SIAM J. COMPUT.
Vol. 7, No. 2, May 1978

ON STRUCTURE PRESERVING REDUCTIONS*

NANCY LYNCH† AND RICHARD J. LIPTON‡

Abstract. The concept of reduction between problems is strengthened. Certain standard problems are shown to be complete in the new and stronger sense. Applications to the number of solutions of particular problems are presented.

Key words. reductions, polynomial time, logspace, complete sets

1. Introduction. One of the most striking features of a large number of the known reductions of one problem to another [3] is that they often preserve a great deal more than they have to. More precisely, suppose that A is many-to-one polynomial time reducible to B where A and B are, as usual, subsets of Σ^* for some finite alphabet Σ. Then all that is required in the usual definition is that

$$(*) \qquad \forall x \in \Sigma^*, \quad x \in A \text{ if and only if } f(x) \in B,$$

where $f(x)$ is some polynomial time computable function. Essentially (*) states that x has a solution exactly when $f(x)$ has a solution. It appears, however, that quite often x and $f(x)$ are more closely related than this.

This imprecise intuitive feeling that reductions often preserve additional structure is the subject of this paper. We introduce a new kind of reduction and prove that some standard complete problems are also complete in our strong sense.

The notion that reduction preserves additional structure also appears in Simon [7]. His main result is that a number of problems are still equivalent when (*) is strengthened to:

$$x \text{ has the same number of solutions as } f(x).$$

He calls reducibilities preserving the number of solutions "parsimonious". There is a difficulty with these results, however; it is not clear what it means for x to have k solutions when A is an arbitrary set. Clearly, either x is in A or it is not. Simon avoids this difficulty by working only with well known and specific problems. In these cases it is reasonable to assume that "x has k solutions" is a meaningful concept. We take an alternative approach. The main virtue of this approach is that it allows us to work with arbitrary problems, and thus we can prove the existence of complete sets.

2. New definition of reduction. The key idea of the new reduction is a focus on relations rather than on sets. Roughly, suppose that

$$\forall x \in \Sigma^*, \quad x \in A \text{ if and only if } \exists y R(x, y).$$

The intuitive concept that "x is an instance of A with k solutions" can be more precisely rendered by "there are k y's such that $R(x, y)$ is true." There are, however, several interesting difficulties in making this rough idea work correctly. In this direction the next definition is the key.

* Received by the editors February 28, 1977, and in revised form July 25, 1977. This work was supported in part by the National Science Foundation under Grant DCR 92373. Part of this work was carried out while the authors were visiting IBM Thomas J. Watson Research Laboratory in June, 1975.

† School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia 30332.

‡ Department of Computer Science, Yale University, New Haven, Connecticut 06520. The work of this author was supported in part by the U.S. Office of Naval Research under Grant N00014-75-C-0752.

DEFINITION. A *combination machine* is a Turing machine with two read-only input tapes with end-markers, the first 2-way and the second 1-way, a 2-way read-write worktape and a 1-way write-only output tape. A combination machine is *logspace* (*polynomial time*) if it always halts and runs within worktape space logarithmic (within time polynomial) in the length of the first input.

As remarked in [5], a set A is in $\mathcal{NL}$, the class of nondeterministic logspace sets ($\mathcal{NP}$, the class of nondeterministic polynomial time sets) if and only if there exist a polynomial p and a relation R such that

$$x \in A \Leftrightarrow (\exists y)[|y| \leqq p(|x|) \wedge R(x, y)],$$

and R is computable by a logspace (polynomial time) combination machine.

Now let R and S be arbitrary binary relations, and let r and s be polynomials. Then we will define reducibility $\leqq^{\mathcal{L}}$ ($\leqq^{\mathcal{P}}$) between (R, r) and (S, s) as follows:

$(R, r) \leqq^{\mathcal{L}} (\leqq^{\mathcal{P}}) (S, s)$ provided there exists an f and g such that

1. f is a function computable by a deterministic logspace (polynomial time) transducer 2-way on its input;
2. g is a function computable by a logspace (polynomial time) combination machine;
3. $\forall x, y \in \Sigma^*$,

$$[R(x, y) \wedge |y| \leqq r(|x|)] \quad \text{implies} \quad [S(f(x), g(x, y)) \wedge |g(x, y)| \leqq s(|f(x)|)];$$

4. g is 1–1 in the sense that $\forall x, y_1, y_2 \in \Sigma^*$,

$$[R(x, y_1) \wedge R(x, y_2) \wedge |y_1| \leqq r(|x|) \wedge |y_2| \leqq r(|x|) \wedge g(x, y_1) = g(x, y_2)] \quad \text{implies} \quad y_1 = y_2;$$

5. g is onto in the sense that $\forall x, z \in \Sigma^*$,

$$[S(f(x), z) \wedge |z| \leqq s(|f(x)|)] \quad \text{implies} \quad [\exists y |y| \leqq r(|x|) \wedge R(x, y) \wedge g(x, y) = z].$$

This definition, while at first appearing to be complex, is actually a natural extension of the usual one. In order to see this, observe that the usual definition states that A is logspace (polynomial time) reducible to B for A and B which are expressed by

$$A = \{x | \exists y |y| \leqq r(|x|) \wedge R(x, y)\}$$

and

$$B = \{x | \exists y |y| \leqq s(|x|) \wedge S(x, y)\}$$

provided

$$[\exists y |y| \leqq r(|x|) \wedge R(x, y)] \Leftrightarrow [\exists y |y| \leqq s(|f(x)|) \wedge S(f(x), y)],$$

where f is some logspace (polynomial time) transduction. Our main idea is to strengthen this condition by putting into 1–1 correspondence specific witnesses to the two existential quantifiers. Note that according to our definitions, if $(R, r) \leqq^{\mathcal{L}} (\leqq^{\mathcal{P}}) (S, s)$ via f and g, then for any x,

$$|\{y | |y| \leqq r(|x|) \wedge R(x, y)\}| = |\{y | |y| \leqq s(|f(x)|) \wedge S(f(x), y)\}|.$$

PROPOSITION 1. $\leqq^{\mathcal{L}}$ ($\leqq^{\mathcal{P}}$) *is transitive.*

Proof. We consider $\leqq^{\mathscr{L}}$. We need only show that if $(R, r) \leqq^{\mathscr{L}} (S, s)$ via f, g and $(S, s) \leqq^{\mathscr{L}} (T, t)$ via f', g' then $(R, r) \leqq^{\mathscr{L}} (T, t)$ via

$$f'' = \lambda x[f'(f(x))] \quad \text{and} \quad g'' = \lambda x, y[g'(f(x), g(x, y))].$$

First, f'' is a logspace transduction; this follows by standard arguments [8]. We show that g'' is computable by a logspace combination machine as follows:

Simulate g'. The only difficulty is in obtaining the appropriate bits of the inputs to g' as needed. The first input is easy: in order to compute the ith bit of $f(x)$ we need only simulate $f(x)$ from the beginning until it outputs the ith bit. This works since x is 2-way; a counter must be maintained for i. The second input is more difficult. In order to compute the ith bit of $g(x, y)$ we simply simulate $g(x, y)$ until it outputs the ith bit. The key is that g' asks for these bits in the same order as produced in the computation of $g(x, y)$; thus, in the simulation of $g(x, y)$ it suffices to have y on a 1-way tape. No counter need be maintained for i in this case.

Now verification of properties 3–5 is reasonably straightforward. Transitivity of $\leqq^{\mathscr{P}}$ is obvious.

DEFINITION. (S, s) is *$\mathscr{L}$-complete* (*$\mathscr{P}$-complete*) if S is a relation computable by a logspace (polynomial time) combination machine, s is a polynomial, and for all (R, r), where R is computable by a logspace (polynomial time) combination machine and r is a polynomial,

$$(R, r) \leqq^{\mathscr{L}} (\leqq^{\mathscr{P}}) (S, s).$$

It is not difficult to show that if (S, s) is $\mathscr{L}$-complete ($\mathscr{P}$-complete), then

$\{x \mid \exists y |y| \leqq s(|x|) \wedge S(x, y)\}$ is complete in $\mathscr{NL}$ ($\mathscr{NP}$) according to the more usual definitions.

Let SAT(x, y) be "x is a conjunctive normal form Boolean formula and y is an assignment of true or false to the variables of x making x true" [3]. SAT is clearly computable by a polynomial time combination machine. Let $s(n) = n$.

PROPOSITION 2. *(SAT, s) is $\mathscr{P}$-complete.*

Proof. Let R be a relation computable by a polynomial time combination machine M and let r be a polynomial. We will construct a nondeterministic Turing machine M' from M and r as follows:

M' on input x simulates M on inputs of the form (x, y) by guessing bits of y (including guessing when the end of y has been reached) when M needs them. M' also keeps a counter and if M tries to read more than $r(|x|)$ bits of y, then M' will reject the input x for this series of guesses. If M rejects (x, y), then M' will also reject x on the corresponding computation path. If M accepts then M' will continue to guess bits of y, and it accepts when it guesses that the end has been reached; again if it tries to exceed $r(|x|)$ total bits of y then it rejects.

We also require that every guess made by M' be actually "written down" when made (at least temporarily) so that distinct values of y satisfying $R(x, y)$ and $|y| \leqq r(|x|)$ will cause M' to follow distinct computation paths. Clearly there is a polynomial q such that M' on any input x and any computation path, halts in at most $q(|x|)$ steps; by standard techniques, we can actually assume that any computation of M' that accepts x halts in exactly $q(|x|)$ steps.

Now M' is coded into (SAT, s) as in Simon [7]. In order to show $(R, r) \leqq$ (SAT, s) we obtain the required mappings as follows:

1. $f(x)$ is the Boolean formula obtained by coding the computations of M' on the input x;

2. $g(x, y)$ is anything we like if $|y| > r(|x|)$; otherwise, let M' operate on input x and guess precisely the input y when simulating the machine M; then let $g(x, y)$ be the Boolean assignment to the variables of the formula $f(x)$ which describes this computation.

We may now assert that these functions have the required properties. Properties 1 and 2 of the definition of $\leqq^{\mathscr{P}}$ are clear. For 3, we must show that

$$[R(x, y) \wedge |y| \leqq r(|x|)] \quad \text{implies} \quad [S(f(x), g(x, y)) \wedge |g(x, y)| \leqq s(|f(x)|)].$$

But this should be clear from the construction of M', $f(x)$ and $g(x, y)$. To see 4, suppose that $R(x, y_1)$, $R(x, y_2)$, $|y_1| \leqq r(|x|)$, $|y_2| \leqq r(|x|)$, and $g(x, y_1) = g(x, y_2)$ are all true. Since M' writes down all its guesses, it must be the case that $y_1 = y_2$.

Finally we will show property 5. Suppose that $S(f(x), z)$ and $|z| \leqq s(|f(x)|)$ are true. Since z encodes the guesses of M', there must be an input y (since M' only guesses the second input) such that $R(x, y)$ with $|y| \leqq r(|x|)$ and by construction $g(x, y) = z$. Thus g is onto, and hence (SAT, s) is $\mathscr{P}$-complete. □

We could, of course, extend Proposition 2 to a collection of other polynomial-computable relations. Rather than pursuing such results, we turn our attention to relations computable by logspace combination machines. Let GAP (x, y) be "x encodes an acyclic directed graph (Savitch [6]) with the property that the total order of nodes induced by the node numbering is a topological ordering of the partial ordering induced by the edge directions, and y encodes a directed path from start to finish." Again let $s(n) = n$. Clearly, GAP is computable by a logspace combination machine.

PROPOSITION 3. (GAP, s) *is* $\mathscr{L}$*-complete.*

Proof. Since this is almost identical to the proof of Proposition 2 we will only sketch it. Let R be a relation computable by a logspace combination machine M, and let r be a polynomial. Define M' as follows:

M' on input x simulates M on inputs of the form (x, y) by guessing bits of y when M needs them. (Note since M is 1-way on this input M' does *not* have to remember all of y). M' then operates just as in Proposition 2.

Since guesses need only be written down temporarily, there is no difficulty with the space bound. Clearly M' will be a nondeterministic logspace Turing machine which we can assume halts for any computation in exactly $q(|x|)$ steps, for some polynomial q.

M' is encoded into GAP as in [6]. Then f and g are obtained as follows:

1) $f(x)$ is the graph obtained by encoding of M' on x.
2) $g(x, y)$ is anything we like if $|y| > r(|x|)$. Otherwise, let M' on input x with guesses y yield the path $g(x, y)$ through the graph $f(x)$.

Then $(R, r) \leqq^{\mathscr{L}}$ (SAT, s) via this f and g. A key again is that the computation of M' encodes the actual y it guesses so that $g(x, y)$ will be 1–1. □

We note that Proposition 3 is also extendible to a variety of other logspace-computable relations.

3. Size of solution sets for relations. We consider $\mathscr{L}$-complete ($\mathscr{P}$-complete) (R, r). We wish to give a complexity classification for

$$(R, r)_k = \{x \mid \exists \text{ exactly } k \text{ values of } y,\ |y| \leqq r(|x|) \wedge R(x, y)\}$$

for various values of k. We will first examine specific problems and then we will use the results of §2 to obtain generalizations. In the following, we let $\leqq^{\mathscr{P}}_{\mathscr{M}}$, $\leqq^{\mathscr{L}}_{\mathscr{M}}$, $\leqq^{\mathscr{P}}_{\mathscr{T}}$ and $\leqq^{\mathscr{L}}_{\mathscr{T}}$ represent the reducibilities used by Karp [3], Jones and Laaser [2], Cook [1] and Ladner and Lynch [4], respectively. For convenience, we let SAT1 (x, y) be "x is an arbitrary form Boolean formula and y is an assignment of true or false to the variables of x making x true." Let $s(n) = n$ as before. Then it is clear that (SAT1, s) is $\mathscr{P}$-complete.

PROPOSITION 4. *For any fixed* $k \geqq 1$,

(a) $$(\mathrm{GAP}, s)_k \underset{\mathcal{M}}{\equiv}^{\mathcal{L}} (\mathrm{GAP}, s)_1,$$

and

(b) $$(\mathrm{SAT1}, s)_k \underset{\mathcal{M}}{\equiv}^{\mathcal{P}} (\mathrm{SAT1}, s)_1.$$

Proof. (a) We first show

$$(\mathrm{GAP}, s)_k \underset{\mathcal{M}}{\leqq}^{\mathcal{L}} (\mathrm{GAP}, s)_1.$$

Assume $k \geqq 2$. If x is not an acyclic directed graph satisfying the given consistency condition, then let $f(x) = x$. Otherwise, let $\not{p}: N^k \times \{0, 1\}^{k-1}$ (where N is the set of natural numbers) be a natural 1–1 $(2k-1)$-tupling function with the property that

$$\left[\bigwedge_{i=1}^{k} (x_i \leqq y_i) \wedge \bigvee_{i=1}^{k} (x_i < y_i)\right]$$
$$\text{implies} \quad \not{p}(x_1, \cdots, x_k, a_1, \cdots, a_{k-1}) < \not{p}(y_1, \cdots, y_k, b_1, \cdots, b_{k-1}).$$

The *start node* of $f(x)$ is

$$\not{p}\ (\underbrace{n_s, \cdots, n_s}_{k}, \underbrace{0, \cdots, 0}_{k-1}),$$

where n_s is the (number of the) start node of x. The *goal node* of $f(x)$ is

$$\not{p}\ (\underbrace{n_G, \cdots, n_G}_{k}, \underbrace{1, \cdots, 1}_{k-1}),$$

where n_G is the (number of the) goal node of x. The edges of graph $f(x)$ will be defined by tupling together edges of x as follows:

$$(\not{p}(x_1, \cdots, x_k, a_1, \cdots, a_{k-1}), \not{p}(y_1, \cdots, y_k, b_1, \cdots, b_{k-1}))$$

will be an edge of $f(x)$ exactly if:

1) $(x_1, \cdots, x_k, a_1, \cdots, a_{k-1}) \neq (n_G, \cdots, n_G, 1, \cdots, 1)$, and
2) for all i, $1 \leqq i \leqq k-1$,
   (a) either (x_i, y_i) is an edge of x, or $x_i = y_i = n_G$, and
   (b) one of (b1)–(b3) holds:
      (b1) $a_i = 0$ and $x_i = x_{i+1} \neq n_G$ and $b_i = 0$ and $y_i = y_{i+1} \neq n_G$,
      (b2) $a_i = 0$ and $x_i = x_{i+1} \neq n_G$ and $b_i = 1$ and $y_i < y_{i+1}$,
      (b3) $a_i = 1$ and $b_i = 1$.

The reader may verify that $f(x)$ is an acyclic directed graph satisfying the given consistency condition, and that $f(x)$ is computable from x in logspace. Intuitively, a single path through $f(x)$ corresponds to parallel simulation of k distinct paths through x, where a flag is changed from 0 to 1 to indicate that two "adjacent" paths have just been discovered to diverge with the path at the left preceding the path at the right lexicographically. Padding is used for shorter paths in x. With this intuition, the reader should be able to verify that x has exactly k solutions iff $f(x)$ has exactly 1 solution.

We now must show $(\mathrm{GAP}, s)_1 \underset{\mathcal{M}}{\leqq}^{\mathcal{L}} (\mathrm{GAP}, s)_k$. But this is straightforward by a construction which adds $k-1$ disjoint "dummy paths" to a graph of the appropriate type. □

(b) We first show

$$(\mathrm{SAT1}, s)_k \underset{\mathcal{M}}{\leqq}^{\mathcal{P}} (\mathrm{SAT1}, s)_1.$$

If x is not a Boolean formula, define $f(x)=x$. Otherwise, assume x is of the form $a(x_1, \cdots, x_n)$. Let $f(x)$ be the formula obtained by selecting new disjoint sets of variables $\{x_{i1}, \cdots, x_{in}\}_{i=1}^{k}$ and expanding into the appropriate form the expression:

$$[a(x_{11}, \cdots, x_{1n}) \wedge a(x_{21}, \cdots, x_{2n}) \wedge \cdots \wedge a(x_{k1}, \cdots, x_{kn}) \wedge (x_{11}x_{12}\cdots x_{1n} < x_{21}x_{22}\cdots x_{2n} < \cdots < x_{k1}x_{k2}\cdots x_{kn})].$$

The last line of inequalities is intended to indicate lexicographic ordering of the given strings. f is computable in polynomial time, and x has exactly k solutions iff $f(x)$ has exactly 1 solution.

Next we show

$$(\mathrm{SAT1}, s)_1 \underset{\mathscr{M}}{\leqq}^{\mathscr{P}} (\mathrm{SAT1}, s)_k.$$

If $k = 1$ there is nothing to prove, so assume $k \geqq 2$. If x is not a Boolean formula, define $f(x)=x$. Otherwise, assume x is of the form $a(x_1, \cdots, x_n)$. Let $f(x)$ be the formula

$$\left[a(x_1, \cdots, x_n) \wedge \bigwedge_{i=1}^{k-1} \overline{x_{n+1}}\right] \vee \bigvee_{i=1}^{k-1} \left[\left(\bigwedge_{j=1}^{n+i} x_j\right) \wedge \left(\bigwedge_{j=n+i+1}^{n+k-1} \overline{x_j}\right)\right].$$

f essentially adds $k-1$ dummy solutions to x, and so x has exactly 1 solution iff $f(x)$ has exactly k solutions. □

We now note that a result similar to Proposition 4 must hold for *all* complete (R, r). That is, addition of dummy solutions and collapsing of several solutions to one are constructions which work for all complete problems. In fact, all such problems must be equivalent to each other:

PROPOSITION 5. *For any fixed* $k \geqq 1$,

(a) *if* (R, r) *is* $\mathscr{L}$*-complete, then*

$$(R, r)_k \underset{\mathscr{M}}{\equiv}^{\mathscr{L}} (\mathrm{GAP}, s)_1,$$

and

(b) *if* (R, r) *is* $\mathscr{P}$*-complete, then*

$$(R, r)_k \underset{\mathscr{M}}{\equiv}^{\mathscr{P}} (\mathrm{SAT1}, s)_1.$$

Proof. By Proposition 4 and the fact that our reducibilities $\leqq^{\mathscr{L}}$ and $\leqq^{\mathscr{P}}$ preserve cardinality of solution sets (as noted immediately prior to Proposition 1). □

Now that we know that all size problems lie in a common complexity class, we would like to be able to say more about the location of this class. The only such information we have so far arises as a result of the following proposition. Here, let $G(S)$ be $\mathscr{NL}$-complete ($\mathscr{NP}$-complete) in the usual sense.

PROPOSITION 6. (a) *If*

$$\bar{G} \underset{\mathscr{M}}{\leqq}^{\mathscr{L}} A \underset{\mathscr{T}}{\leqq}^{\mathscr{L}} G,$$

then $A \in \mathscr{NL}$ *iff* $\mathscr{NL}$ *is closed under complement, and* $A \in \mathscr{L}$ *iff* $\mathscr{L} = \mathscr{NL}$, *and*

(b) *If*

$$\bar{S} \underset{\mathscr{M}}{\leqq}^{\mathscr{P}} A \underset{\mathscr{T}}{\leqq}^{\mathscr{P}} S,$$

then $A \in \mathscr{NP}$ *iff* $\mathscr{NP}$ *is closed under complement, and* $A \in \mathscr{P}$ *iff* $\mathscr{P} = \mathscr{NP}$.

Proof. The arguments are all standard Turing machine constructions, of the type found in [2] or [4], for example. □

Finally, we can conclude:

PROPOSITION 7. *For any fixed* $k \geqq 1$,

(a) *if* (R, r) *is* $\mathcal{L}$*-complete, then* $(R, r)_k \in \mathcal{NL}$ *iff* $\mathcal{NL}$ *is closed under complement, and* $(R, r)_k \in \mathcal{L}$ *iff* $\mathcal{L} = \mathcal{NL}$, *and*

(b) *if* (R, r) *is* $\mathcal{P}$*-complete, then* $(R, r)_k \in \mathcal{NP}$ *iff* $\mathcal{NP}$ *is closed under complement, and* $(R, r)_k \in \mathcal{P}$ *iff* $\mathcal{P} = \mathcal{NP}$.

Proof. (a) It suffices to show

$$\bar{G} \underset{\mathcal{M}}{\leqq}^{\mathcal{L}} (\mathrm{GAP}, s)_1 \underset{\mathcal{T}}{\leqq}^{\mathcal{L}} G,$$

where $G = \{x | \exists y |y| \leqq s(|x|)$ and GAP $(x, y)\}$.

The first reduction follows by adding a single "dummy path" to a graph.

To see the second reduction, note that $(\mathrm{GAP}, s)_k = A \cap \bar{B}$, where

$$A = \{x | \exists \text{ at least } k \text{ values of } y, |y| \leqq s(|x|) \wedge \mathrm{GAP}\ (x, y)\},$$

$$B = \{x | \exists \text{ at least } k+1 \text{ values of } y, |y| \leqq s(|x|) \wedge \mathrm{GAP}\ (x, y)\}.$$

Clearly A, B are both in $\mathcal{NL}$, so that

$$A \underset{\mathcal{M}}{\leqq}^{\mathcal{L}} G \quad \text{and} \quad B \underset{\mathcal{M}}{\leqq}^{\mathcal{L}} G.$$

Then a Turing machine with a G-oracle may easily be constructed to decide membership in $(\mathrm{GAP}, s)_1$.

(b) It suffices to show

$$\bar{S} \underset{\mathcal{M}}{\leqq}^{\mathcal{P}} (\mathrm{SAT1}, s)_1 \underset{\mathcal{T}}{\leqq}^{\mathcal{P}} S.$$

To see the first reduction, we use a special case of construction for Proposition 4(b): if x is not a Boolean formula, define $f(x) = x$. Otherwise, if x is the form $a(x_1, \cdots, x_n)$ let $f(x)$ be the formula $[a(x_1, \cdots, x_n) \wedge \overline{x_{n+1}}] \vee [x_1 \wedge \cdots \wedge x_n \wedge x_{n+1}]$. x has no solutions iff $f(x)$ has exactly one solution.

The second reduction follows from the same argument as in (a). □

Of course, the given complexity classification is still very incomplete; further work remains to be done. For example, is $(\mathrm{SAT}, s)_1$ in $\mathcal{NP}$?

We would expect that there are other interesting properties of solution sets which are preserved by strong reducibilities such as ours. Finding the appropriate strength reducibilities needed to preserve constructions such as those used for approximation, or for finding a *particular* solution when existence is known, seems to be an interesting area for further study.

REFERENCES

[1] S. A. COOK, *The complexity of theorem-proving procedures*, Proc. 3rd ACM Symposium in Theory of Computing, 1971, pp. 151–158.

[2] N. JONES AND W. LAASER, *Complete problems for deterministic polynomial time*, Proceedings of Sixth Annual ACM Symposium on Theory of Computing, April–May 1974, pp. 40–46.

[3] R. M. KARP, *Reducibility among combinatorial problems*, Complexity of Computer Computations, R. Miller and J. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–104.

[4] R. LADNER AND N. A. LYNCH, *Relativization of questions about log space computability*, Math. Systems Theory, 10 (1976), pp. 19–32.

[5] R. LIPTON AND N. A. LYNCH, *A quantifier characterization for nondeterministic log space*, SIGACT News, 7 (1975), no. 4, pp. 24–26.
[6] W. SAVITCH, *Relations between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 14 (1970), pp. 177–192.
[7] J. SIMON, *On some central problems in computational complexity*, TR 75-224, Cornell University, Ithaca, NY, 1975.
[8] L. STOCKMEYER AND A. R. MEYER, *Word problems requiring exponential time: Preliminary report*, 5th Annual ACM Symposium on Theory of Computing, 1973, pp. 1–9.