JOURNAL OF COMPUTER AND SYSTEM SCIENCES 11, 68–85 (1975)

Space-Bounded Reducibility among Combinatorial Problems*

NEIL D. JONES

Computer Science Department, The University of Kansas, Lawrence, Kansas 66045

Received January 25, 1974; revised October 14, 1974

Reducibility and completeness among combinatorial problems can be formulated in terms of space bounds, in some cases refining the polynomial time-reducibility of Cook and Karp. Two versions are defined, by means of Turing machines and by bounded-quantifier formulas. Following are the main results. (1) The problem "Is there a path between specified nodes of a digraph?" is shown to be complete for the sets acceptable in nondeterministic log($\cdot$) space; (2) The problem "Given a finite function $f: \{1,..., n\} \to \{1,..., n\}$, is there a k such that $f^k(1) = n$?" is similarly complete for deterministic log($\cdot$) space; and (3) Each of the problems "Is $T(M) = \varnothing$?" and "Is $T(M)$ infinite?" (for M a deterministic finite automaton) is shown to be complete for nondeterministic (deterministic) log n space in the case in which the alphabet of M is arbitrary (consists of one letter).

1. INTRODUCTION

Considerable interest has been shown recently in a reducibility approach to the complexity of combinatorial problems. The problems are normally formulated in terms of recognition of sets over an alphabet Σ. In its most general form, the reducibility is merely a reflexive, transitive relation $\leqslant$ between languages over Σ. Two languages $L, L' \subseteq \Sigma^*$ are *equivalent* iff $L \leqslant L'$ and $L' \leqslant L$; the conditions on $\leqslant$ cause this to be an equivalence relation.

Thus mutual reducibility breaks the class of all languages into equivalence classes. A single language L_0 is said to be $\leqslant$*-complete* for a class of languages $\mathscr{L}$ iff L_0 is in $\mathscr{L}$, and $L \leqslant L_0$ holds for every language L in $\mathscr{L}$. Note that any two $\leqslant$-complete languages for $\mathscr{L}$ must be equivalent.

Cook [2] and Karp [6] initiated this approach in the context of automata theory by defining two types of reducibility in terms of polynomially time-bounded Turing machines. Karp's reducibility corresponds to the "many–one" reducibility of recursive

* A preliminary version of this paper, written while the author was at The Pennsylvania State University, appeared in the Proceedings of the Seventh Princeton Conference on Information Science and Systems.

functions: L is reducible to L' iff there is a polynomial-time computable function $f(\cdot)$ such that $x \in L$ iff $f(x) \in L'$, for all $x \in \Sigma^*$. Cook's version is analogous to Turing reducibility, expressed in terms of "oracle machines."

They showed that a wide collection of familiar combinatorial problems are complete (and thus equivalent) for the class NP of languages nondeterministically recognizable in polynomial time. These problems include satisfiability of propositional formulas; graph theoretic problems relating to the existence of colorations, cliques, covers, circuits, etc.; and other problems related to matching, partitions, and integer programming.

These problems have been investigated in many contexts. None is known to have a polynomial-time solution, in spite of the efforts of many workers. Thus a large number of problems which have proved *practically* intractable, seeming to require exhaustive methods involving exponential time, were seen to be very closely related to some *theoretically* intractable problems involving recognition by nondeterministic Turing machines, which also seem to require exhaustive methods. This strengthened considerably the connection between the theoretical study of automata and the practical study of algorithms.

Further, many open questions in automata theory are of the form: Given language classes $\mathscr{L}$ and $\mathscr{L}'$ such that $\mathscr{L} \subseteq \mathscr{L}'$, is the inclusion proper? Supposing that $\leqslant$ preserves membership in $\mathscr{L}$ (i.e., $L \leqslant L'$ and $L' \in \mathscr{L}$ implies $L \in \mathscr{L}$), and supposing that L_0 is $\leqslant$-complete for $\mathscr{L}'$, we see that $\mathscr{L} = \mathscr{L}'$ if and only if $L_0 \in \mathscr{L}$. In other words, any $\leqslant$-complete language in $\mathscr{L}'$ can serve as a representative of its entire class, with respect to the question of proper containment of $\mathscr{L}$.

A wide variety of problems have since been classified using this approach. Examples include problems concerning incomplete finite automata (Pfleeger [9]), regular expressions (Meyer [8], Hunt [3]), and problems from logic (Stockmeyer [12]).

It is possible, however, that the polynomial complete class is still too large to model practical computability. Certainly, any practical algorithm works in time bounded by a polynomial in the size of its data; however, many polynomial time algorithms are too complex for real problems. In addition a polynomial *space* bound is allowed, which certainly seems excessive as compared to practical algorithms.

However, *all* problems which can be solved deterministically in polynomial time are trivially equivalent by the Cook and Karp polynomial time reducibilities. It thus appears desirable to develop reducibility notions which refine polynomial time reducibility, so that smaller nontrivial problem classes may be studied. In particular some of the many open questions in this field may prove more susceptible to solution by consideration of special cases of very low complexity.

This paper introduces two notions of reducibility in terms of space bounds on computations, one in terms of Turing machines and the other in terms of rudimentary predicates. In case of a logarithmic space bound, both are refinements of Karp's polynomial time reducibility. Various mathematical properties of these reducibilities are established, and several problems are presented which are complete for the sets recognizable in nondeterministic logarithmic space. An interesting example is the graph accessibility problem, which is essentially the "threadable maze" problem of Savitch expressed in more usual terminology.

The initial motivation for this approach came as a result of this writer's observation that the reductions in [2] and [6] could be done in logarithmic space. This fact was also noticed independently by Stockmeyer and Meyer [12] at about the same time. Their emphasis, however, is in the direction of higher complexity; in addition they did not consider reducibility in terms of general space bounds, or the space-bounded rudimentary reducibility.

2. Reducibility via Space-Bounded Turing Machines

By a *Turing machine* we mean a device with a finite-state control, a two-way read-only input tape with endmarkers, and a single two-way read–write work tape, which halts whenever it enters a final, or accepting, state. The machine is assumed to be deterministic unless otherwise specified; detailed definitions may be found in [4].

Throughout this paper Σ will denote an alphabet with at least two symbols, and $S(\cdot)$ and $T(\cdot)$ will denote nonzero functions on the natural numbers not smaller than the function $\log(\cdot)$. A Turing machine Z *operates in space* $S(\cdot)$ iff, whenever it is started with any $x \in \Sigma^*$ on its input tape, it will never scan more that $S(|x|)$ symbols on its work tape (where $|x|$ denotes the number of symbols in x). In addition we require that Z halt for all inputs. The class DSPACE_S is defined to be the collection of all languages which are recognizable by deterministic Turing machines which operate in space $S(\cdot)$; NSPACE_S is defined analogously, for nondeterministic Turing machines.

Let $f: \Sigma^* \to \Sigma^*$ be a function. We define $f(\cdot)$ to be *computable in space* $S(\cdot)$ iff there is a Turing machine, Z, as above but additionally equipped with a write-only one-way output tape, such that if Z is given any input $x \in \Sigma^*$, Z will eventually halt with $f(x)$ on its output tape, having scanned no more than $S(|x|)$ work tape symbols.

Note that the size of $f(x)$ is not explicitly restricted; however for each Z there must be a constant k such that $|f(x)| \leqslant k^{S(|x|)}$ for all inputs $x \in \Sigma^*$, because of the fact that Z must halt and so cannot repeat a configuration.

Definition 1. Let $L, L' \subseteq \Sigma^*$. Then $L \leqslant_S^{\text{tms}} L'$ (L *is Turing-reducible to* L' *in space* $S(\cdot)$) is true iff there is a function $f: \Sigma^* \to \Sigma^*$ such that

1. $\forall x \in \Sigma^*, x \in L \Leftrightarrow f(x) \in L'$;
2. f is computable in space $S(\cdot)$; and
3. There is a positive integer c such that $\forall x \in \Sigma^*$, $S(|f(x)|) \leqslant c \cdot S(|x|)$.

A single construction is used to prove each of the following. The proofs follow the construction.

THEOREM 2. *The relation* $\leqslant_S^{\text{tms}}$ *is reflexive and transitive.*

LEMMA 3. *Let* L, L', *and* f *satisfy conditions* 1 *and* 2 *of Definition* 1, *and suppose there is a positive integer* c *such that*

$$\forall x \in \Sigma^*, \qquad T(|f(x)|) \leqslant c \cdot S(|x|).$$

Then $L' \in \mathrm{DSPACE}_T(\mathrm{NSPACE}_T)$ *implies that* $L \in \mathrm{DSPACE}_S(\mathrm{NSPACE}_S)$.

THEOREM 4. *If* $L \leqslant_S^{\text{tms}} L'$ *and* $L' \in \mathrm{DSPACE}_S(\mathrm{NSPACE}_S)$, *then*

$$L \in \mathrm{DSPACE}_S(\mathrm{NSPACE}_S).$$

CONSTRUCTION. Let $f: \Sigma^* \to \Sigma^*$ be computed in space $S(\cdot)$ by a given Turing machine Z, and let Z' be another Turing machine (possibly nondeterministic or with an output tape, but not both), which operates in space $T(\cdot)$. A Turing machine Z'' is constructed, as follows.

When given an input $x \in \Sigma^*$, Z'' will simulate the effect of machine Z' as applied to input $f(x)$. This is not done directly (by first computing $f(x)$ and then applying Z'), since $f(x)$ may be too large to store within space $S(|x|)$. Rather, a submachine Z_0 is utilized, which when given any integer i $(0 \leqslant i \leqslant |f(x)| + 1)$, will return the ith symbol of $\$f(x)¢$, thus simulating the effect of reading the kth symbol of the input tape of Z'.

It is clear that Z_0 may be effectively constructed to simulate Z on x, so that it counts the number of input symbols which Z would have produced so far. When this count reaches $i - 1$, the appropriate symbol is produced and Z_0 returns to the machine which called it.

The operation of Z'' is now straightforward. Z'' will have four tracks on its work tape, as follows.

Track 1: The work tape of Z for input x;

Track 2: The work tape of Z' for input $f(x)$;

Track 3: i, the scanning position of Z' on its (simulated) input tape $\$f(x)¢$, in dyadic notation;

Track 4: A counter j, used by Z_0 to count the number of (simulated) output symbols produced so far by Z on x.

In order to simulate one step of Z' on $f(x)$, Z'' will do the following.

(a) Find the ith symbol, a, of $\$f(x)¢$ (using Z_0 and tracks 1, 3, and 4);

(b) Find the current work tape symbol, b, of Z' on track 2;

(c) Using a, b, and the current state of Z', determine the next action of Z'; and

(d) perform that action (this consists of writing and updating the head position on the work tape via track 2, and updating the input position (on $\$f(x)¢$) by incrementing or decrementing the value of i on track 3).

Clearly Z'' may be effectively constructed to behave as specified. We now analyze the amount of tape used by Z''.

Track 1: Size is $\leqslant S(|x|)$, since f is computed by Z in space S;

Track 2: Size is $\leqslant T(|f(x)|)$, since Z' operates in space $T(\cdot)$;

Track 3: Size is $\leqslant \log|f(x)| \leqslant \log k^{S(|x|)} = S(|x|)\log k$, where k depends only on Z;

Track 4: Same as track 3.

Thus Z'' uses at most space $S(\cdot)$ on tracks 1, 3, and 4, and space $T(|f(\cdot)|)$ on track 2.

Proof of Theorem 2. Clearly $L \leqslant_S^{\text{tms}} L$ for any $L \subseteq \Sigma^*$, since the identity function is computable in zero space.

To show transitivity, suppose $L \leqslant_S^{\text{tms}} L'$ and $L' \leqslant_S^{\text{tms}} L''$ as in Definition 1, by virtue of functions f and f', Turing machines Z and Z', and constants c and c', respectively. Then for any $x \in \Sigma^*$,

$$S(|f'(f(x))|) \leqslant c' \cdot S(|f(x)|) \leqslant c' \cdot c \cdot S(|x|).$$

So $c'c$ is a suitable constant for the function $f'f(\cdot)$.

It must now be shown that $f'f(\cdot)$ is computable in space $S(\cdot)$. Let Z'' be the Turing machine of the construction above. Clearly Z'' operates in space bounded by

$$\begin{aligned}\max(S(|x|), \log k \cdot S(|x|), S(|f(x)|)) &\leqslant \max((1+\log k)\cdot S(|x|), c \cdot S(|x|))\\ &\leqslant \max(1+\log k, c)\cdot S(|x|).\end{aligned}$$

By standard results [4] there is an equivalent machine which operates in space $S(\cdot)$. ▮

Proof of Lemma 3. Let Z compute $f(\cdot)$ in space $S(\cdot)$, and let Z' recognize L' in space $T(\cdot)$. Let Z'' be the Turing machine of the construction. Then Z'' operates in space bounded by

$$\begin{aligned}\max(S(|x|), \log k \cdot S(|x|), T(|f(x)|)) &\leqslant \max((1+\log k)\cdot S(|x|), c \cdot S(|x|))\\ &\leqslant \max(1+\log k, c)\cdot S(|x|). \quad ▮\end{aligned}$$

Proof of Theorem 4. This is immediate from the preceding Lemma, with $S(\cdot) = T(\cdot)$.

Remarks. The transitivity of $\leqslant_S^{\text{tms}}$ implies that mutual reducibility between languages is an equivalence relation, thus providing a framework for the comparison of languages, and making the concept of a complete language meaningful in the context of space bounds. In Section 4 several sets will be exhibited which are complete for space-bounded reducibility.

The special case $S(\cdot) = \log(\cdot)$ is of particular interest. Any set in $\text{NSPACE}_{\log}$ is deterministically recognizable in polynomial time (and space). The reason is that if Z operates in $\log(\cdot)$ space, there is a constant k such that Z can have at most $k^{\log n} = n^{\log k}$ distinct configurations for an input of length n. These can be listed exhaustively, and recognition can be tested, in polynomial time. In addition, any function computable in $\log(\cdot)$ space is an appropriate reduction function according to Definition 1, since $\log(|f(x)|) \leqslant \log(k^{\log|x|}) = \log k \cdot \log |x|$.

Clearly any such function is computable in polynomial time, so we see that $\leqslant_{\log}$-reducibility is a refinement of the polynomial time reducibility of Karp [6]. A space-bounded analog of Cook's version of reducibility [2] could easily be obtained by modifying his "query machine" to have a read-only input tape and a write-only query tape. The only problem is that it is not apparent that Theorem 4 holds for NSPACE_S when one uses this type of reducibility. We use our version rather than those of Karp or Cook since it is simpler, and all known reducibility algorithms (in this area) can be expressed in this form.

3. Reducibility via Bounded-Quantifier Formulas

We now define another reducibility relation, $\leqslant_S^{\text{rud}}$, which is a refinement of the relation $\leqslant_S^{\text{tms}}$. A consequence of introducing $\leqslant_S^{\text{rud}}$ is that it now becomes possible to discuss nontrivial languages which are complete for *deterministic* space-bounded classes. This reducibility is defined by a space-bounded analog of the rudimentary predicates of Smullyan [10].

(Note: An n-ary predicate is a relation on n variables, and can be viewed as a subset of $(\Sigma^*)^n$. For $n = 1$, a predicate is the same as a language. Smullyan uses the term "attribute.")

The $S(\cdot)$-bounded rudimentary predicates bear the same relation to DSPACE_S as the rudimentary predicates bear to the languages accepted by deterministic linear-bounded automata (i.e., DSPACE_n). This last relationship was studied by Myhill [7], who showed that every rudimentary language is in DSPACE_n ; the converse question has remained open for many years.

The $S(\cdot)$-bounded rudimentary predicates are defined by formulas involving logical operators and $S(\cdot)$-bounded quantification. Each predicate is of the form

$Q(x, y_1, \ldots, y_n)$, and is constrained so that, when true, every variable except x is bounded in length by a constant multiple of $S(|x|)$. In effect x corresponds to the read-only input tape of a Turing machine; the basis predicate $\sigma(x, i, a)$ intuitively "reads" the ith symbol of x.

In the following $\mathbf{y}_n$ serves as an abbreviation for the sequence $y_1, y_2, \ldots, y_n$. Two symbols, 1 and 2, will be distinguished in the alphabet Σ. Any variable (other than x) whose value is in $\{1, 2\}^*$ may be considered to be either a string, or a numerical value expressed in dyadic notation. For example the i in $\sigma(x, i, a)$ is considered numerically.

Definition 5. The ternary predicate $\sigma(x, i, a)$ is defined to be true iff $a = a_i$, where $x = a_1 \cdots a_n$ for symbols $a_1, \ldots, a_n$ in Σ. (Note: $\sigma(x, i, a)$ is false if $a \notin \Sigma$, or $i \notin \{1, 2\}^+$, or $i > |x|$.)

Definition 6. The class of *$S(\cdot)$-bounded rudimentary* predicates, denoted by RUD_S, is the smallest class of predicates such that

1. $\sigma(x, i, a)$ is in RUD_S;
2. If c is a positive integer, then the predicate $Q(x, u, v, w)$ defined by

$$uv = w \wedge |uvw| \leqslant c \cdot S(|x|)$$

is in RUD_S (notation: $P \wedge Q$, $P \vee Q$, $P \overset{\Rightarrow}{=} Q$, and $\sim P$ are read as P *and* Q, P *or* Q, P *implies* Q, and *not* P, respectively);

3. If $Q(x, \mathbf{y}_n)$ and $R(x, \mathbf{y}_n)$ are in RUD_S, then so are $Q(x, \mathbf{y}_n) \wedge R(x, \mathbf{y}_n)$, $Q(x, \mathbf{y}_n) \vee R(x, \mathbf{y}_n)$ and $\sim Q(x, \mathbf{y}_n)$;

4. If $Q(x, \mathbf{z}_m)$ is in RUD_S and each of $\xi_1, \ldots, \xi_m$ is either a string in Σ^* or one of the variables $y_1, \ldots, y_n$, then the predicate R defined by

$$R(x, \mathbf{y}_n) \Leftrightarrow Q(x, \boldsymbol{\xi}_m)$$

is in RUD_S (this corresponds to the "explicit transformation" of [10]; it allows variables to be permuted, identified, introduced, or replaced by constants);

5. If c is a positive integer, and $Q(x, \mathbf{y}_{n+1})$ is in RUD_S, then the predicates R and R' defined as follows are also in RUD_S.

$$R(x, \mathbf{y}_n) \Leftrightarrow \exists z(|z| \leqslant c \cdot S(|x|) \wedge Q(x, \mathbf{y}_n, z)),$$

$$R'(x, \mathbf{y}_n) \Leftrightarrow \forall z(|z| \leqslant c \cdot S(|x|) \rightarrow Q(x, \mathbf{y}_n, z)).$$

Notation. The right sides of the last two formulas will be abbreviated as $(\exists z)_{cS}\, Q(x, \mathbf{y}_n, z)$ and $(\forall z)_{cS}\, Q(x, \mathbf{y}_n, z)$, respectively. $F \Rightarrow G$ (F implies G) and

$F \Leftrightarrow G$ (*F if and only if G*) will be used as abbreviations for $\sim F \wedge G$ and $F \wedge G \vee \sim F \wedge \sim G$, respectively.

The following relate this definition to the usual rudimentary predicates [2, 5, 10].

THEOREM 7. *If $Q(\mathbf{y}_n)$ is rudimentary and c is a positive integer, then the predicate R defined by*

$$R(x, \mathbf{y}_n) \Leftrightarrow Q(y_n) \wedge |y_1 y_2, \dots, y_n| \leqslant c \cdot S(|x|)$$

is in RUD_S.

(Note: RUD_n denotes RUD_S where $S(n) = n$ is the identity function.)

THEOREM 8. *Any unary predicate $Q(x)$ in* RUD_n *is rudimentary, and conversely.*

Proofs are straightforward, by induction on the number of applications of the operations in Definition 6.

The following is proved by a similarly straightforward but tedious induction, involving the construction of space-bounded Turing machines. The technique is essentially that of Myhill [7]. A function $S(\cdot)$ is said to be *constructable* if there is a deterministic Turing machine with a one-symbol input alphabet which operates in space $S(\cdot)$ but not in any smaller space bound [4].

THEOREM 9. *Let $Q(x, \mathbf{y}_n)$ be in* RUD_S *where $S(\cdot)$ is constructable, and let $\#$ be a symbol not in Σ. Then*

$$\{x \# y_1 \# \cdots \# y_n \mid Q(x, \mathbf{y}_n) \text{ is true}\}$$

is a language in DSPACE_S.

COROLLARY 10. *Any language in* RUD_S *is also in* DSPACE_S.
The converse to this corollary is an open question.

DEFINITION 11. A function $f: \Sigma^* \to \Sigma^*$ is *$S(\cdot)$-bounded rudimentary* iff the predicate R defined as follows is in RUD_S.

$$R(x, i, a) \Leftrightarrow \sigma(f(x), i, a).$$

(Notes: f is $S(\cdot)$-bounded rudimentary iff the predicate "a is the ith symbol of $f(x)$" is in RUD_S. The fact that $|i|$ cannot exceed $k \cdot S(|x|)$ for some k and all x implies that $|f(x)| \leqslant k^{S(|x|)}$.)

THEOREM 12. *If $S(\cdot)$ is constructable, then any $S(\cdot)$-bounded rudimentary function is computable in space $S(\cdot)$.*

Proof. Let $\Sigma = \{a_1, a_2, ..., a_k\}$, and suppose $f\colon \Sigma^* \to \Sigma^*$ is $S(\cdot)$-bounded rudimentary. By Theorem 9 there is a deterministic Turing machine Z which recognizes $\{x \# i \# a \mid \sigma(f(x), i, a) \text{ is true}\}$. Construct a Turing machine Z' to operate as follows.

1. Let $i = 1$ (i is represented in dyadic notation);

2. Given x, evaluate $\sigma(f(x), i, a)$ for $a = a_1, a_2, ...$ until some $\sigma(f(x), i, a_j)$ is true, or until $\sigma(f(x), i, a_k)$ is seen to be false;

3. In the first case write a_j, set $i = i + 1$ and go to step 2;

4. Otherwise halt.

It should be clear that Z' writes out the symbols of $f(x)$, one at a time, and halts after writing the last symbol.

It remains to show that Z' operates in space $S(\cdot)$. The only storage used (other than that which Z uses) is for i and a, and a only assumes values in the fixed set Σ. Now $\sigma(f(x), i, a)$ is in RUD_S, so there is a constant c such that $|i| \leqslant c \cdot S(|x|)$ whenever $\sigma(f(x), i, a)$ is true. Thus Z' never causes i to exceed $c \cdot S(|x|) + 1$ in length, so an equivalent machine exists which operates in space $S(\cdot)$. ∎

DEFINITION 13. Let $L, L' \subseteq \Sigma^*$. Then $L \leqslant_S^{\mathrm{rud}} L'$ (L is *$S(\cdot)$-bounded rudimentary reducible to L'*) iff there is a function $f\colon \Sigma^* \to \Sigma^*$ such that

1. $\forall x \in \Sigma^*,\ x \in L \Leftrightarrow f(x) \in L'$;
2. f is $S(\cdot)$-bounded rudimentary; and
3. There is a positive integer c such that $\forall x \in \Sigma^*,\ S(|f(x)|) = c \cdot S(|x|)$.

LEMMA 14. *Let $f\colon \Sigma^* \to \Sigma^*$ be an $S(\cdot)$-bounded rudimentary function, and let $Q(x, \mathbf{y}_n)$ be a predicate in RUD_T. Suppose there is a positive integer c such that*

$$\forall x \in \Sigma^*, \qquad T(f(x)) = c \cdot S(|x|).$$

Then the predicate $Q(f(x), \mathbf{y}_n)$ is in RUD_S.

Proof. Let $R(x, i, a)$ be in RUD_S according to Definition 6, and let F be a formula which shows that Q is in RUD_T. A new formula F' (with variables $x, y_1, ..., y_n$) which shows that $Q(f(x), \mathbf{y}_n)$ is in RUD_S may be obtained as follows.

(i) Replace every occurrence in F of a subformula $\sigma(x, j, b)$ by the formula for $R(x, j, b)$; and

(ii) Replace every quantifier of the form $(\exists z)_{dT}$ or $(\forall z)_{dT}$ by $(\exists z)_{c \cdot dS}$ or $(\forall z)_{c \cdot dS}$, respectively.

Now F' certainly defines a predicate in RUD_S. Proof that it defines $Q(f(x), \mathbf{y}_n)$ is straightforward, by induction on the structure of F. Conceptually, in step (i) all we have done is to replace each reference to the jth symbol of x by a reference to the jth symbol of $f(x)$. The condition that $T(|f(x)|) = c \cdot S(|x|)$ ensures that for any quantified variable z and constant d,

$$|z| \leqslant d \cdot T(|f(x)|) \quad \text{iff} \quad |z| \leqslant c \cdot d \cdot S(|x|).$$

Thus the quantifiers of F' are bounded appropriately. ▌

Corollary 15. *$\leqslant_S^{\mathrm{rud}}$ is reflexive and transitive.*

Proof. The identity function is certainly $S(\cdot)$-bounded rudimentary, so $\leqslant_S^{\mathrm{rud}}$ is reflexive. Now suppose $L \leqslant_S^{\mathrm{rud}} L'$ and $L' \leqslant_S^{\mathrm{rud}} L''$ as in Definition 13, by virtue of functions f and f', predicates $R(x, i, a)$ and $R'(x, i, a)$, and constants c and c', respectively. Clearly $R'(f(x), i, a)$ is a predicate in RUD_S by the previous lemma, so that $f'(f(\cdot))$ is an S-bounded rudimentary function. Further, for any $x \in \Sigma^*$,

$$S(|f'(f(x))|) = c' \cdot S(|f(x)|) = c' \cdot c \cdot S(|x|).$$

Thus $f'f(\cdot)$ and $c'c$ satisfy Definition 13, so $L \leqslant_S^{\mathrm{rud}} L''$. Thus $\leqslant_S^{\mathrm{rud}}$ is transitive. ▌

Corollary 16. *The relation $\leqslant_S^{\mathrm{rud}}$ is a refinement of $\leqslant_S^{\mathrm{tms}}$.*

Consequently $\leqslant_S^{\mathrm{rud}}$ provides a (possibly) finer classification than that provided by the Turing machine reducibility $\leqslant_S^{\mathrm{tms}}$. The stronger requirement that $S(|f(x)|) = c \cdot S(|x|)$, that is, "$=$" rather than "$\leqslant$", seems to be necessary in order to prove Lemma 14, for otherwise the quantifier ranges of F' might differ from those of F, rendering F and F' nonequivalent.

Bounded Rudimentary Turing Machine Descriptions

In preparation for the main results we show that RUD_S contains several predicates useful for describing computations by Turing machines. As before, Σ is an alphabet containing two distinguished symbols, 1 and 2, and $S(\cdot) \geqslant \max(1, \log(\cdot))$.

Lemma 17. *Let Z be a (possibly nondeterministic) Turing machine with input Σ. There is an encoding scheme which represents each instantaneous description (or* ID *for short) of Z by a corresponding string $\alpha \in \{1, 2\}^+$, and a positive integer c (independent of α), such that*

(a) *Different* ID's *have different encodings*;

(b) *For each $x \in \Sigma^*$, if α is the encoded form of an* ID *for x whose work tape contains at most $S(|x|)$ symbols, then $|\alpha| \leqslant c \cdot S(|x|)$; and*

(c) *The following predicates are all in* RUD_S :

1. ID(x, α), *true iff* $|\alpha| \leqslant c \cdot S(|x|)$ *and* α *is the encoded form of an* ID,
2. INIT(x, α), *true iff* $|\alpha| \leqslant c \cdot S(|x|)$ *and* α *is the encoded form of the initial* ID *of* Z *for the input* x,
3. FINAL(x, α), *true iff* $|\alpha| \leqslant c \cdot S(|x|)$, *and* α *is an encoded accepting* ID *for* x,
4. YIELD(x, α, β), *true iff* $|\alpha| \leqslant c \cdot S(|x|)$, $|\beta| \leqslant c \cdot S(|x|)$, *and* Z *yields* β *from* α *in one step while scanning* x.

Proof. Z may be easily modified to use $\{1, 2\}$ as its work tape alphabet, by multiplying its tape size by a constant factor, c_0. Let the states of Z be $q_1, ..., q_k$. An ID may be uniquely represented by a string $\alpha \in \{1, 2\}^*$ of the form

$$\alpha = i \triangle p_I \triangle p_W \triangle z,$$

where z is the current nonblank work tape contents of Z, and i, p_I, and p_W are dyadic representations of the current state, and the input tape and work tape scanning positions, respectively. The separator $\triangle$ is the string $21^{c_0 \cdot S(|x|)}2$. Clearly α can be uniquely decoded to obtain i, p_I, p_W, and z.

Suppose α is an encoded ID for input x. Then $|z| \leqslant c_0 \cdot S(|x|)$, $i \leqslant k$, $p_I \leqslant |x| + 2$, and $p_W \leqslant S(|x|)$. Thus

$$\begin{aligned}|\alpha| &= 3|\Delta| + |i| + |p_I| + |p_W| + |z| \\ &\leqslant 3|\Delta| + \log k + \log(|x| + 2) + \log S(|x|) + c_0 \cdot S(|x|) \\ &\leqslant 3|\Delta| + \log k + c_1 \cdot S(|x|) + S(|x|) + c_0 \cdot S(|x|) \\ &\leqslant cS(|x|).\end{aligned}$$

For appropriate c_1 and c, independent of x.

The predicate $|\alpha| \leqslant c \cdot S(|x|)$ is trivially in RUD$_S$, since for any x, α

$$|\alpha| \leqslant c \cdot S(|x|) \Leftrightarrow (\exists\beta)_{cS} (\beta = \alpha).$$

From this point standard rudimentary techniques (such as those of [5, 10]) may be applied to show that ID, INIT, FINAL, and YIELD are in RUD$_S$. The method is straightforward so details are omitted. ∎

Note that each ID α is a string over $\{1, 2\}$, and so may be regarded as a dyadic integer. Thus $\alpha < 2^{c \cdot S(|x|)} + 2$ holds numerically for any ID α resulting from x.

4. Some Log-SPACE Complete Problems

In this section several problems are shown to be $\leqslant_{\log}^{\text{rud}}$-complete for DSPACE$_{\log}$ or NSPACE$_{\log}$. As with the case of the polynomial time reducibility of Karp [6], the log$(\cdot)$-bounded rudimentary predicates provide enough power to translate from one problem representation to another within a rather wide range. The chief reason is that the relations $u + v = w$, $u \cdot v = w$, $u^v = w$, and $|u| = v$ are all rudimentary by [1]; consequently by Theorem 7 the same relations are in $\mathrm{RUD}_{\log}$, as long as u, v, w are constrained to be no longer than a constant multiple of $\log |x|$. This constraint turns out to be easily satisfied for normal changes in representation, such as graph representation by means of adjacency matrices, incidence matrices, and lists of adjacent vertex pairs.

Graph Accessibility Problems

Each directed graph, G, discussed will be assumed to have vertex set $V_n = \{1, 2,..., n\}$ for some n. Its structure will be represented by an adjacency matrix, M, which has a "2" in entry (i, j) in case (i, j) is a directed edge of G, and "1" otherwise. M will be written as a string, $\bar{G}$, in $\{1, 2\}^{n^2}$, in the form

$$\bar{G} = \text{row } 1 \text{ row } 2 \cdots \text{row } n.$$

DEFINITION 18.

GAP $= \{\bar{G} \mid G$ is a directed graph on V_n which has a path from node 1 to node $n\}$,

DGAP $= \{\bar{G} \mid \bar{G}$ is in GAP, and no vertex of G has more than one directed edge leading from it$\}$.

(Note: GAP is essentially the set of "threadable mazes" as defined by Savitch [11].)

Next, we define a function which takes any Turing machine Z and input x as arguments, and yields the graph of all $S(\cdot)$-bounded ID's as value.

DEFINITION 19. Let Z be a Turing machine which operates in space $S(\cdot)$, with input alphabet Σ. The function $f_z\colon \Sigma^* \to \{1, 2\}^*$ is defined as follows. Consider the encoding of ID's and the constant c given by Lemma 17. Let $x \in \Sigma^*$ and let $n = 2^{(c+1)\cdot S(|x|)}$. Then $f_z(x) = \bar{G}$, where G is the directed graph whose vertices are $1, 2,..., n$, and whose edges are as follows.

1. If α is an encoded ID then (α, β) is a directed edge iff YIELD (x, α, β) is true;
2. If α is an encoded ID and FINAL (x, α) is true, then (α, n) is a directed edge;
3. $(1, \beta)$ is an edge iff INIT (x, β) is true;
4. G has no other edges.

Clearly any encoded ID α must satisfy $1 < \alpha < n$. Thus x is accepted by Z if and only if G has a path from vertex 1 to vertex n, i.e., iff $\bar{G} = f_Z(x) \in$ GAP. Further, $\bar{G}$ will be in DGAP iff Z is deterministic.

LEMMA 20. *For each fixed Turing machine Z, the function $f_Z(\cdot)$ is $S(\cdot)$-bounded rudimentary.*

Proof. Consider the following formula.

$$\begin{aligned}
R(x, i, a) \Leftrightarrow & (a = 1 \vee a = 2) \wedge \\
& (\exists m, n)_{(c+1)S}\{m = (c+1) \cdot S(|x|) \wedge n = 2^m \wedge \\
& \quad (\exists \alpha, \beta)_{cS}[1 \leqslant \alpha \leqslant n \wedge 1 \leqslant \beta \leqslant n \wedge i = n \cdot \alpha - n + \beta \wedge \\
& \qquad (a = 2 \Leftrightarrow [\mathrm{YIELD}(x, \alpha, \beta) \vee \\
& \qquad\qquad \mathrm{FINAL}(x, \alpha) \wedge \beta = n \vee \\
& \qquad\qquad \alpha = 1 \wedge \mathrm{INIT}(x, \beta)]) \quad]\}.
\end{aligned}$$

CLAIM. $R(x, i, a)$ is true iff a is the ith symbol of $\bar{G} = f_z(x)$.

Suppose the latter is true, and the ith symbol of $\bar{G}$ is element (α, β) of the corresponding adjacency matrix. Clearly $1 \leqslant \alpha \leqslant n$, $1 \leqslant \beta \leqslant n$, and $i = n \cdot (\alpha - 1) + \beta$. The condition "$a = 2 \Leftrightarrow \cdots$" above merely asserts that (α, β) is an edge of G, or not, according to the definition of G. Thus $R(x, i, a)$ will evaluate to "true." The converse is also straightforward.

Proof of Lemma 20 (*continued*). Now the right side above contains only predicates which are already known to be in RUD_S, and arithmetic predicates such as $u + v = w$, $u \cdot v = w$, $uv = w$ and $u \leqslant v$, which are known to be rudimentary by [1]. Each variable (except x) is bounded in length (of its dyadic representation) by a constant multiple of $S(|x|)$. Thus by Theorem 7 the right side defines a predicate in RUD_S. Thus $f(\cdot)$ is an $S(\cdot)$-bounded rudimentary function. ▌

The proof of the following theorem involves construction of a graph from a Turing machine (via function f_z), a technique essentially due to Savitch [11]. However [11] makes only an implicit use of reducibility concepts, and the type of reducibility implied there is Turing machine reducibility, which is more powerful than the $\leqslant_{\log}^{\mathrm{rud}}$ version.

THEOREM 21. GAP *is* $\leqslant_{\log}^{\mathrm{rud}}$*-complete in* $\mathrm{NSPACE}_{\log}$.

Proof. First, GAP is easily seen to be in $\mathrm{NSPACE}_{\log}$, by construction of a Turing machine Z_0 which operates as follows. Given $\bar{G}$ as input, Z_0 will:

1. Find n, the number of vertices of G;
2. Let $j = 1$;
3. If $j = n$ then halt, accepting $\bar{G}$;
4. If $j \neq n$, then let $i = j$ and
5. nondeterministically choose a new vertex j such that (i, j) is an edge of G;
6. Go to step 3.

If i, j and n are stored in dyadic notation, this requires at most $3 \log n \leqslant 2 \log n^2 = 2 \log | \bar{G} |$ work tape symbols.

We must now show that $L \leqslant_{\log}^{\text{rud}}$ GAP for any L in $\text{NSPACE}_{\log}$. Suppose L is accepted by nondeterministic Turing machine Z in $\log(\cdot)$ space. By the previous lemma the function $f_z(\cdot)$ is $\log(\cdot)$-bounded rudimentary, and for any $x \in \Sigma^*$, $x \in L$ iff $f_z(x) \in$ GAP.

Thus $L \leqslant_{\log}^{\text{rud}}$ GAP will be true iff $\log |f(x)| = d \cdot \log | x |$ for some d and all x. But this is immediate, since

$$| f_z(x) | = | \bar{G} | = n^2 = 2^{2(c+1) \cdot \log |x|} = (\log | x |)^{2(c+1)},$$

and so

$$\log | f_z(x) | = 2(c+1) \cdot \log | x |.$$

Consequently GAP is $\leqslant_{\log}^{\text{rud}}$-complete for $\text{NSPACE}_{\log}$. ▌

Corollary 22. *DGAP is $\leqslant_{\log}^{\text{rud}}$-complete in* $\text{DSPACE}_{\log}$.

Proof. To show that DGAP $\in \text{DSPACE}_{\log}$, construct Turing machine Z' to act as follows.

1. Determine whether some row of $\bar{G}$ has more than one "2" in it; if so, reject $\bar{G}$;

2. Apply machine Z_0 (from the previous theorem) to $\bar{G}$.

Step 1 can clearly be done deterministically in $\log(\cdot)$ space. Further, Z_0 can be made deterministic when applied only to graphs not rejected by step 1.

Now let L be accepted by deterministic Turing machine Z in $\log(\cdot)$ SPACE. Let $x \in \{1, 2\}^*$ be given, and consider the graph G such that $\bar{G} = f_z(x)$. Clearly G will have at most one edge leading from any vertex, due to the determinism of Z. Thus $\bar{G} \in$ DGAP if and only if Z accepts x; in other words $f_z(x) \in$ DGAP iff $x \in L$. Consequently $L \leqslant_{\log}^{\text{rud}}$ DGAP, so L is $\leqslant_{\log}^{\text{rud}}$-complete in $\text{DSPACE}_{\log}$. ▌

(Note: An equivalent formulation of the DGAP problem would be: *Given* a finite function $f: \{1, 2, ..., n\} \to \{1, 2, ..., n\}$, *to determine* whether there is a k such that $f^k(1) = n$.)

The following theorem strengthens a theorem of Savitch [11], to the effect that GAP $\in \text{DSPACE}_{\log}$ iff $\text{DSPACE}_S = \text{NSPACE}_S$ for all $S(\cdot) \geqslant \log(\cdot)$. His result is an immediate consequence of Lemma 3 and Corollary 22.

Theorem 23. *If* GAP(DGAP) $\in \text{RUD}_{\log}$ *and* $S(\cdot)$ *is any constructable space bound such that* $S(\cdot) \geqslant \log(\cdot)$, *then*

$$\text{NSPACE}_S(\text{DSPACE}_S) = \{L \mid L \subseteq \Sigma^* \textit{ and } L \in \text{RUD}_S\}.$$

Proof. Suppose GAP $\in$ RUD$_{\log}$, so that there is a log($\cdot$)-bounded rudimentary formula defining the predicate "$x \in$ GAP."

Let L be accepted by nondeterministic Turing machine Z in space $S(\cdot)$, and let $f_z(\cdot)$ be the function of Definition 19. We now apply Lemma 14, with $T(\cdot) = \log(\cdot)$, and $Q(x) \Leftrightarrow x \in$ GAP. There is a constant d such that $(|f_z(x)|) = \log 2^{dS(|x|)}$ for all $x \in \Sigma^*$, so

$$T(|f(x)|) = \log 2^{dS(|x|)} = d \cdot S(|x|).$$

Thus the conditions of Lemma 14 are satisfied, so the predicate $Q(f(x))$ is in RUD$_S$. But for any x

$$x \in L \Leftrightarrow f_z(x) \in \text{GAP} \Leftrightarrow Q(f(x)).$$

Thus $L \in$ RUD$_S$, so NSPACE$_S \subseteq \{L \mid L \subseteq \Sigma^* \text{ and } L \in \text{RUD}_S\}$. Reverse containment is immediate from Corollary 10, so equality holds. The same argument applies to DGAP, mutatis mutandis. ▌

The following result, due to William T. Laaser, shows that the sets of connected, and strongly connected, directed graphs are also complete for NSPACE$_{\log}$.

COROLLARY 24. *Each of the following is* $\leqslant^{\text{rud}}_{\log}$*-complete for* NSPACE$_{\log}$: CON $= \{\bar{G} \mid \forall m\,(1 \leqslant m \leqslant n \Rightarrow \exists\, \textit{path from } 1 \textit{ to } m)\}$, SCON $= \{\bar{G} \mid \forall m, p\,(1 \leqslant m \leqslant n \wedge 1 \leqslant p \leqslant n \Rightarrow \exists \textit{ path from } m \textit{ to } p)\}$.

Proof. Each set is easily seen to be in NSPACE$_{\log}$, by a construction similar to that of Theorem 21.

We now show that GAP $\leqslant^{\text{rud}}_{\log}$ CON, thus establishing completeness of CON. Define G' to contain the edges of G, plus all edges of the form (n, m) such that $1 \leqslant m \leqslant n$. If $\bar{G} \in$ GAP there is clearly a path in G' from node 1 to any node m, so $\bar{G}' \in$ CON. Conversely if $\bar{G}' \in$ CON there must be a shortest path from node 1 to node n. This path must also be a path in G, so $\bar{G} \in$ GAP. Thus $\bar{G} \in$ GAP iff $\bar{G}' \in$ CON. G' is clearly constructible from G by a log($\cdot$)-bounded rudimentary function, so GAP $\leqslant^{\text{rud}}_{\log}$ CON.

To show GAP $\leqslant^{\text{rud}}_{\log}$ SCON, let G'' contain all edges of G' plus those of the form $(m, 1)$ with $1 \leqslant m \leqslant n$. Now suppose $\bar{G} \in$ GAP. Then for any m, p G'' has a path of the form: $m \to 1 \to \cdots \to n \to p$, so $\bar{G} \in$ SCON. Conversely if $\bar{G}'' \in$ SCON, then G'' has a shortest path from node 1 to node n. This path may not contain any edge of the form (n, m) or $(m, 1)$, else it would not be shortest. Thus the path consists only of edges in G, so $\bar{G} \in$ GAP. Again $\bar{G} \in$ GAP iff $\bar{G}'' \in$ SCON, and it is easily seen that G'' is constructible from G by a log($\cdot$)-bounded rudimentary function. Thus SCON is also complete for NSPACE$_{\log}$. ▌

Open problem: Is UGAP $= \{\bar{G} \mid G$ is an *un*directed graph with a path from node 1 to node $n\}$ complete for $\text{NSPACE}_{\log}$?

We can also see that corresponding theorems hold for directed acyclic graphs, provided the node numbering is consistent with their natural partial order.

COROLLARY 25. *Define* DAG $= \{\bar{G} \mid \forall i,j$ *if* (i,j) *is an edge of* G, *then* $i < j\}$. *Then* GAP $\cap$ DAG *and* DGAP $\cap$ DAG *are* $\leqslant_{\log}^{\text{rud}}$*-complete for* $\text{NSPACE}_{\log}$ *and* $\text{DSPACE}_{\log}$, *respectively.*

Proof (Sketch). Membership in DAG is clearly decidable in deterministic $\log(\cdot)$ space, so GAP $\cap$ DAG $\in \text{NSPACE}_{\log}$ and DGAP $\cap$ DAG $\in \text{DSPACE}_{\log}$.

Now suppose L is accepted by Turing machine Z in $\log(\cdot)$ space, let $x \in \Sigma^*$, and let G be the graph built in Definition 19. Construct a new graph G' whose nodes are pairs (i, α) such that $1 \leqslant i \leqslant n$ and α is a node of G. The edges of G' are the form $((i, \alpha), (i+1, \beta))$ where (α, β) is an edge of G.

It is clear that G has a path from node 1 to n iff G' has a path from node $(1, 1)$ to (k, n) for some $k \leqslant n$. It is also clear that each pair (i, α) can be coded as a number $c_{i,\alpha}$ so that $i < j$ implies $c_{i,\alpha} < c_{j,\beta}$ for all α, β.

The remainder of the proof consists of adding initial and final nodes to G' so that $\bar{G} \in (D)$ GAP iff $\bar{G}' \in (D)$ GAP $\cap$ DAG, and showing that $\bar{G}'$ can be built from x by a $\log(\cdot)$-rudimentary function. Details are very similar to those of Lemma 20. ▮

(Note: Sudborough [13] independently obtained this result for $\leqslant_{\log}^{\text{tms}}$-reducibility.)

Note that this argument depends on the property that if (i,j) is an edge, then $i < j$. The situation is rather different if this condition is relaxed, as we now see.

Similar arguments show that $\{\bar{G} \mid G$ has at least one cycle$\}$ is also complete, with the natural deterministic analog. To see this, note that this set (call it CYCLE) is clearly in $\text{NSPACE}_{\log}$. Further, if G is any graph in DAG, then the result of attaching node n to node 1 will be cyclic iff $\bar{G} \in$ GAP $\cap$ DAG. Thus a complete problem reduces to CYCLE, so CYCLE is complete.

It is not known whether $\{\bar{G} \mid G$ is acyclic$\}$ is in $\text{NSPACE}_{\log}$. However if it is, then $\text{NSPACE}_{\log}$ must be closed under complementation, since $L \in \text{NSPACE}_{\log}$ implies $L \leqslant_{\log}^{\text{tms}}$ CYCLE, which implies $\Sigma^* - L \leqslant_{\log}^{\text{tms}} \{\bar{G} \mid G$ is acyclic$\}$, and this by assumption and Theorem 4 implies that $\Sigma^* - L$ must be in $\text{NSPACE}_{\log}$.

Finite Automaton Problems

THEOREM 26. *Each of the following problems is* $\leqslant_{\log}^{\text{rud}}$*-complete for* $\text{NSPACE}_{\log}$. *M denotes an arbitrary deterministic finite automaton, and $T(M)$ is the set accepted by M.*

(a) *Is $T(M)$ nonempty?*

(b) *Is $T(M)$ an infinite set?*

Further, each of these problems is $\leqslant_{\log}^{\text{rud}}$*-complete for* $\text{DSPACE}_{\log}$, *in case* M *is restricted to one input symbol.*

Proof (Sketch). $T(M) \neq \varnothing$ may be determined by merely "guessing" the symbols of an input string x, one symbol at a time, and recording the current state on the work tape; M is accepted iff a final state is ever reached. Further, $T(M)$ is infinite iff an input x is accepted with $|x| \geqslant k$, the number of states of M.

Consequently each of these problems is in $\text{NSPACE}_{\log}$.

To reduce GAP to these problems, let G be a directed graph with vertices $1, 2,\ldots, n$; define $M_G = (\{0, 1,\ldots, n\}, \{0, 1,\ldots, n\}, \delta, 1, \{n\})$, where

(a) $\delta(i,j) = j$ if (i,j) is an edge of G (in $\delta(i,j)$, i is the state, and j, the output);

(b) $\delta(n, 0) = 1$; and

(c) $\delta(i,j) = 0$ for all other values of i, j.

It is immediate that $\bar{G} \in \text{GAP}$ iff $T(M) \neq \varnothing$ and $\bar{G} \in \text{GAP}$ iff $T(M)$ is infinite. Proof that this is a $\leqslant_{\log}^{\text{rud}}$ reduction is straightforward.

For the one-letter case, the procedures given above are deterministic, so each problem is in $\text{DSPACE}_{\log}$. However, machine M_G has more than one input symbol, so a new construction is necessary. Suppose $\bar{G} \in \text{DGAP}$.

Let M_G' equal $(\{0, 1,\ldots, n\}, \{a\}, \delta, 1, \{n\})$, where

(a) $\delta(i, a) = j$ if (i,j) is an edge of G and $i \neq n$,

(b) $\delta(n, a) = 1$, and

(c) $\delta(i, a) = 0$ for other values of i.

M_G' is clearly deterministic, since $\bar{G} \in \text{DGAP}$. Again $T(M)$ is nonempty and infinite iff $\bar{G} \in \text{DGAP}$. ∎

5. Conclusions

A very strict notion of reducibility among combinatorial problems has been defined, making it possible to investigate equivalence among problems at a very low level of complexity, such as graph accessibility, finiteness of regular sets, and Turing machine computations involving at most $\log n$ space. If any one of these problems can be shown to require certain minimal computational resources, then all of them do. Because of their simplicity, it is hoped that insights into their intrinsic complexity may be acquired which are more illuminating than the traditional diagonalization and growth rate arguments. Results in this paper indicate that insights at this level also have interesting consequences at higher levels of complexity.

Acknowledgments

The author thanks Charles Pfleeger, Ravi Sethi, and Charles Hughes for helpful discussions while this work was in progress.

References

1. J. Bennett, On spectra, Doctoral Dissertation, Princeton University, 1962.
2. S. A. Cook, The complexity of theorem-proving procedures, "Conf. Rec. Third ACM Symp. on Theory of Computing," pp. 161–158, Association For Computing Machinery, New York, 1971.
3. H. B. Hunt III, On the time and tape complexity of languages I, "Fifth ACM Symp. on Theory of Computing," Association For Computing Machinery, New York, 1973.
4. J. Hopcroft and J. Ullman, "Formal Languages and Their Relation to Automata," Addison–Wesley, Reading, Mass., 1969.
5. N. D. Jones, "Computability Theory: An Introduction," ACM Monographs, Academic Press, New York, 1973.
6. R. M. Karp, Reducibility among combinatorial problems, *in* "Complexity of Computer Computations" (R. E. Miller and J. W. Thatcher, Eds.), Plenum Press, New York, 1972.
7. J. Myhill, Linear bounded automata, Wright Air Dev. Command Tech. Rept. 57-624, pp. 112–237 (1957).
8. A. R. Meyer and L. J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, "Proc. 13th Symp. on Switching and Automata Theory," pp. 125–129, 1972.
9. C. Pfleeger, State reduction in incompletely specified finite machines, *IEEE Trans. Electronic Computers* **EC23** (1973), 1099–1103.
10. R. Smullyan, "Theory of Formal Systems," Annals of Math. Studies No. 47, Princeton University Press, Princeton, N.J., 1961.
11. W. Savitch, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4** (1970), 177–192.
12. L. J. Stockmeyer and A. R. Meyer, Word problems requiring exponential time: Preliminary report, "Fifth ACM Symp. on Theory of Computing," **EC23** (1973), 1099–1103.
13. I. H. Sudborough, On tape-bounded complexity classes and multihead finite automata, *J. Comput. System Sci.*, to appear.