B$, $\mathcal{NL}(k)^B$, $\mathcal{NL}(\omega)^B$ and $\mathcal{NP}^B$.

3. Deterministic log space and polynomial time reducibilities

Motivated by considerations mentioned in the Introduction, we show that the new definitions of (deterministic) log space reducibility yield relations which are subsets of polynomial time reducibility.

Theorem 3.1. *For any set B, $\mathcal{L}(\omega)^B \subseteq \mathcal{P}^B$.*

Proof. We show that any deterministic k-oracle Turing machine M which runs in log space, in fact runs in time p for some polynomial p.

Clearly, there exists a polynomial p_1 such that the number of distinct i.d.'s for M and any input x is at most $p_1(|x|)$. Let $p(n) = 2^k p_1(n)$.

Consider the computation of M on any fixed input x, and any fixed oracle B. Assume at least $p(|x|)$ steps (and thus $p(|x|)+1$ successive i.d.'s) occur during the computation. Then there must be some i.d. α which occurs 2^k+1 distinct times during the computation. Then, there must be integers t_1 and t_2, $0 \leq t_1 < t_2 \leq p(|x|)$, such that M has i.d. α after exactly t_1 steps and also after exactly t_2 steps, and such that for all i, $1 \leq i \leq k$, the following is satisfied:

(1) If M enters state Que_i any time after (or at) t_2 steps, if r_1 is the string on oracle tape i the first time after (or at) t_1 steps that M enters Que_i, and if r_2 is the string on oracle tape i the first time after (or at) t_2 steps that M enters Que_i, then $r_1 \in B$ iff $r_2 \in B$.

This is because to each t, $0 \leq t \leq p(|x|)$, may be associated a vector $a_1, \ldots, a_k$ of zeros and ones such that for all i,

$$a_i = \begin{cases} 0 & \text{if } M \text{ does not enter } \text{Que}_i \text{ after (or at) } t \text{ steps, or} \\ & \text{if } M \text{ enters state } \text{Que}_i \text{ after (or at) } t \text{ steps with } r \text{ the string on} \\ & \quad \text{oracle tape } i \text{ the first time this occurs, and } r \notin B, \\ 1 & \text{otherwise (i.e. if } M \text{ enters state } \text{Que}_i \text{ after (or at) } t \text{ steps with } r \\ & \text{the string on oracle tape } i \text{ the first time this occurs, and } r \in B). \end{cases}$$

Of the 2^k+1 distinct times for which α is the i.d., two may be chosen with the same associated vector. These two will be the needed t_1 and t_2.

But then the determinism of M ensures that the computation after t_2 steps proceeds in exactly the same way as the computation after t_1 steps, so that the considered computation cannot terminate. (Note that the fact that the oracle tapes are erased after queries is important here.) □

For the same reasons as given in the Introduction, we would like to have a result analogous to Theorem 3.1 for nondeterministic machines. We do not know if such a result is true.

Question 3.2. Is it true that for all set B, $\mathcal{NL}(\omega)^B \subseteq \mathcal{NP}^B$?

Although a reasonable definition of log space reducibility should provide a relation which is a subset of polynomial time reducibility, it seems unlikely that such a definition could include all of polynomial time reducibility. Verifying proper inclusion for our definitions involves careful analysis of the set of oracle questions which are capable of influencing the course of a computation.

Theorem 3.3. *There is a recursive set B such that $\mathscr{L}(\omega)^B \neq \mathscr{P}^B$.*

Proof. B will be defined in stages; A will be determined by

$$x \in A \quad \text{iff} \quad xy_1 \cdots y_{|x|} \in B,$$

where for all i with $1 \leqslant i \leqslant |x|$, $y_i \in \{0, 1\}$ and $[y_i = 1$ iff $xy_1 \cdots y_{i-1} \in B]$. Clearly, this relationship ensures that $A \leqslant^{\mathscr{P}} B$.

Let $\{M_n\}$ be an effective enumeration of log space k-oracle Turing machines (for all k) sufficient to compute all functions computable by such machines. Before stage n, an initial segment B_n of B will be defined; during stage n, the definition of B will be extended to ensure that M_n with oracle set B does not accept exactly the members of A.

Before stage 0, define $\lambda \in B$.

Begin *Stage n.*

Assume M_n is a k-oracle Turing machine. Choose p to be a polynomial such that for all strings x, $p(|x|)$ is an upper bound on the number of i.d.'s of M_n on input x. Choose $x \notin B_n$ with $2^{|x|} \geqslant 2^{2k^2+k} p^k(|x|)$.

Let $\mathscr{R}$ be the collection of sets of the form $D = B_n \cup C$, where

$$C \subseteq \{xy \mid |y| \leqslant |x|\},$$

and where there is some string z_D, $|z_D| = |x|$, such that

(2) $xy \in C$ implies y is a prefix of z_D, and

(3) if y is a proper prefix of z_D, then $[xy \in C$ iff $y1$ is a prefix of $z_D]$.

Claim. *There is a set $D \in \mathscr{R}$ such that either:*

(4) *$xz_D \in D$ and M_n on input x and oracle D rejects, or*

(5) *$xz_D \notin D$ and M_n on input x and oracle D accepts.*

Assuming that the claim is true, fix such a set D. Extend the definition of B to coincide with D on string xz_D and all its prefixes, and on all strings whose membership in D is queried during the computation of M_n on input x and oracle D. Further extend the definition of B so that B again becomes defined on an initial segment.

End

Then (3), (4) (5) and the definition of A are exactly what is needed to ensure that M_n with oracle B does not accept exactly the members of A.

The hard part, namely the verification of the claim, remains. We will simultaneously construct three sequences: $\mathcal{R}_0 \subseteq \mathcal{R}_1 \subseteq \cdots \subseteq \mathcal{R}_k$, a sequence of collections of sets, $x_0, x_1, \ldots, x_k$, a sequence of strings and $Q_1, \ldots, Q_k$, a sequence of sets of strings. Actually, for all j we will have

$$\mathcal{R}_j = \{D \in \mathcal{R} \mid x_j \text{ is a prefix of } z_D\}.$$

Let $\mathcal{R}_0 = \mathcal{R}$ (and $x_0 = \lambda$).

Roughly speaking, each Q_j will contain all queries generable using oracle sets in $\mathcal{R}_{j-1}$ and at most $j-1$ auxiliary oracle tapes. Then $\mathcal{R}_j$ will contain a sufficiently large number of the sets in $\mathcal{R}_{j-1}$, all of which agree on queries in Q_j.

Begin *Construction of Q_j.*

Fix any i.d. α for M_n and input x, any set T of indices of oracle tapes with $|T| \leq j-1$, any tape contents $(c_1, \ldots, c_k)$ for M_n having $c_i = \lambda$ for i not in T, and any oracle set $D \in \mathcal{R}_{j-1}$. Run M_n started as above just until Que_i is entered for some $i \notin T$ or until the computation terminates. If termination occurs, we make no additions to Q_j. However, if Que_i has been entered as above, we add the string then on tape i to Q_j.

End

Begin *Construction of $\mathcal{R}_j$ and x_j.*

Seek a string x_j such that:

(6) x_{j-1} is a prefix of x_j,

(7) $|x_j| \leq |x_{j-1}| + 2k + 1 + \log_2(p(|x|))$, and

(8) no string with prefix x_j is in Q_j.

Define $\mathcal{R}_j$ from x_j as indicated above.

End

The following assertions about the sequences will be verified inductively:

(9) For all j, $0 \leq j \leq k$, x_j exists and $|x_j| \leq j(2k + 1 + \log_2 p(|x|))$.

(10) For all j, $1 \leq j \leq k$, $|Q_j| \leq 2^{2k} p(|x|)$.

(11) For all j, $1 \leq j \leq k$, if $D, D' \in \mathcal{R}_j$ then $D \cap Q_j = D' \cap Q_j$.

Once these assertions have been verified, we see from (9) and the lower bound on $2^{|x|}$ that $|x_k| \leq |x|$. Thus by the relationship between $\mathcal{R}_k$ and x_k, there exist two sets D and D' in $\mathcal{R}_k$ having $z_D = z_{D'}$, $xz_D \in D$ and $xz_{D'} \notin D'$. Now if M_n is started with input x, the start i.d., $(\lambda, \ldots, \lambda)$ as tape contents and either oracle D or D', the construction of Q_k shows that each query generated on an oracle tape is in Q_k. But then by (11), the resulting answers to the queries are the same in the two computations. Thus, the two computations proceed through identical sequences of i.d.'s and therefore have the same outcome. But then either D satisfies (4) above or D' satisfies (5), as needed.

We turn now to the verification for (9)–(11). x_0 clearly satisfies (9). We now show (10) for Q_j.

Assume T and α are fixed ($2^k p(|x|)$ is an upper bound on the number of such choices). Assume $|T| \leq j-1$. Consider tape contents $(c_1, \ldots, c_k)$ and $(c_1', \ldots, c_k')$ for M_n, and oracles D and $D' \in \mathcal{R}_{j-1}$. Assume $c_i = \lambda$ and $c_i' = \lambda$ for i not in T. Assume M_n is started with input x, i.d. α, contents $(c_1, \ldots, c_k)$ and oracle D, or with the same input and i.d., contents $(c_1', \ldots, c_k')$ and oracle D'. Assume further that for each i, $1 \leq i \leq k$, the first answer to a query (if any) of tape i is the same in the two computations. (There are 2^k possible choices of such sets of answers.) Then the two computations cause the same string, if any, to get deposited into Q_j. This is because either there are no intervening queries (if $j = 1$), or all intervening queries made (after the first on each tape) are of strings in Q_{j-1} (if $j \neq 1$). But by (11) for $j-1$, the queries are answered in the same way in two computations.

Finally, we show (9) and (11) for x_j and $\mathcal{R}_j$, $j \geq 1$. Because of (10), we know that there must exist some x_j satisfying (6)–(8). But then (9) is clearly satisfied. Then by (8) and the definition of $\mathcal{R}$, (11) follows. □

4. A hierarchy based on the number of oracle tapes

In this section, we examine the relationships among the reducibilities $\leq^{\mathcal{L}(k)}$ for different values of k. As noted in the Introduction, a desirable property for a reducibility to be used for classification is transitivity. While it is unlikely that each reducibility $\leq^{\mathcal{L}(k)}$ is transitive, we prove that the natural generalization of the transitivity (proved in [9]) of $\leq^{\mathcal{L}(1)}$. Our result implies the transitivity $\leq^{\mathcal{L}(\omega)}$. Afterwards, we show proper containment of successive $\leq^{\mathcal{L}(k)}$ reducibilities.

Theorem 4.1. *For any sets A, B and C and integers $k, l \geq 0$, if $A \in \mathcal{L}(k)^B$ and $B \in \mathcal{L}(l)^C$, then $A \in \mathcal{L}(kl)^C$.*

Proof. Let M_1 be a k-oracle Turing machine which runs in log space and with oracle B computes membership in A. Fix a polynomial p such that M_1 runs in time p (see Theorem 3.1). Let M_2 be an l-oracle Turing machine which runs in log space and with oracle C computes membership in B. We construct a kl-oracle Turing machine M_3 which runs in log space and with oracle C computes membership in A.

M_3 is a Turing machine which simulates the action of three procedures, a main procedure MP and two mutually recursive procedures M_1SIM and M_2SIM. The input x and oracle set X are global. The key to the complexity bound is the fact that the maximum depth of procedure calls is $2k+1$.

MP, on x and X, simulates M_1 on x and oracle Y, where membership in Y is computed by M_2 with oracle X. The simulation is straightforward except that MP does not deposit bits on M_3's oracle tapes when M_1 makes such deposits. When M_1

reaches a query state Que_i, MP calls M_2SIM to compute the needed answer. MP passes to M_2SIM the number i of the queried oracle tape, the i.d. of M_1 following the *last*, if any, query of tape i (or the start i.d. if no such query exists), together with the *first following* answer, if any, to a query of each other oracle tape. (Here, "first following" refers to the first answer following that last query.) Using the answer returned by M_2SIM, MP continues its simulation of M_1 until M_1 halts with the needed answer. (The construction is essentially inductive on the number of oracle tapes of M_1, and is based on the following consideration. M_1 on x and X uses k oracle tapes. Consider times $t_1, t_2, \ldots, t_{k+1}$ during the computation, and assume a fixed tape is queried at time t_{k+1} and was last queried at time t_1. Assume that $t_2, \ldots, t_k$ are the first times after t_1 when each of the other tapes was queried. Then if the i.d. after time t_1 and the query answers obtained at times $t_2, \ldots, t_k$ are known, then in the interval from t_1 to t_{k+1} M_1 behaves as a $k-1$ tape machine only.)

M_2SIM simulates M_2 using X; the appropriate input for M_2 would appear on an oracle tape of M_1, so M_2SIM must find an alternative way to obtain this input. M_2SIM receives as parameters an oracle tape number i, an i.d. of M_1, and a length k vector of values in $\{0, 1, \not b\}$ ($\not b$ is the blank symbol). M_2SIM, during its simulation of M_2, knows at any time where M_2's input head would be. To obtain each needed input symbol, M_2SIM passes to M_1SIM all of its own parameters, together with the position of M_2's input head. M_1SIM returns either 0, 1 or # (the *endmarker*). Using this symbol, M_2SIM continues its simulation of M_2, until M_2 eventually halts with the required answer.

M_1SIM receives as parameters an oracle tape number i, an i.d. of M_1, a length k vector of values in $\{0, 1, \not b\}$ and a count n. If $n = 0$, M_1SIM returns #. Otherwise, M_1SIM simulates M_1 with x and X, starting at the given i.d., with oracle tape i blank, using the given vector for the set of *first* answers on all of M_1's other oracle tapes. For later answers, M_2SIM is called to compute the needed answers. M_1SIM passes to M_2SIM the same type of information as MP passes to M_2SIM. Using the answer returned by M_2SIM, M_1SIM continues its simulation of M_1. This simulation continues until M_1 deposits a bit on oracle tape i for the n^{th} time (in which case M_1SIM returns this bit) or until M_1 enters state Que_i (in which case M_1SIM returns #).

A key point to observe in the following slightly more detailed construction is that the set of first answers passed into a call of M_1SIM always includes an answer for every oracle tape that will be queried during the particular procedure call. Another key point is that the depth of procedure calls is bounded by $2k+1$. The kl oracle tapes are used by k successively nested calls of M_2.

Begin MP.

Local variables: $STORE_{i,}MAIN$, $STORE_{i,}j$, $1 \leq i, j \leq k$. (These will hold the "last i.d." and "first following answers" as described above.) For all i,

$STORE_{i,}MAIN :=$ the start i.d. for M_1 and x.

For all i, j, STORE$_i$.$j := \not b$.

Simulare M_1 on x and X. When a state Que$_i$ is entered, call

$$M_2\text{SIM}\ (i, \text{STORE}_i\text{.MAIN}, \text{STORE}_i\text{.}1, \ldots, \text{STORE}_i\text{.}k).$$

When M_2SIM returns 1 (yes) or 0 (no), change STORE$_i$.MAIN to the i.d. corresponding to this value, and STORE$_i$.j to $\not b$ for all j. Also, for all $j \neq i$, if STORE$_j$.$i = \not b$, then let STORE$_j$.i be assigned the value returned by M_2SIM. Finally, use the returned value to continue the simulation of M_1.

When M_1 accepts or rejects, do likewise.

End

Begin M_2SIM $(i, j, m_1, \ldots, m_k)$.

Parameters: i represents the number of an oracle tape of M_1. j is an i.d. of M_1. $m_1, \ldots, m_k$ are in $\{0, 1, \not b\}$.

Simulate M_2 on X. Use l oracle tapes when they are needed. When an input bit is required, call M_1SIM $(i, j, m_1, \ldots, m_k, n)$, where n is the position of M_2's input head. M_1SIM will return 0, 1 or $\#$. Use this symbol to continue the simulation of M_2.

When M_2 accepts or rejects, return 1 or 0 accordingly.

End

Begin M_1SIM $(i, j, m_1, \ldots, m_k, n)$.

Parameters: $i, j, m_1, \ldots, m_k$ as for M_2SIM above. n represents a position for M_2's input head.

Local variables: As for MP above.

For all q, STORE$_q$.Main $:= j$. (This initialization is for convenience only.) For all q, r, STORE$_q$.$r := \not b$.

If $n = 0$, return $\#$.

Otherwise, simulate M_1 with x and X, starting in i.d. j, with oracle tape i blank. When a state Que$_q$ is entered for the first time, consider the value of m_q. Change STORE$_q$.MAIN to this value and STORE$_q$.r to $\not b$ for all r. Also, for all $r \neq q$, if STORE$_r$.$q = \not b$, then let STORE$_r$.q be assigned the value in m_q. Finally, use m_q to continue the simulation of M_1. (Just for completeness, if $m_q = \not b$, the computation diverges. This will never occur during execution.)

When a state Que$_q$ is entered for the second or later time, call

$$M_2\text{SIM}\ (q, \text{STORE}_q\text{.MAIN}, \text{STORE}_q\text{.}1, \ldots, \text{STORE}_q\text{.}k).$$

When M_2SIM returns a value, change STORE$_q$.MAIN to that value, and STORE$_q$.r to $\not b$ for all r. Also for all $r \neq q$, if STORE$_r$.$q = \not b$, then let STORE$_r$.q be assigned the value returned by M_2SIM. Finally, use the returned value to continue the simulation of M_1.

The simulation of M_1 is continued until n bits are deposited on oracle tape i of M_1, or until M_1 enters state Que_i, whichever comes first. In the former case, M_1SIM returns the n^{th} bit. In the latter case, M_1SIM returns $\#$.

End

We leave the reader to convince himself that the constructed M_3 actually simulates the needed combination of M_1 and M_2. When M_1SIM is called, its parameters $m_1, \ldots, m_k$ actually provide the needed first answers, since the called execution of M_1SIM is only repeating a part of the simulation of M_1 already done and recorded by the next outer execution of M_1SIM (or of MP).

To show that the given workspace suffices, we now argue that the total depth of recursion at any time during the computation of M_3 is at most $2k+1$. Assume that some time during the computation of M_3 on input x and oracle X there is a sequence $c_1, \ldots, c_k$ of active successively nested calls of M_2SIM. Each call c_i was made in order to compute an answer to a query on some tape t_i of M_1, at some step s_i of the computation of M_1 on input x and the appropriate oracle. During call c_i, each time M_1SIM is called, it is from the i.d. of M_1 following the last query of tape t_i preceding step s_i (or from the start i.d. if no such query exists). Call the step of the above computation at which this i.d. was produced, step r_i.

Then $r_i < r_{i+1} < s_{i+1} < s_i$. This is because M_2SIM is called only after the *second* occurrence of a query of a particular tape during a particular call of M_1SIM.

Thus, $r_1 < \cdots < r_k < s_k < \cdots < s_1$.

Now it should be clear that during call c_k, no queries can arise causing a further call to M_2SIM. Thus, at most k calls to M_2SIM can be active at any one time, so that the total recursion depth is at most $2k+1$. □

We now use a combination of the proof techniques for Theorems 2.2 and 3.3 to show that the successive $\leqslant^{\mathcal{L}(k)}$ reducibilities form a hierarchy.

Theorem 4.2. *For any $k \geqslant 0$, there exist recursive sets A and B for which $A \leqslant^{\mathcal{L}(k+1)} B$ but $A \not\leqslant^{\mathcal{L}(k)} B$.*

Proof. The case for $k = 0$ is trivial, so we assume $k \geqslant 1$. B will be defined in stages; A will then be determined by

$$x \in A \quad \text{iff} \quad xy_1y_2 \cdots y_k \in B,$$

where $|y_i| = |x|$ for all i, and [the j^{th} bit of $y_i = 1$ iff $xy_1 \cdots y_{i-1}1^{j-1} \in B$, for all i, j]. It is not difficult to see that this relationship ensures that $A \leqslant^{\mathcal{L}(k+1)} B$.

Let $\{M_n\}$ be an effective enumeration of a sufficient set of k-oracle Turing machines (for our fixed value of k) which run in log space. Before stage n, an initial segment B_n of B will be defined; during stage n, the definition of B will be extended to ensure that M_n with oracle B does not accept exactly the members of A.

Before stage 0, define $\lambda \in B$.

Begin *stage n.*

Choose p to be a polynomial such that for all strings x, $p(|x|)$ is an upper bound on the number of i.d.'s for M_n and input x. Choose $x \notin E_n$ with $2^{|x|} > 2^{2k}p(|x|)$.

Let $\mathcal{R}$ be the collection of sets of the form $D = B_n \cup C$, where $C \subseteq \{xy \mid |y| \leq k|x|\}$ and where there exist strings $z_{D1}, \ldots, z_{Dk}$, $|z_{D1}| = \cdots = |z_{Dk}| = |x|$, such that:

(12) $xy \in C$ implies either $y = z_{D1} \cdots z_{Dk}$ or for some i $(0 \leq i \leq k-1)$ and j $(0 \leq j \leq |x|-1)$ $y = z_{D1} \cdots z_{Di}1^j$), and

(13) for all i, j as in (12), $xz_{D1} \cdots z_{Di}1^j \in C$ iff the $(j+1)^{st}$ bit of $z_{D(i+1)}$ is 1.

Claim. *There is a set $D \in \mathcal{R}$ such that either*

(14) $xz_{D1} \cdots z_{Dk} \in D$ *and M_n on input x and oracle D rejects, or*

(15) $xz_{D1} \cdots z_{Dk} \notin D$ *and M_n on input x and oracle D accepts.*

Then the construction is completed in a similar way to that for Theorem 3.3.

End

As before, it remains to verify the claim. We will simultaneously construct three sequences: $\mathcal{R}_0 \supseteq \cdots \supseteq \mathcal{R}_k$, a sequence of collections of sets, $x_1, \ldots, x_k$, a sequence of strings, and $Q_1, \ldots, Q_k$, a sequence of sets of strings. Actually, for all $j \geq 1$ we will have

$$Q_j = \{D \in \mathcal{R} \mid x_1 = z_{D1}, \ldots, x_j = z_{Dj}\},$$

so that we are only defining two nontrivial sequences.

Let $\mathcal{R}_0 = \mathcal{R}$.

The construction of Q_j is defined exactly as for Theorem 3.3, using M_n, x and $\mathcal{R}_{j-1}$ from the present theorem.

Begin *Construction of $\mathcal{R}_j$ and x_j.*

Seek x_j such that

(16) $|x_j| = |x|$, and

(17) no string with prefix $xx_1 \cdots x_j$ is in Q_j.

Define $\mathcal{R}_j$ from x_j as indicated above.

End

The following assertions will be used:

(18) For all j, $1 \leq j \leq k$, x_j exists.

(19) For all j, $1 \leq j \leq k$, $|Q_j| \leq 2^{2k}p(|x|)$.

(20) For all j, $1 \leq j \leq k$, if $D, D' \in \mathcal{R}_j$, then $D \cap Q_j = D' \cap Q_j$.

Once these assertions have been verified, the relationship between $\mathcal{R}_k$ and $x_1, \ldots, x_k$ shows that there exist two sets $D, D' \in \mathcal{R}_k$ having $z_{Di} = z_{D'i}$ for all i, $1 \leq i \leq k$, with $xz_{D1} \cdots z_{Dk} \in D$ and $xz_{D'1} \cdots z_{D'k} \notin D'$. If M_n is started on input x, the start i.d., $(\lambda, \ldots, \lambda)$ as tape contents and either oracle D or D', the construction of Q_k shows that each query generated is in Q_k. But then by (20), the two

computations have the same outcome, so that either D satisfies (14) or D' satisfies (15), as needed.

The verification of (18)–(20) is now similar to that of (9)–(11) of Theorem 3.3. □

5. Summary, further remarks and questions

Allowing more than one oracle tape has been shown to provide a more general log space reducibility than allowing only one. In fact, a hierarchy of reducibilities is generated as the number of oracle tapes is increased. This hierarchy is well behaved; it is properly included in polynomial time Turing reducibility, and a sort of transitivity result holds.

Since, for example, $\mathcal{NL}^A \subseteq \mathcal{NL}(\omega)^A$ for all A, results in [7] imply that the known result $\mathcal{NL} \subseteq \mathcal{P}$ does not relativize using the new definitions in this paper. We feel that this failure is due partly to the lack of a satisfactory notion of relativized space, but is also due partly to the indirect nature of the simulation used to show $\mathcal{NL} \subseteq \mathcal{P}$. General simulations should probably not be so readily discarded as possible methods for examining basic open questions.

We have not examined in detail any properties of the defined nondeterministic reducibilities. It has been shown by Ladner [10] that:

Proposition 4.3. *There exists a recursive set B such that $\mathcal{P}^B$ is a proper subset of $\mathcal{NL}(2)^B$.*

We do not know, for example, if there exists a recursive set B such that $\mathcal{P}^B$ is not a subset of $\mathcal{NL}(2)^B$. Many other technical questions will no doubt suggest themselves; a few carefully chosen ones may be worth pursuing for the insights they may give into the limits on the power of log space machines.

We have also not examined the analogous definitions and constructions for space-bounded reducibilities defined by larger space bounds.

References

[1] T. Baker, J. Gill and R. Solovay, Relativizations of the $\mathcal{P} =^? \mathcal{NP}$ question, *SIAM J. Comput.* 4 (1975), 431–442.

[2] T. Baker and A. Selman, A second step toward the polynomial hierarchy, *Proc. 17th Ann. IEEE Symp. Foundations Comput. Sci.*, 71–75.

[3] S. Cook, The complexity of theorem-proving procedures, *Proc. 3rd Ann. ACM Symp. Theory Comput.* (1971), 151–158.

[4] J. Hopcroft, W. Paul and L. Valiant, On time versus space and related problems, *Proc. 16th Ann. IEEE Symp. Foundations Comput. Sci.*, 57–64.

[5] N. Jones and W. Laaser, Complete problems for deterministic polynomial time, *Proc. 6th Ann. ACM Symp. Theory Comput.* (1974) 40–46.

[6] R. Karp, Reducibility among combinatorial problems, in: R. Miller and J. Thatcher, eds., *Complexity of Computer Computations* (Plenum, New York, 1972).

[7] R. Ladner and N. Lynch, Relativization of questions about log space computability, *Math. Syst. Theory* **10** (1976) 19–32.

[8] R. Ladner, N. Lynch and A. Selman, A comparison of polynomial time reducibilities, *Theoret. Comput. Sci.* **1** (1975) 103–123.

[9] R. Ladner, On the structure of polynomial time reducibility, *J. Assoc. Comput. Mach.* **22** (1975) 155–171.

[10] R. Ladner, private communication.

[11] N. Lynch, Relativization of the theory of computational complexity, *Trans. Amer. Math. Soc.* **220** (1976), 243–287.

[12] A. Meyer and L. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, *Proc. 13th Ann. IEEE Symp. Switching and Automata Theory* (1972), 125–129.

[13] M. Paterson, Tape bounds for time-bounded Turing Machines, *J. Comput. System Sci.* **6** (1972) 116–124.

[14] H. Rogers, *Theory of Recursive Functions and Effective Computability* (McGraw-Hill, New York, 1967).

[15] L. Stockmeyer and A. Meyer, Word problems requiring exponential time, preliminary report, *Proc. 5th Ann. ACM Symp. Theory of Computing* (1973) 1–9.

[16] W. Savitch, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System. Sci.* **4** (1970) 177–192.

[17] I. Simon, On some subrecursive reducibilities, Computer Science Dept., Stanford University, Ph.D. dissertation (1977).

[18] I. Simon and J. Gill, Polynomial reducibilities and upward diagonalizations, *Proc. 9th Ann. ACM Symp. Theory of Computing* (1977).

[19] D. Symes, The extension of machine-independent computational complexity theory to oracle machine computation and to computation of finite functions, Dept. of Applied Analysis and Computer Science, University of Waterloo, Ph.D. thesis (October 1971).