THE JOURNAL OF SYMBOLIC LOGIC
Volume 52, Number 1, March 1987

SURVEY/EXPOSITORY PAPER

CLASSIFYING THE COMPUTATIONAL COMPLEXITY OF PROBLEMS

LARRY STOCKMEYER

§1. Introduction. One of the more significant achievements of twentieth century mathematics, especially from the viewpoints of logic and computer science, was the work of Church, Gödel and Turing in the 1930's which provided a precise and robust definition of what it means for a problem to be computationally solvable, or decidable, and which showed that there are undecidable problems which arise naturally in logic and computer science. Indeed, when one is faced with a new computational problem, one of the first questions to be answered is whether the problem is decidable or undecidable. A problem is usually defined to be decidable if and only if it can be solved by some Turing machine, and the class of decidable problems defined in this way remains unchanged if "Turing machine" is replaced by any of a variety of other formal models of computation. The division of all problems into two classes, decidable or undecidable, is very coarse, and refinements have been made on both sides of the boundary. On the undecidable side, work in recursive function theory, using tools such as effective reducibility, has exposed much additional structure such as degrees of unsolvability. The main purpose of this survey article is to describe a branch of computational complexity theory which attempts to expose more structure within the decidable side of the boundary.

Motivated in part by practical considerations, the additional structure is obtained by placing upper bounds on the amounts of computational resources which are needed to solve the problem. Two common measures of the computational resources used by an algorithm are *time*, the number of steps executed by the algorithm, and *space*, the amount of memory used by the algorithm. Given a specific decidable problem, one typically obtains an upper bound on the required time and space by exhibiting a particular algorithm which solves the problem and bounding its resources. However, in judging the optimality of algorithms it is necessary also to have corresponding lower bounds on the computational complexity of the problem; that is, one must show that a certain amount of time or space is used by any of the (in general, infinitely many) algorithms which solve the problem. One method, based on diagonalization, has been developed for proving lower bounds, and many applications of this method have been made to problems in areas including logic,

Received September 13, 1985; revised April 18, 1986.

combinatorial games, and algebra. However, for many interesting and important problems this method does not appear to be applicable, and proofs of nontrivial lower bounds are not known. Even if close explicit upper and lower bounds on the complexity of a problem are not known, it may still be possible to implicitly classify the problem by relating its complexity to that of any problem in some large class of problems. Both types of classification, explicit and implicit, are surveyed here. Before getting into technical details in later sections, the basic ideas are outlined informally in this Introduction.

We first consider an explicit classification. Historically, the first explicit exponential lower bounds on the complexities of natural decision problems were proved by Meyer and Stockmeyer [MeS] for a certain problem in formal language theory and by Meyer [Me1] for the weak monadic second-order theory of one successor; the method introduced in these papers was used in subsequent papers to prove lower bounds. By "natural problem" we mean a problem with a reasonable practical or mathematical motivation, not a problem which is constructed specifically to be complex. As our first example we choose another logical decision problem, the first-order theory of the real numbers with addition, since this problem is probably more familiar to many readers than the problems considered in [MeS] and [Me1]. Consider formulas written in first-order predicate calculus with equality using the binary function symbol $+$ and the binary relation symbol $<$, together with the usual logical connectives such as *and*, *or* and *implies*, quantifiers $\exists$ and $\forall$, variables, and parentheses. A *sentence* is a formula in which every variable is bound by a quantifier. When variables are interpreted as ranging over the real numbers, $+$ is interpreted as addition of real numbers, and $<$ is interpreted as the order relation on real numbers, every sentence is either true or false. Denote the set of true sentences by $\mathrm{Th}(\mathbf{R}, +)$. Decidability of $\mathrm{Th}(\mathbf{R}, +)$ follows from the result of Tarski [Tars] that $\mathrm{Th}(\mathbf{R}, +, \cdot)$, the first-order theory of the reals with both addition and multiplication, is decidable. We say that a Turing machine *accepts* $\mathrm{Th}(\mathbf{R}, +)$ if, when started with (an encoding of) a sentence s on its tape, the machine always halts and it halts in a distinguished accepting state iff s is true. The time and space used by a Turing machine depend, in general, on the particular input. One simplification which is commonly made is to measure the consumption of these resources as a function of the *length* of the input. Let us take the length of a sentence s, denoted $|s|$, to be the number of occurrences of variable symbols and other symbols, $\exists$, $+$, $)$, etc., in s. The following theorem, due to Fischer and Rabin [FiR], is an example of an explicit lower bound on the computational complexity of a problem.

THEOREM 1.1. *There is a rational constant $c > 1$, such that if M is a Turing machine which accepts* $\mathrm{Th}(\mathbf{R}, +)$, *then M runs for at least $c^{|s|}$ steps when started on input s, for infinitely many sentences s.*

Of course, the same lower bound holds for the decision problem $\mathrm{Th}(\mathbf{R}, +, \cdot)$ originally considered by Tarski. The fact that any Turing machine requires exponentially growing time implies the same (with possibly a different constant c) for more realistic models of computers such as random-access register machines (see [CoR] and [AHU]); this is true because there are sufficiently efficient simulations of random-access machines by Turing machines.

Regarding upper bounds, Ferrante and Rackoff [FeR1] have shown that there is a constant d and a Turing machine M which accepts $\mathrm{Th}(\mathbf{R}, +)$ and which uses space (i.e., amount of tape) at most $d^{|s|}$ when deciding the truth of sentence s. The time used by this procedure grows double-exponentially in $|s|$. Ben-Or, Kozen and Reif [BeKR] establish the upper bound $d^{|s|^2}$ on the space complexity of Th($\mathbf{R}, +, \cdot$), where d is a constant. (Collins [Col] and Monk [Mo] had earlier given decision procedures for Th($\mathbf{R}, +, \cdot$) where the time and space are both doubly exponential in $|s|$. The time and space used by Tarski's original decision procedure cannot be bounded above by the composition of any fixed number of exponential functions.) Thus the computational complexities of deciding Th($\mathbf{R}, +$) and Th($\mathbf{R}, +, \cdot$) have been roughly classified as lying between exponential time and exponential space.

One consequence of results such as Theorem 1.1 has been to blur the classical distinction between decidable and undecidable problems. The fact that Th($\mathbf{R}, +$) and Th($\mathbf{R}, +, \cdot$) are decidable is of little use in designing practical decision algorithms. The exponential growth of the time required to accept Th($\mathbf{R}, +$) suggests that any decision procedure for this problem will use hopelessly large amounts of time on relatively short sentences, and therefore that Th($\mathbf{R}, +$) is "practically undecidable" even though it is technically decidable. Even though the value of c and the density of sentences for which the theorem applies have not yet been determined well enough to draw solid conclusions, the term practical undecidability seems apt, since classical undecidability results are prone to similar objections. At the very least, an exponential lower bound on the time complexity of a problem serves as a warning not to seek a uniformly efficient decision algorithm but either to settle for an algorithm which does not work in all cases or to simplify the problem so that it becomes tractable. The implications of lower bounds such as Theorem 1.1 are discussed in more depth by Rabin [Rab4].

To demonstrate that it is possible to prove an astronomical lower bound on the complexity of a decidable theory, even when the length of sentences is restricted, one example has been studied in detail. This example is the weak monadic second-order theory of one successor (WS1S) restricted to sentences of length 616 or less. Decidability of WS1S in general is proved by Büchi [Bu] and Elgot [Elg]. For a particular model of computation, the *logical network* model, which is appropriate for measuring the computational complexity of finite decision problems, Meyer and Stockmeyer [Sto1] show that if a logical network decides truth or falsity of sentences of length 616 or less in WS1S, then the size of the network must exceed the size of the known universe. This result, together with more discussion of the logical network model, is the subject of §7.4. A paper of Ehrenfeucht [Ehr] (originally written in 1967) must also be mentioned as an early influence in this context. It is shown there that if the first-order theory of the integers with addition and multiplication is made decidable by requiring that all quantifiers be bounded by constants defined using iterated exponential notation (e.g., 2^{7^3}), then the sizes of logical networks which decide truth of sentences of length n must grow faster than any polynomial in n.

Beginning with papers of Cobham [Cob] and Edmonds [Edm], the class of "practically decidable" problems has been identified with the class P of problems which can be solved in polynomial time, that is, in time cn^k for some constants c and k, where n is the length of the input. Of course, this identification must be taken with several grains of salt, since an exponential time algorithm may be preferable in practice to an algorithm with running time n^{100}, at least for small enough n.

However, as a mathematical concept, the class P has proven to be a useful one. One reason is that P, like the class of decidable problems, is invariant under a variety of reasonable choices for the model of computation, e.g., Turing machines or random-access machines. After a problem has been found to be decidable, a reasonable next question to resolve is whether the problem belongs to P. If so, one can then seek more precise information about the time required to solve the problem, e.g., whether time proportional to n, or $n \log n$, or n^2, etc., is sufficient. The search for efficient algorithms for problems in P, both general techniques and algorithms for specific problems, is an active and interesting branch of theoretical computer science which lies outside the scope of this paper; see, for example, [AHU], [PaS], and [Tarj].

Unfortunately, there are an embarrassingly large number of important problems whose membership in P is unresolved. Work initiated by Cook [Co1] has decreased the embarrassment somewhat by showing that many of these problems can be grouped into classes such that either all of the problems in a class belong to P or none of them do. Again, we use an example. Let SAT denote the set of formulas $F(y_1, \dots, y_m)$ written in propositional calculus which are satisfiable, i.e., for which there is an assignment of truth values to the propositional variables $y_1, \dots, y_m$ such that F becomes true. An obvious algorithm is to try all 2^m assignments; since m will, in general, increase as the length of F increases, this is not a polynomial time algorithm. No polynomial time algorithm for SAT is known, but there is no proof that one does not exist. However, SAT can be solved in "polynomial time" by a nondeterministic Turing machine. The class NP of problems solvable in polynomial time by nondeterministic Turing machines is defined in §2. For now, we can define NP to be the class of decision problems (formally, sets of words) A which can be specified by

$$(1.1) \qquad x \in A \quad \text{iff} \quad (\exists y)[|y| \leq p(|x|) \text{ and } R(x, y)]$$

for some polynomial p and some binary relation R on words which is computable by an ordinary (deterministic) Turing machine in polynomial time. It is not hard to see that SAT $\in$ NP: x is the formula F being tested for satisfiability, the word y specifies an assignment of truth values to the variables in x, and $R(x, y)$ is true iff the formula x is true under the assignment y. Another example of a problem in NP which is not known to be in P is HC, the set of all undirected graphs which have a Hamiltonian cycle, i.e., a cycle which visits every vertex of the graph exactly once. In this case, the word y would specify a cyclic permutation of the vertices of the graph x, and $R(x, y)$ would check that y specifies a Hamiltonian cycle in x. In general, NP is the class of sets A such that $x \in A$ iff there is a short "proof" y of membership of x in A, where "short" means that $|y| \leq p(|x|)$ for some polynomial p, and such that the validity of the proof can be checked by a relation R computable in polynomial time. Even though a given proof can be checked in polynomial time, there are $2^{p(|x|)}$ potential proofs to be checked, so the algorithm which checks all potential proofs does not run in polynomial time. Of course, this does not rule out the possibility that there is some more clever way of determining whether $x \in A$ in time polynomial in $|x|$, and the question of whether or not P = NP is one of the central open questions in complexity theory.

The importance of the P vs. NP question is due to results of the following type which were first proved independently by Cook [Co1] and Levin [Lev]. The following result, for example, shows that the complexities of SAT and HC are closely related and implicitly classifies them as being among the most complex problems in NP.

THEOREM 1.2 (COOK [Co1], KARP [Ka1]). SAT $\in$ P *iff* HC $\in$ P *iff* P = NP.

There are now known to be a few hundred natural decision problems A with the property that $A \in$ P iff P = NP [GaJ].

We now describe informally the concepts which are used in the proofs of Theorems 1.1 and 1.2 and related results. These are described more formally in later sections of the paper. One concept, borrowed from recursive function theory [Rog], is reducibility. A set A is many-one reducible to a set B if there is a computable function f such that $x \in A$ iff $f(x) \in B$. Then, for example, the undecidability of A implies the undecidability of B. In 1971, Cook [Co1] made a key observation concerning reducibility: if one also has an upper bound on the resources used in computing f, then one can relate the complexity of accepting the set B to the complexity of accepting A. Informally,

$$\text{(1.2)} \qquad \text{“complexity of } A\text{”} \le \text{“complexity of } B\text{”} + \text{“complexity of } f\text{”},$$

since from any algorithm M_B which accepts B we can obtain an algorithm M_A which accepts A: given an input x, the algorithm M_A first computes $f(x)$ and then applies M_B to $f(x)$. In applications of efficient reducibility, the resources used in computing f are negligible compared to what we are trying to establish about the complexities of A and B. Let us informally write $A \le_{\text{eff}} B$ to indicate that A is reducible to B via a function f which is so efficiently computable that its complexity can be ignored. For example, Theorem 1.2 is proved by first observing that SAT and HC belong to NP, and then showing that if A is any decision problem in NP then $A \le_{\text{eff}}$ SAT (Cook [Co1]) and $A \le_{\text{eff}}$ HC (Karp [Ka1]), where in this case $A \le_{\text{eff}} B$ means that A is reducible to B via a function f which is computable in polynomial time.

Although Cook and Karp did not use efficient reducibility to prove explicit lower bounds, it was not long before others, beginning with Meyer and Stockmeyer [MeS], noticed that (1.2) could be used in this way. If we somehow knew that a set A were complex, then we could show other sets B to be complex by efficiently reducing A to B. The existence of complex sets goes back to the early years (1960's) of complexity theory and the important papers of Rabin [Rab1] and Hartmanis and Stearns [HarS] which show that the complexity of decision problems can be as large as any recursive function. In particular, Hartmanis and Stearns [HarS] show that there is a set of words A_{exp} such that any Turing machine which accepts A_{exp} requires time $d^{|x|}$ for infinitely many inputs x, where $d > 1$ is a constant; moreover, A_{exp} is accepted by a Turing machine which uses time at most 2^n on any input x of length n. This is proved by showing that a Turing machine M running in time 2^n can diagonalize and differ from all Turing machines which run within time d^n for a suitably small constant $d > 1$. Then A_{exp} is the set of words accepted by M. (See [AHU, Chapter 11] or [HoU, Chapter 12] for details.)

To prove Theorem 1.1, Fischer and Rabin show that $A_{\text{exp}} \le_{\text{eff}} \text{Th}(\mathbf{R}, +)$, where again $\le_{\text{eff}}$ is polynomial time reducibility. Actually, they show that any member of a large class of sets is efficiently reducible to $\text{Th}(\mathbf{R}, +)$. In proofs of this type, one usually chooses the class to be a *complexity class*, i.e., the class of all sets accepted by Turing machines running within some prescribed time or space bound. In Fischer and Rabin's case, this class is the class of all sets which can be accepted within time 2^n. They show that for any set A in this class and for any input word x, there is a sentence $s_{A,x}$ in the language of $\mathrm{Th}(\mathbf{R}, +)$ such that $x \in A$ iff $s_{A,x} \in \mathrm{Th}(\mathbf{R}, +)$. Moreover, $s_{A,x}$ is computable from x in time polynomial in $|x|$, and there is a constant b_A depending only on A such that $|s_{A,x}| \leq b_A \cdot |x|$ for all x. In particular, $A_{\mathrm{exp}} \leq_{\mathrm{eff}} \mathrm{Th}(\mathbf{R}, +)$ and Theorem 1.1 follows as outlined above; i.e., if there were a fast algorithm for $\mathrm{Th}(\mathbf{R}, +)$ then there would be a fast algorithm for A_{exp}, contradicting the fact that A_{exp} requires time d^n for infinitely many inputs. The constant c in Theorem 1.1 depends on the dilation constant b_A in the reduction and on the constant d such that A_{exp} requires time d^n. The reduction is reminiscent of the "arithmetizations" of Turing machines used in the early undecidability proofs of Church and Turing (see [Dav]), although the requirement that the reducibility be efficiently computable introduces some new complications. Details can be found in [FiR] or [HoU, §13.6].

A general scenario for classifying problems goes as follows. Suppose that one has a specific decision problem (i.e., for each instance of the problem there is a yes/no answer) which is viewed as a recognition problem for a set B of words. The goal is to find complexity classes $\mathscr{C}_{\mathrm{lower}}$ and $\mathscr{C}_{\mathrm{upper}}$ such that

(1) $\mathscr{C}_{\mathrm{lower}} \leq_{\mathrm{eff}} B$ (i.e., every set A in $\mathscr{C}_{\mathrm{lower}}$ is efficiently reducible to B), and

(2) B belongs to $\mathscr{C}_{\mathrm{upper}}$.

Of course, it is desirable to make the two classes $\mathscr{C}_{\mathrm{lower}}$ and $\mathscr{C}_{\mathrm{upper}}$ as close as possible. Ideally, if $\mathscr{C}_{\mathrm{lower}} = \mathscr{C}_{\mathrm{upper}} = \mathscr{C}$, then we say that B is *$\mathscr{C}$-complete*. If $\mathscr{C}_{\mathrm{lower}}$ is known to contain complex sets (say, by invoking the theorem of Hartmanis and Stearns), then one can infer a lower bound on the complexity of accepting B, as outlined above for the example $\mathrm{Th}(\mathbf{R}, +)$.

Sometimes one shows that B is $\mathscr{C}$-complete where the class $\mathscr{C}$ is not presently known to contain complex sets. Even in this case there is a sense in which such a result classifies B. Several open questions in automata-based complexity theory ask, for certain complexity classes $\mathscr{D}$ and $\mathscr{C}$, whether or not $\mathscr{D} = \mathscr{C}$. Quite often, inclusion is known in one direction, say, that $\mathscr{D} \subseteq \mathscr{C}$. If B is $\mathscr{C}$-complete and $\mathscr{D}$ satisfies certain technical conditions related to the particular efficient reducibility being used, then $B \in \mathscr{D}$ iff $\mathscr{D} = \mathscr{C}$. To answer the general question, "Does every set in $\mathscr{C}$ also belong to $\mathscr{D}$?" it is sufficient to focus on a single complete set B and answer the question "Does B belong to $\mathscr{D}$?". Theorem 1.2 is an example of such a classification where $\mathscr{D} = \mathrm{P}$, $\mathscr{C} = \mathrm{NP}$, and SAT and HC are both NP-complete. Following the work of Cook and Karp, complete problems have been found for other interesting complexity classes such as P, polynomial space, and exponential time.

We now outline the remainder of the paper. §2 contains basic definitions, including those of Turing machines, efficient reducibilities, and complete problems. In §3 we state some general consequences of a problem being complete in a complexity class. §4 gives examples of complete problems in four important complexity classes. §5 touches on some other interesting complexity classes, such as classes defined by probabilistic Turing machines and by parallel models of computation. In §6 we survey a number of results giving upper and lower bounds on the complexities of decision problems from logic, game theory, formal language theory, and algebra. §7 discusses some related issues. §8 contains a few closing remarks. Most of the paper is carried out at a level of formality sufficient to state results precisely. Space does not permit the inclusion of full proofs, although a few proofs are sketched to give the flavor of how results of the form $\mathscr{C} \leq_{\text{eff}} B$ are proved. Readers who would prefer a less formal introduction to computational complexity theory should see the Turing Award papers of Cook [Co6] and Karp [Ka2]. No attempt has been made to give an exhaustive list of known lower bounds and complete problems. Rather, the goal has been to mention early seminal papers as well as representative later papers in various subareas. Historically, the period covered by this article begins with Cook's 1971 paper [Co1] which gave the first example of an NP-complete problem, although much foundational work occurred earlier. Hartmanis [Har] gives a subjective account of this earlier period.

§2. Definitions.

2.1. *Decision problems and encodings.* Most of the computational problems considered here are viewed as problems of recognizing particular sets of words. If Σ is a finite alphabet, Σ^* denotes the set of finite words, i.e., finite strings of symbols, over Σ. A *decision problem*, or simply *problem*, is a subset of Σ^* for some finite Σ. In the literature, a set of words is also often called a *language*. In reality, a decision problem is usually defined as the problem of recognizing some specific set of mathematical objects such as true logical formulas, graphs having a certain property, etc. This requires encoding each object as a string of symbols over some finite alphabet. In the case of Th($\mathbf{R}$, +), for example, each well-formed sentence is already a string of symbols over a finite alphabet containing $+$, $\wedge$, $\rightarrow$, $\exists$, etc., except that as the length of sentences grows, the number of variables need not remain fixed. This is handled by tagging each distinct variable symbol with a different string over $\{0, 1\}$, that is, distinct variables are $v0, v1, v10, v11, v100, \ldots$. We prefer not to get bogged down in the details of encodings in this paper since in each case there is an obvious and natural encoding, and the results are invariant under minor differences in the encoding used.

The length of a word x is denoted $|x|$. If $A \subseteq \Sigma^*$ is a problem, $\bar{A} = \Sigma^* - A$ denotes the *complement* of A. If $\mathscr{C}$ is a class of problems, co-$\mathscr{C} = \{\bar{A} \mid A \in \mathscr{C}\}$. $\mathbf{N}$ denotes the nonnegative integers and $\mathbf{R}$ denotes the real numbers.

2.2. *Turing machines and complexity classes.* The formal model of computation used here is the Turing machine. In the literature, this choice is made in part for historical reasons, but mainly because the simplicity of the definition of Turing machines makes them convenient to work with. The results obtained in terms of Turing machines usually carry over to more realistic models of computation; we discuss this further below. For completeness we review the definition of Turing machines, although we assume that the reader is already somewhat familiar with this model. More information, if needed, can be found in [AHU] and [HoU]. We first define *nondeterministic Turing machines* (NTM's) with a finite number of read/write work-tapes and a 2-way read-only input-tape. "Nondeterministic" means that there may be several ways for the computation to proceed. A particular NTM M is specified by finite sets Q (the states), Σ (the input alphabet), Γ (the work-tape alphabet), designated states q_0 (the initial state) and q_{a} (the accepting state), an endmarker $\$ \notin \Sigma$, an integer $k \geq 1$ (the number of work-tapes), and a transition function δ,

$$\delta: Q \times (\Sigma \cup \{\$\}) \times \Gamma^k \rightarrow \text{powerset}(Q \times \Gamma^k \times \{\text{left}, \text{right}, \text{stationary}\}^{k+1}),$$

where powerset(S) denotes the set of subsets of S. If the machine is in state q scanning σ on the input-tape and scanning γ_i on the ith work-tape for $1 \leq i \leq k$, and if

$$(q', \gamma'_1, \ldots, \gamma'_k, m_1, \ldots, m_{k+1}) \text{ belongs to } \delta(q, \sigma, \gamma_1, \ldots, \gamma_k),$$

then M can in one step change its state to q', print γ'_i on the ith work-tape and move the head in direction m_i, and move the input head in direction m_{k+1}. A *configuration* of M is specified by giving the nonblank contents of all tapes, the positions of all heads, and the state. Write $C \vdash_M C'$ if configuration C can reach C' in one step of M. The machine is given input $x \in \Sigma^*$ by writing $\$x\$$ on the input-tape with the head scanning the first symbol of x; the state is q_0 and all work-tapes are blank. Let Init_x denote this initial configuration on input x. An *accepting computation* of M on input x is a sequence $C_0, C_1, \ldots, C_t$ of configurations such that $C_0 = \text{Init}_x$, $C_i \vdash_M C_{i+1}$ for $0 \leq i < t$, the state of C_t is q_a, and the state of C_i is not q_a for $0 \leq i < t$. *M accepts* x iff there exists an accepting computation of M on x. Let $L(M)$ denote the subset of Σ^* that M accepts.

An NTM is a *deterministic Turing machine* (DTM) if the range of its transition function δ contains only singleton sets or the empty set. In other words, for each configuration C of a DTM there is at most one C' with $C \vdash_M C'$, so for each input x the accepting computation of a DTM is unique when it exists.

The *time* of an accepting computation $C_0, C_1, \ldots, C_t$ is t; the *space* of the computation is the number of work-tape squares visited by heads on work-tapes during the computation. Let $F: \mathbf{N} \rightarrow \mathbf{R}$ and let M be a Turing machine. *M accepts within time* (*space*) $F(n)$ if for each $x \in L(M)$ there is an accepting computation of M on x such that the time (space) of the computation does not exceed $F(|x|)$. Let NTIME($F(n)$) (resp., NSPACE($F(n)$)) denote the class of problems accepted by NTM's which accept within time (resp., space) $F(n)$. Let DTIME($F(n)$) (resp., DSPACE($F(n)$)) denote the class of problems accepted by DTM's which accept within time (resp., space) $F(n)$.

In specifying resource bounds, we often use the O-notation. If $G(n)$ is a function from $\mathbf{N}$ to $\mathbf{R}$, $\mathrm{O}(G(n))$ is the set of functions G' satisfying $G'(n) \leq c \cdot G(n)$ for some positive real constant c. The O-notation is used inside NTIME, DTIME, etc. in the obvious way. For example, DTIME($2^{\mathrm{O}(n)}$) denotes the union of DTIME(2^{cn}) taken over all constants c. We also let poly($G(n)$) abbreviate $\mathrm{O}(G(n)^{\mathrm{O}(1)})$, that is, the class of functions which are bounded above by some polynomial function of G. Of particular interest are complexity classes defined by resource bounds which grow logarithmically or polynomially in n:

$$\text{DLOG} = \text{DSPACE}(\log n), \qquad \text{NLOG} = \text{NSPACE}(\log n),$$
$$\text{P} = \text{DTIME}(\text{poly}(n)), \qquad \text{NP} = \text{NTIME}(\text{poly}(n)),$$
$$\text{PSPACE} = \text{DSPACE}(\text{poly}(n)).$$

For definiteness, we let $\log n$ denote the base 2 logarithm for $n \geq 2$, and $\log 0 = \log 1 = 1$. Note that since the space bound is imposed only on the work-tapes, but not on the input-tape, it makes sense to consider space bounds such as $\log n$ which grow much more slowly than n. We henceforth assume that all time bounds satisfy $T(n) \geq n$ and all space bounds satisfy $S(n) \geq \log n$. We also assume that every time bound $T(n)$ (resp., every space bound $S(n)$) has the property that there is a DTM which, when started on any input x, runs for $T(|x|)$ steps and halts (resp., uses $S(|x|)$ squares on its work-tapes and halts). Such functions are called *fully constructible* (see, for example, [HoU, Chapter 12]). We impose the constructibility condition to rule out pathological functions $F(n)$ such that the time or space required to compute F grows much faster than F itself. This condition is of no concern to us since all the familiar examples of resource bounds such as logarithmic, polynomial, and exponential functions are constructible.

The relationships between nondeterministic and deterministic time, between nondeterministic and deterministic space, and between time and space are presently not well understood. Some known relationships are given next.

(2.1) *Nondeterministic versus deterministic time*:

$$\text{DTIME}(T(n)) \subseteq \text{NTIME}(T(n)), \tag{2.1a}$$

$$\text{NTIME}(T(n)) \subseteq \text{DTIME}(2^{O(T(n))}). \tag{2.1b}$$

(2.2) *Nondeterministic versus deterministic space*:

$$\text{DSPACE}(S(n)) \subseteq \text{NSPACE}(S(n)), \tag{2.2a}$$

$$\text{NSPACE}(S(n)) \subseteq \text{DSPACE}(S(n)^2). \tag{2.2b}$$

(2.3) *Time versus space*:

$$\text{NTIME}(T(n)) \subseteq \text{DSPACE}(T(n)), \tag{2.3a}$$

$$\text{DTIME}(T(n)) \subseteq \text{DSPACE}(T(n)/\log T(n)), \tag{2.3b}$$

$$\text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))}). \tag{2.3c}$$

(2.1a) and (2.2a) are immediate. (2.1b) and (2.3a) are proved by having the DTM try all possible computations of the NTM whose time does not exceed $T(n)$ and see if any of these computations are accepting. (2.2b) is proved by Savitch [Sav]. (2.3b) is proved by Hopcroft, Paul and Valiant [HoPV]. (2.3c) follows since there are at most $2^{cS(n)}$ configurations which use at most $S(n)$ squares on the work-tapes, for some constant c depending on the NTM; if the NTM has an accepting computation on input x there must be one in which no configuration repeats, so the time bound follows.

The questions of closing the large gaps in (2.1), (2.2) and (2.3) have been focused down to several open questions of central importance such as:

Does P = NP?

Does P = PSPACE?

Does DLOG = NLOG?

Is P $\subseteq$ DSPACE(poly($\log n$))?

These are apparently very difficult questions. Paul, Pippenger, Szemerédi and Trotter [PPST] have made the first progress toward showing that nondeterministic time is more powerful than deterministic time by proving that there is a problem in NTIME(O(n)) which is not in DTIME(O(n)). Even this slight separation between

NTIME and DTIME required combining several tools from complexity theory with a clever combinatorial lemma. Technical evidence suggests that their method cannot push the separation much farther and, in particular, is not strong enough to prove that P $\neq$ NP.

2.3. *Other models of computation.* The Turing machine is obviously not a very realistic model of computation since its storage medium is linear tapes. Other formal models which more closely mimic the capabilities of actual digital computers have been defined and studied. One such model is the *random access machine* (RAM). The storage of a RAM consists of an infinite sequence of numbered "registers", $r_1, r_2, \ldots$, each of which can hold an integer. A particular RAM is specified by a finite sequence of labelled instructions such as

$r_i \leftarrow 8$ (load the integer 8 into register r_i),
$r_i \leftarrow r_j + r_k$ (add the contents of registers r_j and r_k and store the result in r_i),
$r_i \leftarrow r_{r_j}$ (load the contents of r_k into r_i where k is the contents of r_j),
go to X if $r_i < r_j$ (transfer execution of the program to the statement labelled X if the contents of r_i is less than the contents of r_j),
read r_i (read the next input symbol into register r_i).

RAM programs can also be made nondeterministic in the obvious way. Given reasonable definitions of the time and space of RAM computations, it is not difficult to show that RAM's and Turing machines can simulate one another with time bounds preserved to within composition with a polynomial and space bounds preserved to within a constant factor. More details of the definition of RAM's, definitions of time and space for RAM's, and details of the simulations can be found in [AHU, Chapter 1] or [CoR]. In particular, the complexity classes P, NP and PSPACE do not change if defined in terms of RAM's rather than TM's. It also follows that if we prove a lower bound of the form c^n on the TM time or space complexity of some problem, a lower bound of the same form, with possibly a different constant $c > 1$, holds also for RAM's. To this extent, the definitions and results do not depend on the particular choice of Turing machines as the model of computation. Indeed, there is a thesis, a complexity-theoretic version of the Church-Turing thesis, which states that a Turing machine can simulate any reasonable model of computation with at most a polynomial increase in time and space. There is no precise definition of "reasonable" but the idea is that the model should not be able to do an unrealistic amount of computation in one step. For example, a machine which could add numbers of length 2^n in one step would not be considered reasonable.

2.4. *Efficient reducibility.* We must first extend the definition of DTM's to permit them to compute functions rather than just accept sets of words. For function computation, a DTM also has an output-tape which is scanned by a head which can only move from left to right and which can print symbols from some output alphabet Δ. The transition function δ is augmented to specify at each step either the symbol from Δ to be printed (which is followed by the head moving one square to the right) or the fact that the output head does not print or move. In defining the space of a computation, space used on the output-tape is not included, so that we can consider functions being computed within space which is smaller than the length of the output.

Let $f\colon \Sigma^* \to \Delta^*$ and let M be a DTM. *M computes f* within time (space) $F(n)$ if (i) M accepts within time (space) $F(n)$, (ii) M accepts Σ^*, and (iii) for each $x \in \Sigma^*$, if $C_0, \ldots, C_t$ is the unique accepting computation of M on input x then $f(x)$ is the word written on the output tape in configuration C_t.

Let *logspace* (resp., *polytime*) be the class of functions computable by DTM's within space $\log n$ (resp., within time poly(n)). Let $A \subseteq \Sigma^*$ and $B \subseteq \Delta^*$ be problems. A is *logspace-reducible* to B, written $A \leq_{\log} B$ (resp., A is *polynomial-time-reducible* to B, written $A \leq_{\mathrm{p}} B$), if there is a function $f\colon \Sigma^* \to \Delta^*$ with $f \in$ logspace (resp., $f \in$ polytime) such that $x \in A$ iff $f(x) \in B$ for all $x \in \Sigma^*$. If moreover there is a constant $b > 0$ such that $|f(x)| \leq b|x|$ for all x with $|x| > 0$, then we write $A \leq_{\text{log-lin}} B$ (resp., $A \leq_{\text{p-lin}} B$).

Polynomial-time reducibility was introduced by Cook [Co1] and Karp [Ka1]. The reducibilities defined above are resource bounded versions of the many-one reducibility of recursive function theory [Rog]. One can similarly define polynomial time bounded versions of other types of reducibility such as Turing reducibility and truth-table reducibility. In fact, Cook's original paper on the subject [Co1] used polynomial time Turing reducibility. Polynomial time many-one reducibility, $\leq_{\mathrm{p}}$, was defined and used by Karp [Ka1] shortly thereafter. For most applications, many-one reducibility suffices. The relative power of various types of polynomial time reducibilities, when viewed as relations on the recursive sets, is studied by Ladner, Lynch and Selman [LaLS]. Logspace reducibility was introduced independently by Cook [Co3], Jones [Jon], and Meyer and Stockmeyer [StoM]. Since logspace $\subseteq$ polytime (cf. (2.3c)), $A \leq_{\log} B$ implies $A \leq_{\mathrm{p}} B$. Since polytime is clearly closed under functional composition and since Jones [Jon] and Meyer [StoM] show that logspace is closed under composition, $\leq_{\log}$, $\leq_{\text{log-lin}}$, $\leq_{\mathrm{p}}$ and $\leq_{\text{p-lin}}$ are all transitive relations on problems. In the sequel, we let $\leq_{\text{eff}}$ denote one of $\leq_{\log}$, $\leq_{\text{log-lin}}$, $\leq_{\mathrm{p}}$, or $\leq_{\text{p-lin}}$.

Let B be a problem and let $\mathscr{C}$ be a class of problems.

(1) $\mathscr{C} \leq_{\text{eff}} B$, and we say that B is *$\mathscr{C}$-hard* with respect to $\leq_{\text{eff}}$, if $A \leq_{\text{eff}} B$ for all $A \in \mathscr{C}$.

(2) B is *$\mathscr{C}$-complete* with respect to $\leq_{\text{eff}}$ if both $\mathscr{C} \leq_{\text{eff}} B$ and $B \in \mathscr{C}$.

When we say that B is $\mathscr{C}$-hard or $\mathscr{C}$-complete without specifying a particular reducibility, $\leq_{\mathrm{p}}$ reducibility is understood, except when $\mathscr{C}$ is either NLOG or P in which case $\leq_{\log}$ is understood. (The reason for the exception should be clear because of the general requirement that the resources used in computing the reducibility function should be smaller than the resource bounds which define the complexity class.) For most of the polynomial time reductions found in the literature, closer inspection reveals that the function can actually be computed in logarithmic space, although this is a level of detail which we avoid in this paper. In general we use $\leq_{\log}$ only when this stronger reducibility is needed to obtain the desired result.

§3. Consequences of efficient reducibility.

3.1. *A complete problem represents a class.* As outlined in the Introduction, a consequence of a problem B being NP-complete is that $B \in \mathrm{P}$ iff $\mathrm{P} = \mathrm{NP}$. This is a general phenomenon, as pointed out by Book [Bo] using the following definition. A class $\mathscr{D}$ of problems is *closed under* $\leq_{\text{eff}}$ provided that if $B \in \mathscr{D}$ and $A \leq_{\text{eff}} B$ then $A \in \mathscr{D}$. For example, P is closed under $\leq_{\text{p}}$. The proof of the following is immediate from definitions.

PROPOSITION 3.1. *Let $\mathscr{D}$ and $\mathscr{C}$ be classes of problems such that $\mathscr{D} \subseteq \mathscr{C}$ and $\mathscr{D}$ is closed under $\leq_{\text{eff}}$, and say that B is $\mathscr{C}$-complete with respect to $\leq_{\text{eff}}$. Then $B \in \mathscr{D}$ if and only if $\mathscr{D} = \mathscr{C}$.*

3.2. *Lower bounds on computational complexity.* As outlined in the Introduction, if $\mathscr{C} \leq_{\text{eff}} B$ and if $\mathscr{C}$ is known to contain sufficiently complex problems, then one can infer an explicit lower bound on the computational complexity of B. In order to make this formal, we need two lemmas.

When $A \leq_{\text{eff}} B$ the first lemma provides an upper bound on the complexity of A in terms of an upper bound on the complexity of B. Two relevant parameters of $\leq_{\text{eff}}$ are the resources used in computing the reducibility function f and the extent to which f increases the length of its argument.

LEMMA 3.1. *Let A and B be problems and let S and T be nondecreasing functions from reals to reals.*

(1) *If $A \leq_{\text{eff}} B$ via f, and $L: \mathbf{N} \to \mathbf{R}$ is such that $|f(x)| \leq L(|x|)$ for all x, then*

$$B \in \text{DTIME}(T(n)) \textit{ implies } A \in \text{DTIME}(T(L(n)) + p(n))$$

for some polynomial $p(n)$.

(2) *If $A \leq_{\log} B$ via f, and L is as above, then*

$$B \in \text{DSPACE}(S(n)) \textit{ implies } A \in \text{DSPACE}(S(L(n)) + \log n).$$

Moreover, (1) *is true with* NTIME *in place of* DTIME, *and* (2) *is true with* NSPACE *in place of* DSPACE.

PROOF (sketch). (1) Say that M is a DTM which accepts B within time $T(n)$. A DTM M' accepts A as follows: given input x, first compute $f(x)$ (this takes time polynomial in $|x|$ and produces a word $f(x)$ of length $\leq L(|x|)$) and then apply M to $f(x)$ (this takes time at most $T(L(|x|))$ by assumption). The proof for NTIME is identical.

(2) The obvious approach of part (1) does not work here because writing $f(x)$ on a work-tape might exceed the desired space bound $S(L(n)) + \log n$. However, it is possible to simulate the computation of M on $f(x)$ by simply keeping track of the position of M's input head on the input $f(x)$ (which can be done using an integer with at most $\log n$ digits) and then recomputing the symbols of $f(x)$ whenever they are needed in this simulation. This technique is due to Jones and Meyer [Jon], [StoM]. $\square$

The second lemma concerns "hierarchy" theorems for time and space complexity which state that for small increases in the growth rate of the resource bounding function $S(n)$ or $T(n)$, more problems can be accepted.

We use the following terminology: For a problem A and a function $G(n)$, the phrase "A requires nondeterministic time $G(n)$ i.o." means that if A is accepted by an NTM within time $T(n)$ then $T(n) \geq G(n)$ for infinitely many n (i.o. stands for "infinitely often"). Similar terminology is obtained by replacing nondeterministic by deterministic and/or replacing time by space.

The following lemma, which concerns hierarchies for NTM's, is due to Seiferas, Fischer and Meyer [Se], [SeFM] and refines earlier results of Ibarra [Ib] and Cook [Co2].

LEMMA 3.2. (1) *Let $T_1(n)$ and $T_2(n)$ be functions such that*

$$\lim_{n \to \infty} T_1(n + 1)/T_2(n) = 0.$$

There is a problem $A \in \mathrm{NTIME}(T_2(n))$ such that A requires nondeterministic time $T_1(n)$ i.o.

(2) *Let $S_1(n)$ and $S_2(n)$ be functions such that*

$$\lim_{n \to \infty} S_1(n + 1)/S_2(n) = 0.$$

There is a problem $A \in \mathrm{NSPACE}(S_2(n))$ such that A requires nondeterministic space $S_1(n)$ i.o.

Similar hierarchies are known for DTM's [HarS], [StHL] (see also [HoU, Chapter 12]) although the known time hierarchy is slightly coarser in the deterministic case.

To illustrate the use of Lemmas 3.1 and 3.2, we restate and outline the proof of Theorem 1.1.

THEOREM 3.1. *There is a constant $c > 1$ such that* $\mathrm{Th}(\mathbf{R}, +)$ *requires nondeterministic time c^n i.o.*

PROOF (sketch). The key fact, proved by Fischer and Rabin [FiR], is that

$$\mathrm{NTIME}(2^n) \leq_{\text{p-lin}} \mathrm{Th}(\mathbf{R}, +). \tag{3.1}$$

The proof of (3.1) is quite involved and we do not attempt to reproduce it here. Our goal is only to show how (3.1) is used to prove a lower bound on $\mathrm{Th}(\mathbf{R}, +)$.

Using Lemma 3.2(1), let A be a problem in $\mathrm{NTIME}(2^n)$ such that A requires nondeterministic time $2^{n/2}$ i.o. From (3.1), $A \leq_{\text{p-lin}} \mathrm{Th}(\mathbf{R}, +)$ via a function f such that $|f(x)| \leq b|x|$ for some constant b. Choose c such that $0 < c < 2^{1/2b}$, and assume for contradiction that $\mathrm{Th}(\mathbf{R}, +) \in \mathrm{NTIME}(c^n)$. Using Lemma 3.1(1) it follows that $A \in \mathrm{NTIME}(c^{bn} + p(n))$, where $p(n)$ is a polynomial. But $c^{bn} + p(n) < 2^{n/2}$ for all but finitely many n, contradicting the fact that A requires nondeterministic time $2^{n/2}$ i.o. $\square$

The reader should have no trouble generalizing this proof to other situations. For example, if B is a problem such that $\mathrm{DSPACE}(2^n) \leq_{\log} B$, and moreover for each $A \in \mathrm{DSPACE}(2^n)$ there is a function f and a constant b such that $A \leq_{\log} B$ via f and $|f(x)| \leq b|x|^2$, then one concludes that there is a constant $c > 1$ such that B requires deterministic space $c^{n^{1/2}}$ i.o.

We should point out that it is not strictly necessary to use the hierarchy theorems in proving lower bounds. Once one has shown that $\mathscr{C} \leq_{\text{eff}} B$ for a sufficiently rich class $\mathscr{C}$, then it should be possible to carry out a direct diagonalization (in the spirit of classical undecidability results, cf. [Dav]) to establish a lower bound. In fact, this is how Fischer and Rabin proceed in their proof of Theorem 3.1. We prefer to make explicit use of the hierarchy theorems.

A lower bound on the computational complexity of a problem sometimes implies a lower bound on some other measure of the complexity of the problem. For a logical decision problem another measure is the length of proofs required to prove the true statements. Suppose that AX is a system of axioms for Th($\mathbf{R}$, +) such that (i) a sentence s is provable from AX iff s is true, and (ii) AX, when viewed as a set of words, belongs to P. Then there is a constant $c > 1$ and an infinite set $S \subseteq$ Th($\mathbf{R}$, +) of true sentences such that, for all $s \in S$, the shortest proof of s from AX has length at least $c^{|s|}$. Briefly, this is true because if proofs were always "short" then there would be a "fast" NTM which accepts Th($\mathbf{R}$, +): given an input s, such an NTM would nondeterministically guess a "short" string ρ and then verify that ρ is a valid proof of s from AX. Fischer and Rabin [FiR] discuss this further.

3.3. *Speed-up.* Given a sufficiently complex problem A and given an algorithm M which accepts A, one can sometimes find a more clever algorithm M' which runs much faster than M on infinitely many inputs. Blum in [Blu1] and [Blu2] shows that there exist problems with this speed-up property for *any* given M. To simplify the discussion, let us restrict attention to DTM's which halt on all inputs, and let $\text{Time}_M(x)$ denote the number of steps in the computation of DTM M on input x. Let us say that a problem $A \subseteq \Sigma^*$ has *$T(n)$-to-polynomial effective i.o. speed-up* if there is a polynomial $p(n)$ such that from any DTM M which accepts A we can effectively produce a DTM M' which accepts A and an infinite recursive set $U \subseteq \Sigma^*$ such that

$$\text{Time}_M(x) \geq T(|x|) \quad \text{for all } x \in U,$$

and

$$\text{Time}_{M'}(x) \leq p(|x|) \quad \text{for all } x \in U.$$

Blum's work is carried out for a general axiomatic complexity measure. However, the proof of Theorem 3 in [Blu2] can be carried out for the particular measure of DTM time, and the proof yields the following result (this was pointed out to me by A. Meyer).

THEOREM 3.2. *For any constructible $T(n)$, there is a problem A in* DTIME($O(T(n)^2)$) *which has $T(n)$-to-polynomial effective i.o. speed-up.*

However, the problem A in this theorem is constructed specifically to have the speed-up property; it is not a natural problem. Stockmeyer [Sto1], following a suggestion of Meyer, has shown that if $A \leq_{\text{eff}} B$ via f, if f satisfies the mild conditions of being one-to-one and efficiently invertible (all of the known reductions satisfy these conditions), and if A has i.o. speed-up then so does B. (Although the details are carried out in [Sto1] only for the space measure, the details for the time measure are analogous.) Combining this with Theorem 3.2 and results such as (3.1), it follows that natural problems such as Th($\mathbf{R}$, +) have i.o. speed-up. In particular, the following can be shown.

THEOREM 3.3. *There is a constant $c > 1$ such that* Th($\mathbf{R}$, +) *has c^n-to-polynomial effective i.o. speed-up.*

Berman [Ber1] has shown that if a problem is DTIME($T(n)$)-complete where $T(n)$ grows faster than polynomially in n, then the problem has the i.o. speed-up property even without imposing an efficient invertibility condition on the reducibility function.

§4. Complete problems in four important complexity classes.

The purpose of this section is to give examples of complete problems in four important complexity classes, attempt to give the reader some feeling for how such results are proved, and examine the consequences of complete problems to open questions in complexity theory. From the inclusions (2.1), (2.2), and (2.3) is it known that

$$\text{DLOG} \subseteq \text{NLOG} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE}. \tag{4.1}$$

It is presently an open question whether any of these containments are proper. (However it is known that NLOG $\neq$ PSPACE, which follows from (2.2b) and Lemma 3.2(2).) Since it is not known that P $\neq$ PSPACE, we cannot use the method of §3.2 to show that problems complete in these classes require more than polynomial time. However, using the method of §3.1 it is easy to relate the complexities of complete problems to questions of proper inclusion in (4.1). It is an easy consequence of Lemma 3.1 that all five of the classes in (4.1) are closed under $\leq_{\log}$ and that the last three are closed under $\leq_{\text{p}}$. Let $\mathscr{C}_i$ denote the ith class from the left in (4.1) for $1 \leq i \leq 5$. By Proposition 3.1, if B is $\mathscr{C}_i$-complete then $B \in \mathscr{C}_{i-1}$ iff $\mathscr{C}_{i-1} = \mathscr{C}_i$.

In each of the following four subsections we choose as our main example a problem which has been useful as a starting point for proving other problems to be complete. Note that since $\leq_{\log}$ and $\leq_{\text{p}}$ are transitive, once a particular problem A has been proved $\mathscr{C}$-hard, one can show other problems B to be $\mathscr{C}$-hard by showing that $A \leq_{\text{eff}} B$ for the appropriate $\leq_{\text{eff}}$. This is sometimes easier than showing $\mathscr{C} \leq_{\text{eff}} B$ directly.

Since several of our examples involve propositional variables and formulas, we first introduce some terminology for this. Let $\{X_0, X_1, X_2, \ldots\}$ be a set of propositional variables where variables are encoded as described in §2.1 by writing subscripts in binary notation, X0, X1, X10, X11, X100, etc. A *propositional formula* either is a propositional variable or is of the form $(\neg F)$, $(F \to G)$, $(F \wedge G)$, or $(F \vee G)$ where F and G are propositional formulas. An assignment of truth values to the propositional variables in a formula determines a truth value for the formula in the obvious way. A propositional formula is *satisfiable* if the formula is true for some assignment of truth values to its variables. A formula is in *conjunctive normal form* if it is a conjunction of disjunctions of literals, where a *literal* is either a variable or the negation of a variable. Disjunctive normal form is defined dually.

4.1. NLOG-*complete problems.* The first NLOG-complete problem is implicit in a paper of Savitch [Sav], although the terminology of logspace reducibility had not yet been formalized at that time (1970). Savitch's problem, the set of "threadable mazes", is now usually referred to in the literature as the *graph accessibility problem* or GAP: Given a directed graph G and two distinguished vertices s and t, does there exist a directed path in the graph from s to t? More precisely, a graph on m vertices is specified by giving its edge relation as an $m \times m$ matrix E of zeros and ones. For $1 \leq i, j \leq m$, $E(i,j) = 1$ iff there is an edge directed from vertex i to vertex j. E is written as a word in $\{0,1\}^*$ of length m^2 by concatenating its rows. Define GAP to be the set of all graphs such that there is a directed path from vertex 1 to vertex m, where m is the number of vertices in the graph. In other words, $E \in$ GAP iff there is a sequence of integers $z_1, \ldots, z_k$ such that $z_1 = 1$, $z_k = m$, and $E(z_i, z_{i+1}) = 1$ for all $1 \leq i < k$.

THEOREM 4.1. GAP *is* NLOG-*complete.*

PROOF (sketch) (1) GAP $\in$ NLOG. An NTM M can accept GAP within space $\log n$ simply by nondeterministically guessing the sequence $z_1, \ldots, z_k$ which certifies that $E \in \mathrm{GAP}$, and accepting E iff such a certificate is found. During this procedure, the work-tape holds two integers i and j in the range from 1 to m; by writing the integers in binary, this takes space $\log m \le \log|E|$. First M sets $i = 1$. M then nondeterministically chooses a j and checks that $E(i, j) = 1$ (how this is done in space $\log n$ is an easy exercise). If the check succeeds, M sets i to j and continues by guessing another j with $E(i, j) = 1$, and so on. M accepts if ever $j = m$.

(2) $\mathrm{NLOG} \le_{\log} \mathrm{GAP}$. Let $A \in \mathrm{NLOG}$ and let M be an NTM which accepts A within space $\log n$. We must show how to transform each input word x to an edge relation E such that M accepts x iff $E \in \mathrm{GAP}$. Fix an input x of length n and let $\mathscr{M}$ be the set of configurations (head positions and contents of work-tapes) of M such that the space used on work-tapes does not exceed $\log n$. There are at most poly(n) such configurations (cf.(2.3c)) and each configuration C in $\mathscr{M}$ can be encoded as a distinct binary word bin(C) of length $O(\log n)$. Let $z(C)$ be the integer whose binary representation is bin(C). Let m be one more than the largest $z(C)$. Set $E(z(C), z(C')) = 1$ for all C and C' such that $C \vdash_M C'$ when x is written on the input tape of M. Also set $E(1, z(C_0)) = 1$ where C_0 is the initial configuration of M, and set $E(z(C), m) = 1$ for all accepting configurations C. All other entries of E are set to zero. It is clear by the definition of acceptance for NTM's that M accepts x iff $E \in \mathrm{GAP}$. It is not hard to see that the function mapping x to E can be computed in space $\log n$. $\square$

Jones, Lien and Laaser [JoLL] show other problems to be NLOG-complete. One example is the set of unsatisfiable propositional formulas in conjunctive normal form with at most two literals per conjunct. (As discussed shortly in §4.3, if there can be three literals per conjunct the satisfiability problem becomes NP-complete.) Sudborough [Sud] shows that there is a (linear) context-free language which is NLOG-complete.

4.2. P-*complete problems.* Problems which are P-complete are tied to the relationship between time and space. In particular, if A is P-complete then

$$A \in \mathrm{DSPACE}(\mathrm{poly}(\log n)) \quad \text{iff} \quad \mathrm{P} \subseteq \mathrm{DSPACE}(\mathrm{poly}(\log n)).$$

We shall see in §5.3 that P-completeness is also related to the question of whether problems can be solved much faster by using many computing elements in parallel.

The first P-complete problem was exhibited by Cook [Co3]. We give here another problem which was proved P-complete by Ladner [Lad2] and which has been useful in showing other problems to be P-complete. An instance of the *circuit value problem* (CVP) is a sequence of equations among propositional variables $X_1, \ldots, X_m$ where each equation is of one of the forms:

$$X_i = \text{true}, \qquad X_i = \text{false}, \qquad X_i = X_j \mathbin{@} X_k \quad \text{for some } j, k < i,$$

where @ denotes one of the 2-ary Boolean functions (*and*, *or*, *exclusive-or*, etc.), and where each variable appears exactly once on the left-hand side of an equation. Variables can appear any number of times within right-hand sides of equations. Given such a sequence, it is clear how to assign truth values to all the variables in order $X_1, X_2, \ldots, X_m$. The condition $j, k < i$ ensures that there are no cyclic dependencies among the variables. CVP is the set of such sequences of equations such that X_m evaluates to true. (To explain the name "circuit value problem", think of each variable as representing the Boolean logic value on some wire in a combinational logic circuit. Each equation says either that the wire has a fixed value or that the value of the wire should be computed as a Boolean function of other wires.) Clearly CVP $\in$ P. The intuition why CVP is apparently not solvable in poly($\log n$) space is that any algorithm must remember the truth values of variables as they are computed since these values will be needed later to evaluate other variables. Of course, this is not a proof that poly($\log n$) space does not suffice.

THEOREM 4.2. CVP *is P-complete.*

PROOF (sketch). Having noted that CVP $\in$ P is obvious, we need only show that P $\leq_{\log}$ CVP. For this proof, and the proofs in the next two subsections, it is convenient to assume that Turing machines have a particular simple form. A *simple Turing machine* has only one tape which is one-way infinite to the right. The machine is given input x by writing x left-justified on the tape with the head scanning the leftmost symbol of x. For each simple TM M we assume that there is a polynomial $p(n)$ such that M never visits more than $p(|x|)$ squares on its tape. M accepts by returning its head to the leftmost tape square and entering the accepting state q_a. Once in state q_a, we assume that M stays in state q_a without moving the head. It is easy to see that P is the class of problems accepted by simple DTM's which accept in polynomial time. This is true because the information on the tapes of a DTM with many tapes can be stored on the single tape of a simple DTM, and the running time of the simple DTM is at most proportional to the square of the running time of the multi-tape DTM (see Theorem 12.5 in [HoU]).

Let M be a simple DTM which accepts in polynomial time $p(n)$, which for simplicity we assume is the same as the space bound of M. Let Q be the states of M, let Γ be the tape alphabet of M, and let \$ be a symbol not in Q or Γ. A configuration of M on input x (with $n = |x|$) is a word of length $p(n) + 3$ of the form $\$\alpha q\beta\$$ where $q \in Q$ and $\alpha, \beta \in \Gamma^*$. The meaning is that $\alpha\beta$ is written on the tape and M is in state q scanning the leftmost symbol of β. In particular, the initial configuration is $\$q_0x\#\#\#\cdots\#\$$ where $\#$ denotes the blank symbol, and accepting configurations have q_a as their second symbol. Let $C_0, C_1, \ldots, C_{p(n)}$ denote the sequence of configurations in the computation of M on input x and let $c_{i,j}$ denote the jth symbol of C_i for $1 \leq j \leq p(n) + 3$. Since M is deterministic, it is easy to see that for all $i > 1$ and all j, $c_{i,j}$ is determined by $c_{i-1,j-1}$, $c_{i-1,j}$, $c_{i-1,j+1}$, and $c_{i-1,j+2}$. For example, if one of $c_{i-1,j-1}, c_{i-1,j}$, or $c_{i-1,j+1}$ is a state symbol, then $c_{i,j}$ is determined by the transition function of M. If none of $c_{i-1,j-1}$, $c_{i-1,j}$, or $c_{i-1,j+1}$ is a state symbol, then $c_{i,j} = c_{i-1,j}$.

To write a set of equations, we associate a variable $X(i,j,\gamma)$ with each $i\,(0 \leq i \leq p(n))$, $j\,(1 \leq j \leq p(n) + 3)$, and $\gamma \in Q \cup \Gamma \cup \{\$\}$. The intended meaning is that $X(i,j,\gamma)$ is true iff $c_{i,j} = \gamma$. The variables $X(0,j,\gamma)$ associated with the initial configuration C_0 are set to true or false to make these variables encode C_0. Each variable $X(i,j,\gamma)$ with $i > 0$ is defined in terms of some of the variables $X(i-1,\cdot,\cdot)$ associated with C_{i-1}. (In order to bring each equation into the simple form in the definition of CVP, some additional variables must be introduced in defining $X(i,j,\gamma)$ from previous variables.) Finally, we number the variables so that the largest numbered variable is $X(p(n), 2, q_a)$. This variable evaluates to true iff M accepts x. Since the definition of $X(i,j,\gamma)$ in terms of previous variables is essentially the same for all i and j, a DTM can carry out the transformation of x to the sequence of equations while using an amount of tape proportional to the space required to store the indices i and j. By writing the indices in binary, this space is $O(\log n)$. $\square$

Goldschlager [Gol] shows P-completeness for the *monotone circuit value problem*, defined like CVP but allowing variables to be defined from other variables using only $\wedge$ and $\vee$. Other interesting P-complete problems include the problem of determining if a given context-free grammar generates the empty language (Jones and Laaser [JoL]), finding the maximum flow in a network (Goldschlager, Shaw and Staples [GoSS]), checking whether two terms are unifiable—a problem which arises in resolution theorem proving (Dwork, Kanellakis and Mitchell [DKM]), and checking whether a system of linear inequalities with rational coefficients has a rational solution—a special case of linear programming (Dobkin, Lipton and Reiss [DoLR] show P-hardness and Khachiyan [Kh] shows that this problem belongs to P). Many other P-complete problems are surveyed by Hoover and Ruzzo [HooR].

4.3. NP-*complete problems*. The first problem to be proved NP-complete was the satisfiability problem for the propositional calculus (Cook [Co1]). The importance of the concept of NP-completeness was further established by Karp [Ka1], who proved NP-completeness for several classical problems such as checking whether a graph is k-colorable, checking whether a graph has a Hamiltonian cycle, and checking whether a system of linear inequalities has a 0-1 solution. There are now a few hundred NP-complete problems known, each having some reasonable practical or mathematical motivation [GaJ]. Since it is conjectured that $P \neq NP$, proving a problem NP-complete is viewed as evidence that the problem cannot be solved in deterministic polynomial time. We are content here just to sketch the proof that SAT, the set of satisfiable propositional formulas, is NP-complete. The reader is referred to [GaJ] for more information about NP-completeness.

THEOREM 4.3. SAT *is* NP-*complete.*

PROOF (sketch). It is easy to see that SAT $\in$ NP: given a propositional formula $F(X_1, \ldots, X_m)$, an NTM can nondeterministically choose an assignment of truth values to the variables $X_1, \ldots, X_m$, evaluate the truth value of F under the chosen assignment, and accept iff the answer is true.

To sketch the proof that NP $\leq_p$ SAT, we use definitions and notation as in the proof of Theorem 4.2. We first note that NP is the class of problems accepted by simple NTM's in polynomial time. Let M be a simple NTM with time and space bound $p(n)$. The proof of Theorem 4.2 does not work for NTM's because $c_{i,j}$ is not uniquely determined by the symbols of C_{i-1}. However, it is not difficult to see that there is a 6-ary relation R_M such that $C_0, C_1, \ldots, C_{p(n)}$ is a valid accepting computation of M on input x (with $n = |x|$) iff $C_0 = \$q_0x\#\#\#\cdots\#\$$, $C_{p(n)}$ contains the accepting state q_a, and

$$R_M(c_{i-1,j-1}, c_{i-1,j}, c_{i-1,j+1}, c_{i,j-1}, c_{i,j}, c_{i,j+1})$$

for all i and j with $1 \leq i \leq p(n)$ and $2 \leq j \leq p(n) + 2$. In other words, the validity of the computation can be checked by making "local checks" within the computation. Each local check, as well as the check that C_0 is the initial configuration and that $C_{p(n)}$ is an accepting configuration, can be expressed as a propositional formula involving the variables $X(i,j,\gamma)$. The conjunction of all these formulas is satisfiable iff M accepts x. $\square$

SAT remains NP-complete even when restricted to formulas in conjunctive normal form with at most three literals per conjunct [Col]. Similarly, the set of tautologous propositional formulas is co-NP-complete even when restricted to formulas in disjunctive normal form with at most three literals per disjunct.

4.4. PSPACE-*complete problems.* PSPACE-completeness is, in a certain technical sense, even stronger evidence of intractability than NP-completeness. If NP $\neq$ PSPACE as is conjectured, PSPACE-complete problems cannot be solved in polynomial time even nondeterministically. The first problem shown to be PSPACE-complete was the equivalence problem for Kleene regular expressions (Meyer and Stockmeyer [MeS]). As our main example, we choose the *quantified Boolean formula* problem (QBF) which was shown PSPACE-complete by Stockmeyer [StoM], [Sto2]. QBF is the set of propositional formulas $F(X_1,\ldots,X_m)$ such that

$$(\exists X_1)(\forall X_2)(\exists X_3)(\forall X_4)\cdots(Q_m X_m)[F(X_1,\ldots,X_m)]$$

where the quantifiers alternate (so that Q_m is $\exists$ ($\forall$) if m is odd (even)).

THEOREM 4.4. QBF *is* PSPACE-*complete.*

PROOF (sketch). (1) QBF $\in$ PSPACE. Given a formula $F(X_1,\ldots,X_m)$, a DTM determines whether $F \in$ QBF by the brute-force approach of trying both truth assignments to X_1, for each of these trying both truth assignments to X_2, and so on. This requires space to record a truth assignment to all the variables of F, which is certainly polynomial in the length of F.

(2) PSPACE $\leq_p$ QBF. We again use terminology as in the proofs of Theorems 4.2 and 4.3. PSPACE is the class of problems accepted by simple DTM's with a polynomial space bound, but no time bound other than the bound $2^{\text{poly}(n)}$ implied by (2.3c). Let M be a simple DTM with polynomial space bound $p(n)$. Since M can run for $2^{\text{poly}(n)}$ steps, we cannot associate a sequence of propositional variables with each configuration in a computation of M as was done in the proofs of Theorem 4.2 and 4.3. Instead, we write quantified formulas $F_k(V, V')$, where V and V' are sequences of propositional variables $V(j,\gamma)$ and $V'(j,\gamma)$, for $1 \leq j \leq p(n) + 3$ and $\gamma \in Q \cup \Gamma \cup \{\$\}$, which occur free in F_k and which encode configurations of M as described in the proof of Theorem 4.2. $F_k(V, V')$ expresses the fact that V and V' both encode configurations of M and that if M is started in the configuration encoded by V then M can reach the configuration encoded by V' by a computation involving 2^k steps. The formula F_0 checks that one configuration can reach another in one step of M, and this formula is written as in the proof of Theorem 4.3 as a conjunction of local checks within the two configurations. A first attempt to write F_k for $k > 0$ is

$$F_k(V, V') = (\exists W)[F_{k-1}(V, W) \wedge F_{k-1}(W, V')].$$

Here, W is another sequence of variables which encodes a configuration of M. The idea of checking that V can reach V' in 2^k steps by existentially choosing the "midpoint" W is due to Savitch [Sav]. This does not quite work here because the length of F_k grows exponentially in k, and ultimately we need F_k where k is some polynomial function of n. One more trick allows us to write F_k in terms of one copy of F_{k-1} so that the length of F_k grows only polynomially in k:

$$F_k(V, V') \\ = (\exists W)(\forall Y)(\forall Z)[((Y = V \wedge Z = W) \vee (Y = W \wedge Z = V')) \rightarrow F_{k-1}(Y, Z)].$$

Let $q(n)$ be a polynomial such that M accepts within time $2^{q(n)}$. We can use $F_{q(n)}$ to express the fact that the initial configuration of M on input x can reach an accepting configuration within $2^{q(n)}$ steps. $\square$

QBF remains PSPACE-complete even when restricted to formulas in conjunctive or disjunctive normal form with at most three literals per conjunct or disjunct [Sto2]. By noting that the reduction of PSPACE to QBF described above can be computed in logarithmic space and that the reduction maps an input x of length n to a formula of length $O(n^2 \log n)$ in the case that M is an NTM with a linear space bound (i.e., $p(n) = n$), the method of §3.2 can be used to show that QBF requires nondeterministic space n^δ i.o. for any fixed $\delta < \frac{1}{2}$. Theorem 4.4 can be used to show PSPACE-hardness of *any* first-order theory which has the ability to emulate QBF. One example from [Sto2] is the first-order theory of equality which is, in fact, PSPACE-complete. We shall see other examples of PSPACE-complete problems in §6.

§5. Other classes.

5.1. *Alternation and the polynomial-time hierarchy.* Alternating Turing machines (ATM's) were defined by Chandra, Kozen and Stockmeyer [ChKS]. One of the motivations for the definition was to generalize the concept of a nondeterministic computation. Another way to think of the definition of acceptance for NTM's is that a configuration C of an NTM M leads to acceptance iff there exists a configuration C', with $C \vdash_M C'$, such that C' leads to acceptance. The definition of alternation generalizes this to alternating quantifiers. A particular ATM is specified like an NTM as described in §2.2, but in addition a particular subset U of the states Q is specified. The states in U are called *universal states* and the states in $Q - U$ are called *existential states*. Configurations are called universal or existential according to the state of the configuration. Informally, a universal configuration C leads to acceptance if C' leads to acceptance for all C' such that $C \vdash_M C'$. More formally, an *accepting computation* of an ATM M on an input x is a rooted tree whose nodes are labelled by configurations of M such that the root is labelled by Init_x, all leaves are labelled by accepting configurations, if an internal (i.e., nonleaf) node is labelled by an existential configuration C then the node has one son which is labelled by some C' with $C \vdash_M C'$, and if an internal node is labelled by a universal configuration C and if $C_1, \ldots, C_d$ are the configurations reachable from C in one step of M then the node has d sons labelled $C_1, \ldots, C_d$. The *time* of a computation is the maximum length of a path from the root to some leaf. The *space* of a computation is the maximum number of work-tape squares visited in each of the configurations labelling nodes of the configuration. A particular root-to-leaf path has k *alternations* if the number of times the machine switches from an existential to a universal configuration, or vice versa, on the path is $k - 1$. The *alternations* of a computation is the maximum, over all root-to-leaf paths, of the alternations of the path. In complete analogy to the definitions of §2.2 for NTM's and DTM's, we can define the set of words $L(M)$ accepted by the ATM M, the notion of M accepting in time $T(n)$, space $S(n)$, or alternations $A(n)$, and the alternating complexity classes ATIME($T(n)$) and ASPACE($S(n)$) defined by ATM's which accept within time $T(n)$ or space $S(n)$, respectively. Alternating complexity classes are related to deterministic complexity classes by the following result from [ChKS] which shows that alternating time (space) corresponds to deterministic space (time).

THEOREM 5.1. (a) ATIME(poly($T(n)$)) = DSPACE(poly($T(n)$)).

(b) ASPACE($S(n)$) = DTIME($2^{O(S(n))}$).

These relationships have been useful in classifying the complexities of certain problems in logic and game theory, two areas where alternating quantifiers arise naturally. We shall see some examples in §6.

Of special interest are classes defined by ATM's which accept within polynomial time and within a constant number of alternations. Such classes give a polynomial time bounded analogue of Kleene's arithmetical hierarchy [Rog, Chapter 14], where P is the analogue of the recursive sets and NP is the analogue of the r.e. sets. (To help see the analogy, note that the definition of NP given by (1.1) defines the r.e. sets if the polynomial bound on the length of y is omitted.) The classes of the *polynomial-time hierarchy* are Σ_k^p and Π_k^p for $k \geq 0$. $\Sigma_0^p = \Pi_0^p = \mathrm{P}$. For $k > 0$, Σ_k^p (resp., Π_k^p) is the class of problems accepted by ATM's whose initial state is existential (resp., universal) and which accept simultaneously within time poly(n) and within k alternations. The original definition of the polynomial-time hierarchy, given by Meyer and Stockmeyer [MeS], more closely mimics the definition of the arithmetical hierarchy. Specifically, Σ_k^p can be defined equivalently as the class of problems accepted in polynomial time by nondeterministic oracle Turing machines with oracles in Σ_{k-1}^p. (An oracle machine can write words on a special oracle tape, and in one step find out whether or not the word written on the oracle tape belongs to the oracle set.) The definition in terms of oracle machines also allows us to define Δ_k^p to be the class of problems accepted in polynomial time by deterministic oracle machines with oracles in Σ_{k-1}^p. Trivial relationships are

$$\Sigma_1^p = \mathrm{NP}, \qquad \Pi_k^p = \text{co-}\Sigma_k^p,$$
$$\Sigma_k^p \cup \Pi_k^p \subseteq \Delta_{k+1}^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p,$$
$$\Sigma_k^p \subseteq \mathrm{PSPACE}$$

for all $k \geq 0$, although none of these inclusions are known to be proper. It is easy to show that if $\Sigma_k^p = \Sigma_{k+1}^p$ for some $k \geq 0$, then $\Sigma_k^p = \Sigma_j^p$ for all $j \geq k$ [Sto2]. In particular, if $\mathrm{P} \neq \Sigma_k^p$ for any $k \geq 1$, then $\mathrm{P} \neq \mathrm{NP}$.

One of the original hopes in defining the polynomial-time hierarchy was that it would be useful in classifying certain recursive problems, just as the arithmetical hierarchy has been useful in classifying certain nonrecursive problems. This hope has not been completely borne out, although a few results are known. Sagiv and Yannakakis [SaY] show that a certain problem in relational database theory is Σ_2^p-complete. Huynh [Huy] shows that the equivalence problem for context-free grammars with a one-letter terminal alphabet is Π_2^p-complete. Papadimitriou

[Pal] shows that, given a graph with an integer distance assigned to each edge, determining uniqueness of the optimal traveling salesman tour is Δ_2^p-complete. Jeroslow [Je] shows that a certain linear programming game is complete in various levels of the hierarchy, the level depending on the number of players. Other, more artificial, problems complete in various levels of the hierarchy are given in [Sto2] and [Wr].

5.2. *Probabilistic classes.* There are certain problems which are not known to be solvable in deterministic polynomial time but which can be solved in polynomial time by algorithms which can utilize random numbers in their computations and which may make errors with small probability. A simple example of this, as noted by Schwartz [Schw], is the problem of checking whether a polynomial expression is identically zero. Let $x_1, x_2, \ldots$ be a sequence of indeterminates. An indeterminate is a *polynomial expression* of degree 1. An integer is a *polynomial expression* of degree 0. If F and G are polynomial expressions then $(F + G)$ and $(F - G)$ are *polynomial expressions* of degree max(deg(F), deg(G)), and (FG) is a *polynomial expression* of degree deg(F) + deg(G). Let ZERO be the set of polynomial expressions which are identically zero. For example, $(x_1 + x_2)(x_1 - x_2) - x_1x_1 + x_2x_2$ is in ZERO. An obvious approach for solving this problem is to expand the given expression into a sum of monomials. This is not a polynomial time algorithm since the expansion could cause an exponential blow-up in the length of the expression. However, it is not hard to prove that if a polynomial expression $F(x_1, \ldots, x_m)$ is not identically zero and if $N \geq c \cdot \deg(F)$, then F has at most $c^{-1}N^m$ integral roots in $[1, N]^m$ [Scwh]. Thus, another approach would be to independently choose m random integers from the interval $[1, N]$ for $N = 2 \cdot \deg(F)$, evaluate F on the chosen random integers, and accept iff F evaluates to 0. For inputs F with $F \in$ ZERO, this procedure is always correct. For inputs with $F \notin$ ZERO, this procedure makes an error with probability at most 1/2. By repeating this procedure t times, using a new random assignment at each trial, the error probability decreases to at most 2^{-t}.

The concept of a resource bounded Turing machine which can make random choices was formalized by Gill [Gi]. The definition of a *probabilistic Turing machine* is actually identical to the definition of an NTM. The difference arises in the definition of acceptance. When a probabilistic Turing machine M is in a configuration C and there are d configurations C' such that $C \vdash_M C'$, we imagine that M generates a random integer between 1 and d to choose which of the d successors of C to enter next. For simplicity, let us restrict attention to Turing machines with the property that all computations, accepting or not, halt within $p(n)$ steps for some polynomial p. For each input x, there is a well-defined probability $\rho_M(x)$ that M accepts x.

A problem A is in the class R, called *random polynomial time*, if there is a probabilistic Turing machine M which always halts in polynomial time and a constant $\varepsilon > 0$ such that $\rho_M(x) \geq \varepsilon$ for all $x \in A$ and $\rho_M(x) = 0$ for all $x \notin A$. Note that M, when viewed as an NTM, accepts A and has the additional property that, for each accepted input, at least the fraction ε of the possible computations are accepting. It follows that

$$\mathrm{P} \subseteq \mathrm{R} \subseteq \mathrm{NP}.$$

We have noted above that the complement of ZERO belongs to R. Some generalizations of this result are given by Ibarra and Moran [IbM]. An interesting example of a problem which is known to belong to R, but whose membership in P is unresolved, is the set of binary representations of composite integers. The proof that the set of composite integers belongs to R is due to to Solovay and Strassen [SoS] and independently to Rabin [Rab5], who bases the algorithm on a method of Miller [Mi]. Berlekamp [Berl] and Rabin [Rab6] give probabilistic algorithms for other number-theoretic problems, such as factoring polynomials over finite fields, which are faster than the known deterministic algorithms.

Gill also defines the probabilistic complexity class BPP which allows errors on both accepted and rejected inputs. Precisely, A is in BPP if there is a polynomially time bounded probabilistic Turing machine M and a constant $\delta > 1/2$ such that $\rho_M(x) \geq \delta$ for all $x \in A$ and $\rho_M(x) \leq 1 - \delta$ for all $x \notin A$. It is known that

$$\mathrm{R} \subseteq \mathrm{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p.$$

The second inclusion is due to Sipser and Gács [Sip3] and is not obvious; a simpler proof is given by Lautemann [Lau]. From a practical point of view, showing a problem to be in R or BPP is almost as good as showing the problem to be in P, since by repeating the probabilistic algorithm t times, the error probability decreases exponentially in t. There are no problems known to be R-complete or BPP-complete. Gill [Gi] also defines probabilistic classes by requiring only $\rho_M(x) > \frac{1}{2}$ for $x \in A$ and $\rho_M(x) \leq \frac{1}{2}$ for $x \notin A$, so that the probabilities of accepting and rejecting need not remain bounded away from one another; such classes are more of mathematical than practical interest. Space bounded probabilistic classes are also considered in [Gi]. Further discussion of probabilistic complexity classes and related issues can be found in [Joh3].

5.3. *Parallel computation.* As the price of computer hardware continues to decrease, it becomes quite attractive to speed up the solution to problems by having many processors work on the problem in parallel. For this to succeed, it must be possible to partition the computation among the processors so that the various parts can be carried out more or less independently. This raises the theoretical question of determining which problems can be solved much faster in parallel than sequentially.

A very simple example of a problem which can be solved much faster in parallel than sequentially is the problem of recognizing the set of words x such that $x = uu$ for some $u \in \{0, 1\}^*$. Given an input x of even length n, and assuming that we have $n/2$ processors, the ith processor can compare the ith symbol of x with the $(n/2 + i)$th symbol of x for $1 \leq i \leq n/2$. All of these comparisons are done in one parallel step. In the next parallel step, the ith processor communicates the outcome of its comparison to the $(i - n/2)$th processor for $n/2 + 1 \leq i \leq n$. Then the ith processor communicates to the $(i - n/4)$th processor for $n/4 + 1 \leq i \leq n/2$, and so on. After $\log n$ of these communication steps, the information of whether or not all the comparisons resulted in equality has been fanned into the first processor. Thus, a problem which requires time n to be solved sequentially can be solved in time $O(\log n)$ in parallel. For some problems in P, it is not known whether they can be solved in parallel time $O(\log n)$, although we can do almost as well by showing that poly$(\log n)$ parallel time suffices. One example is the problem GAP discussed in §4.1

which can be solved in parallel time $O((\log n)^2)$ using $O(n^{3/2})$ processors (see, for example, Borodin [Boro]). For certain other problems in P, such as the problem CVP defined in §4.2, it is not known whether parallel time poly($\log n$) suffices. In the case of CVP, since a variable X_i depends on other variables X_j for $j < i$, there is apparently no way to partition the computation into many independent pieces. The following question is open.

(5.1) *Can every problem in* P *be solved in parallel time* poly($\log n$)?

Of course, before we can begin to answer this question, a precise definition of "parallel time" is needed. Several parallel computational models have been proposed. We do not attempt to reproduce a detailed definition of one of these models, but we will attempt to give the idea. Probably the most popular theoretical model is a parallel version of the random access machine described in §2.3. The PRAM, defined by Fortune and Wyllie [FoW], consists of a sequence of deterministic RAM processors, $R_1, R_2, \ldots, R_{P(n)}$, operating in parallel. The number $P(n)$ of processors can be a function of the length n of the input. In order for the RAM's to communicate among one another, we assume that there is a *common memory* consisting of registers c_1, c_2, $c_3, \ldots$, and there are instructions in the programs of the individual RAM's for writing into and reading from the common memory. Each RAM has the same program. However, initially i is written in one of the local memory registers of R_i, so that different RAM's can operate differently. Initially, the symbols of the input are written in the first n registers of common memory. The computation of a PRAM proceeds synchronously, that is, at each parallel step all RAM's execute the next step of their program in parallel. We assume that the program is such that two different RAM's never attempt to write into the same common memory register at the same parallel step. The PRAM accepts if R_1 accepts. For function computation, the output is in common memory when the computation halts. Two resource bounds of interest are the parallel time $T(n)$, that is, the number of parallel steps in a computation, and the number of processors $P(n)$. In this subsection, we permit parallel time bounds $T(n)$ to be as small as $\log n$. Another feature of the definition of the PRAM is that initially there is only one "active" processor, R_1, but a processor can activate another at any step. Thus, after t parallel steps, 2^t processors could be active but no more.

Investigation of a variety of parallel models has shown that if no explicit upper bound is placed on the number of processors, then the class of problems accepted by parallel machines in time poly($T(n)$) is precisely the class of problems accepted by DTM's in space poly($T(n)$). This was first proved by Pratt and Stockmeyer [PrS] for a parallel model called the vector machine. Fortune and Wyllie [FoW] prove this equivalence for PRAM's. If we view ATM's (see §5.1) as a parallel model, then Theorem 5.1(a) is another example of the equivalence. This has led to the formulation of a "parallel computation thesis" by Chandra and Stockmeyer [ChS] and Goldschlager [Go2] that parallel time is equivalent to Turing machine space, at least to within composition with a polynomial. Adopting this thesis, the question (5.1) is equivalent to a question we have seen before in §4.2, namely,

(5.2) *Is* P $\subseteq$ DSPACE(poly($\log n$))?

As a result, proving a problem to be P-complete is viewed as evidence that the problem cannot be solved in poly($\log n$) parallel time. We should point out that the parallel models for which the parallel computation thesis has been verified all have the property that if a parallel computation has t parallel steps then at most $2^{O(t)}$ processors can take part in the computation; thus, there is the implicit bound that $P(n)$ is $2^{O(T(n))}$. Blum [Blu3] points out that the parallel computation thesis can fail if many more than $2^{O(T(n))}$ processors can be employed.

Parallel time classes can be refined by also placing an upper bound on the number of processors, and such upper bounds are necessary if parallel algorithms are to have any practical significance. For example, it was shown in [PrS] that any context-free language can be accepted in poly($\log n$) parallel time, but the number of processors was not bounded above by a polynomial in n. Pippenger [Pi] suggested that an interesting complexity class to study would be the class of problems solvable by parallel machines in poly($\log n$) time using poly(n) processors; this class has come to be known as NC (for "Nick's class"). Pippenger also provided a characterization of NC in terms of Turing machines: NC is the class of problems solvable by DTM's which accept in polynomial time and whose work-tape heads make at most poly($\log n$) reversals. Another characterization of NC, by Ruzzo [Ruz], is that NC is the class of problems accepted by ATM's which accept simultaneously within space $O(\log n)$ and within time poly($\log n$). Ruzzo [Ruz] also shows that any context-free language is in NC. One of the first problems shown to be in NC (even before the class was formally defined) was the problem of inverting and computing the determinant of a matrix (Csanky [Cs]). Placing problems in NC, including problems which are not obviously amenable to significant parallelization, is currently a very active area. The reader interested in finding out more about the theory of parallel computation would do well to start with two survey articles of Cook, [Co4] and [Co5].

5.4. #P. Valiant [Va1] has introduced the class #P to capture the complexity of various counting problems. #P is a class of functions rather than decision problems. A function $g: \Sigma^* \to \mathrm{N}$ is in #P if there is an NTM M which always halts in polynomial time such that, for all $x \in \Sigma^*$, M has $g(x)$ accepting computations on input x. For example, the function which maps a propositional formula $F(X_1, \ldots, X_m)$ to the number of satisfying truth assignments of F is easily seen to be in #P. Polynomial time Turing reducibility is the appropriate reducibility to use in defining #P-completeness. A function g is *#P-complete* if $g \in$ #P and, for all $h \in$ #P, h can be computed in polynomial time by a deterministic oracle Turing machine with an oracle for g; the oracle machine can write words on a special oracle tape and in one step obtain the binary representation of $g(w)$, where w is written on the oracle tape.

It is clear that any function in #P can be computed in polynomial space, but the relation between #P and the polynomial-time hierarchy is not known. #P-completeness is also evidence of computational intractability, since if any #P-complete function can be computed in polynomial time then the number of accepting computations of an arbitrary polynomially time bounded NTM can be computed in deterministic polynomial time; of course, this implies that $\mathrm{P} = \mathrm{NP}$.

Not surprisingly, #P-complete problems include counting versions of many NP-complete problems, such as counting the number of satisfying assignments to a given propositional formula and counting the number of Hamiltonian cycles in a given graph. A more surprising result, due to Valiant [Va1], is that the problem of computing the permanent of a matrix with integer entries is #P-complete, and the problem remains #P-complete even for zero-one entries. The permanent of an $m \times m$ matrix $\{a_{i,j}\}_{i,j=1}^{m}$ is the sum, over all permutations $\pi: \{1,\dots,m\} \to \{1,\dots,m\}$, of the product

$$t_\pi = a_{1,\pi(1)} a_{2,\pi(2)} \cdots a_{m,\pi(m)}.$$

The #P-completeness of the permanent is interesting because the existential version of the problem (Does there exist a permutation π such that $t_\pi \neq 0$?) can be solved in deterministic polynomial time by a simple reduction to the problem of checking whether a bipartite graph has a perfect matching, which can be decided in polynomial time [PaS]. Other #P-complete problems in [Va2] include the problem of computing the probability that a graph is connected when each edge of the graph is present with a given probability.

5.5. $\mathrm{D^p}$. Papadimitriou and Yannakakis [PaY] consider the class $\mathrm{D^p}$ defined as

$$\mathrm{D^p} = \{A \cap B \mid A \in \mathrm{NP} \text{ and } B \in \text{co-NP}\}.$$

Obviously,

$$\mathrm{NP} \cup \text{co-NP} \subseteq \mathrm{D^p} \subseteq \Delta_2^{\mathrm{p}}.$$

$\mathrm{D^p}$ has been useful for classifying various extremal graph problems such as recognizing the set of undirected graphs G such that G does not have a Hamiltonian cycle but adding any new edge to G produces a graph with a Hamiltonian cycle, and recognizing the set of pairs (G, k) such that G is an undirected graph, k is an integer, and the largest complete subgraph of G has exactly k vertices. Both of these problems are $\mathrm{D^p}$-complete (see [PaY] and [PaW]). (In contrast, the set of (G, k) such that G has a complete subgraph with *at least* k vertices is NP-complete.) One of the original motivations for $\mathrm{D^p}$ was to classify the problem of recognizing the set of linear inequalities which describe facets of the traveling salesman polytope. This problem has now been shown $\mathrm{D^p}$-complete by Papadimitriou and Wolfe [PaW]. Valiant and Vazirani [VaV] show that the *unique satisfiability problem* (given a propositional formula F, does it have exactly one satisfying assignment?) is $\mathrm{D^p}$-complete with respect to *randomizing* polynomial-time reducibility (i.e., the reducibility function is computed by a probabilistic Turing machine which can make errors with small probability). Further discussion of $\mathrm{D^p}$ and the complexity of problems having unique solutions can be found in [Joh4].

§6. Complexities of some specific problems. In this section we survey several classification results. The problems are grouped by the area (logic, games, algebra, and formal languages) from which they are drawn. In each subsection it is assumed that the reader is already somewhat familiar with the area being discussed. This section is by no means an exhaustive list of known results. In particular, [GaJ] lists many other NP-complete problems as well as a few PSPACE-complete ones.

6.1. *Logic.* The complexities of nearly all the classical decidable theories have been classified in terms of time or space requirements, although gaps between known upper and lower bounds remain in a few cases. Since many of the relevant time and space bounds involve iterated exponentiation, we define notation for this.

Define the function $g(k,n)$ for n, $k \in \mathbf{N}$ by $g(0,n) = n$ and $g(k+1,n) = 2^{g(k,n)}$ for $k \geq 0$.

We first consider decision problems which cannot be accepted within space $g(k,n)$ for any fixed k; such problems are termed *nonelementary*. The first example of a decidable but nonelementary logical decision problem, the weak monadic second-order theory of one successor, was due to Meyer [Me1]. In this theory, formulas involve individual variables, set variables, primitive relations "$y = x + 1$" and "$x \in S$" where x and y denote individual variables and S denotes a set variable, quantifiers $\exists$ and $\forall$ for both individual variables and set variables, and the usual logical connectives. Let WS1S denote the set of sentences (i.e., closed formulas) which are true when individual variables range over $\mathbf{N}$ and set variables range over finite subsets of $\mathbf{N}$.

THEOREM (MEYER [Me1]). *There is a constant $c > 0$ such that*

$$\text{DSPACE}(g(\ulcorner c \cdot \log n \urcorner, 1)) \leq_{\text{p}} \text{WS1S},$$

and WS1S *requires space* $g(\ulcorner c \cdot \log n \urcorner, 1)$ *i.o.*

M. Rabin has observed that "$\log n$" can be replaced by "n" in this theorem. This improved lower bound of $g(\lceil cn \rceil, 1)$ is closer to the known upper bound of space $g(n,1)$ which is implicit in the decision procedures of Büchi [Bu] and Elgot [Elg]. Meyer and Stockmeyer [Sto1] and Robertson [Ro] have shown that if "$x < y$" is also included as a primitive relation, then just one set quantifier suffices to make the decision problem nonelementary. Meyer's theorem implies that the more powerful theory SnS, proved decidable by Rabin [Rab3], is also nonelementary. Thus, the popular method of proving decidability by reduction to SnS, although a powerful method for proving decidability, might give a poor upper bound on computational complexity.

Provably nonelementary first-order theories include the theory of linear order (Meyer [Me2], [Sto1]) and the theory of any nonempty family of pairing functions (Rackoff [Rac1], [FeR2]). Other examples are given in the survey article by Meyer [Me2].

We now turn to elementary theories, i.e., those which can be decided in space $g(k_0, n)$ for some fixed constant k_0. Table 1 summarizes known upper and lower bounds for several first-order theories. In each row, $\mathscr{C}_{\text{lower}}$ is reducible to the decision problem for the theory via the indicated reducibility, and the decision problem belongs to $\mathscr{C}_{\text{upper}}$. In each case it is straightforward to obtain an explicit lower bound using the method of §3.2. Row 1 is due to Stockmeyer [Sto2], row 2 is due to Ferrante [Fe], [FeR2], the lower bounds in rows 3–7 are by Fischer and Rabin [FiR], the upper bounds in rows 3 and 5 are by Ferrante and Rackoff [FeR1], the upper bound in row 4 is by Rackoff [Rac1], [Rac3], [FeR2], the upper bound in row 6 is due to Ben-Or, Kozen and Reif [BeKR], and the upper bound in row 7 is by Lo [Lo] who also gives upper bounds for other decision problems concerning abelian groups.

Berman [Ber2] has given a more precise classification for Th($\mathbf{R}$, +) by showing that it is complete in the class of problems accepted by ATM's (see §5.1) which accept simultaneously within time $2^{O(n)}$ and within $O(n)$ alternations. In a technical sense, this class lies between NTIME($2^{O(n)}$) and DSPACE($2^{O(n)}$), since if we fix the time bound at $2^{O(n)}$, then NTIME($2^{O(n)}$) corresponds to 1 alternation and DSPACE($2^{O(n)}$) corresponds to $2^{O(n)}$ alternations (cf. Theorem 5.1(a)). (However it is not known whether the class lies properly between; it is not even known that NTIME($2^{O(n)}$) $\neq$ DSPACE($2^{O(n)}$).) Similar classifications using ATM's which are both time and alternation bounded are known for Th(**N**, +) (Berman [Ber2]) and the theory of Boolean algebras (Kozen [Ko2]). Further evidence that Th(**R**, +) does not fit neatly into a time or space class is given by Bruss and Meyer [BrM], who show that there is a constant $c > 0$ such that Th(**R**, +) either requires nondeterministic space 2^{cn} i.o. or requires nondeterministic time 2^{cn^2} i.o. It is an interesting open question to give a more precise classification of Th(**R**, +, ·) than the one given in row 6 of Table 1.

Theory	$\mathscr{C}_{\text{lower}}$	$\mathscr{C}_{\text{upper}}$	reducibility
1. equality	NSPACE($\sqrt{n}$)	DSPACE($n \log n$)	$\leq_{\text{log-lin}}$
2. **N** with successor	NSPACE(n)	DSPACE(n^2)	$\leq_{\text{log-lin}}$
3. **N** with addition	NTIME($g(2,n)$)	DSPACE($g(2, O(n))$)	$\leq_{\text{p-lin}}$
4. **N** with multiplication	NTIME($g(3,n)$)	DSPACE($g(3, O(n))$)	$\leq_{\text{p-lin}}$
5. **R** with addition	NTIME($g(1,n)$)	DSPACE($g(1, O(n))$)	$\leq_{\text{p-lin}}$
6. **R** with addition and multiplication	NTIME($g(1,n)$)	DSPACE($g(1, O(n^2))$)	$\leq_{\text{p-lin}}$
7. finite abelian groups	NTIME($g(2,n)$)	DSPACE($g(2, O(n))$)	$\leq_{\text{p-lin}}$

TABLE 1. Complexities of some first-order theories

Many of the lower bounds on the complexities of logical decision problems (including all those in Table 1) rely on the ability to write formulas with many alternations of quantifiers. The effect of the number of alternations on the complexity of Th(**N**, +) is studied by Reddy and Loveland [ReL] (upper bounds) and Fürer [Fu2] (lower bounds). Sistla, Vardi and Wolper [SiVW] show that for sentences of WS1S in prenex normal form, each additional alternation of second-order quantifiers causes an exponential increase in the space complexity of the decision problem; moreover, the same result holds for the (full) monadic theory S1S, allowing quantification over infinite sets. This improves a previous result of Robertson [Ro].

Meyer and Rackoff [Rac2] show that the satisfiability problem for the monadic predicate calculus requires nondeterministic time $c^{n/\log n}$ for some constant $c > 1$, and Lewis [Lew] gives a similar upper bound of NTIME($d^{n/\log n}$) for some constant d. Lewis [Lew] also gives upper and lower bounds for other decidable subcases of the predicate calculus obtained by restricting the form of the quantifier prefix. For formulas in Schönfinkel-Bernays form, Plaisted [Pl] considers the effect of also restricting the form of the matrix, for example, to a conjunction of Horn clauses.

Compton and Henson [CoH] describe a method for proving lower bounds on the computational complexity of decidable theories, which is analogous to known machinery for proving undecidability of logical theories (see [Rab2]). In particular, the method gives new proofs of the nonelementary complexity of the first-order theory of one unary function and the first-order theory of a linear order.

There has also been much work on the complexity of modal logics. Ladner [Lad3] shows that the satisfiability problems for the classical modal logics T and S4 are PSPACE-complete and that S5 is NP-complete in the case of one modality, i.e., one "knower". For the case of many modalities, Halpern and Moses [HalM] show that T and S4 remain PSPACE-complete and that S5 becomes PSPACE-complete. Halpern and Moses [HalM] also consider the effect of adding an operator for "common knowledge" to these logics, and find that all three become DTIME($2^{O(n)}$)-complete in the case of m knowers for any $m \geq 2$. This line of work has been continued by Halpern and Vardi [HalV], who classify the complexities of various logics for reasoning about knowledge and time together. Propositional modal logics are of considerable current interest in computer science as languages for expressing the behavior of programs. Pnueli [Pn] has suggested this role for linear temporal logic. Sistla and Clarke [SisC] and Halpern and Reif [HalR] show that linear temporal logic is PSPACE-complete. Several new modal logics have also been defined for this purpose. One of the more important is *propositional dynamic logic* (PDL) which was defined by Fischer and Ladner [FiL] as a propositional version of a first-order modal logic proposed earlier by Pratt [Pr1]. Formulas in PDL have primitive symbols α, β, etc. which stand for programs. Semantics is in terms of Kripke structures, and each primitive program is interpreted as a binary relation on Kripke states (in general, a nonsymmetric nontransitive relation). New programs can be formed using combining operators such as $\alpha; \beta$ (run program α, then run program β), $\alpha \cup \beta$ (nondeterministically run either α or β), and α^* (run α zero or more times); the interpretation of the new program is obtained by relational composition, union, and transitive closure, respectively. There are modal operators $[\gamma]$ and $\langle\gamma\rangle$ for each program γ. For example, $[\gamma]\phi$ is true in state s iff ϕ is true in all states t such that (s, t) is in the relation γ. Fischer and Ladner [FiL] show that PDL is DTIME($2^{O(n)}$)-hard and Pratt [Pr2] shows that PDL is in DTIME($2^{O(n)}$). Various extensions of PDL have been defined to capture the ongoing behavior of programs by adding temporal operators which can, for example, express the fact that a formula holds infinitely often during every infinite execution of a program. A recent paper of Vardi and Stockmeyer [VarS] gives results on the complexities of some of these temporal logics and contains references to earlier work.

6.2. *Games.* In this section we consider decision problems of the following form: Given a position in some two-player game, decide whether the player moving first has a "forced win", that is, a strategy which leads to victory no matter which strategy the second player employs. For many common games, including chess and go and their generalization to boards of arbitrary size, this question is clearly decidable simply by examining the full game tree rooted at the given starting position. But since the size of the game tree could grow exponentially in the size of the initial position, this is not a polynomial time algorithm.

We have seen one example of a game already in the problem QBF defined in §4.4. To view QBF as a game, imagine that the first (second) player controls the odd (even) numbered variables, and the players alternately fix the truth values of these variables in order $X_1, X_2, \ldots, X_m$. After all the variables have been set, the first player wins iff $F(X_1, \ldots, X_m)$ is true. Obviously, $F \in$ QBF iff the first player has a winning strategy, so this is an example of a PSPACE-complete game. Variations of this game are proved PSPACE-complete by Schaefer [Scha]. Reisch [Reis] shows that the game of hex is PSPACE-complete when generalized to boards of arbitrary size and to arbitrary starting positions. The games which have been proved PSPACE-complete all have the property that when the game is played starting from a position of size n, the game is certain to end after at most poly(n) moves. It is not hard to see that the acceptance condition for an ATM M (§5.1) can be defined as a game between two players, the existential player and the universal player, in such a way that M accepts x iff the existential player has a winning strategy starting from the initial configuration on input x. Since alternating polynomial time equals PSPACE (Theorem 5.1(a)), it should not be surprising that polynomially time bounded games are PSPACE-complete.

However, some games such as chess and go are not polynomially time bounded when generalized to large boards, since the game could conceivably cycle through most of the exponentially many positions before ending. But since the board does not grow in size as the game is played, these games are O(n) space bounded. The equivalence ASPACE(O(n)) = DTIME($2^{O(n)}$) (Theorem 5.1(b)) suggests that these games are DTIME($2^{O(n)}$)-complete. Stockmeyer and Chandra [StoC] first proved games to be DTIME($2^{O(n)}$)-complete. These were artificial games played on propositional formulas. These artificial games were later used to show that generalizations of natural games are exponential-time-complete. Fraenkel and Lichtenstein [FrL] show that chess is DTIME($2^{O(n)}$)-complete. Robson [Rob1], [Rob2] shows that go and checkers are DTIME($2^{O(n)}$)-complete. In all three cases, the games are generalized to large boards in a natural way. For example, in the case of chess, an input is an m-by-m chessboard for some m together with a placement of pieces on some squares of the board (the placement need not correspond to the usual starting position of a game of chess). Each player has only one king, but the other pieces can occur any number of times.

Papadimitriou [Pa2] and Babai [Ba] study the complexity of games where one of the players chooses his moves randomly. Papadimitriou calls these "games against nature"; the decision problem here is whether the nonrandom player has a strategy which leads to victory with probability greater than 1/2. In Babai's case, the probabilities of winning or losing must remain bounded away from one another, say, greater than 2/3 or less than 1/3, respectively.

The games discussed above are all games of perfect information; both players have complete knowledge of the position of the game at all times. Reif [Reif] has considered games of imperfect information where some or all of the position is hidden from one of the players. He defines new types of ATM's to model games of imperfect information and characterizes their complexity classes in terms of deterministic complexity classes. Ladner and Norman [LaN] define automata to model solitaire games where one player (the "dealer") makes a number of moves at the beginning of the game, but after the other player enters the game the dealer must respond deterministically; solitaire games of both perfect and imperfect information are considered. A recent direction is to consider the effect of partial information on the random games of Papadimitriou and Babai (see, for example, Condon and Ladner [CoL], Goldwasser and Sipser [GoSi]).

Other results on the complexity of games are summarized in [Joh1].

6.3. *Algebraic word problems.* Let Θ be a finite alphabet of symbols. A *semi-Thue system* over Θ is a finite set $\mathscr{P}$ of productions of the form $l \to r$, where $l, r \in \Theta^*$. The word α *derives* the word β in one step, written $\alpha \to \beta$, if $\alpha = \mu l v$ and $\beta = \mu r v$ for some $l \to r$ in $\mathscr{P}$. Let $\overset{*}{\to}$ denote the transitive closure of $\to$. The *reachability set* R($\alpha, \mathscr{P}$) is the set of words β such that $\alpha \overset{*}{\to} \beta$. A semi-Thue system $\mathscr{P}$ is *commutative* if $ab \to ba$

is in $\mathscr{P}$ for all $a, b \in \Theta$ (when it is clear that a semi-Thue system is commutative, these productions do not appear explicitly in $\mathscr{P}$). A *Thue system* is a semi-Thue system such that, for all l and r, $(l \to r) \in \mathscr{P}$ iff $(r \to l) \in \mathscr{P}$. The *uniform word problem for commutative semigroups* (CSG) is the set of all $(\alpha, \beta, \mathscr{P})$ such that $\mathscr{P}$ is a commutative Thue system and $\beta \in \mathrm{R}(\alpha, \mathscr{P})$. Mayr and Meyer [MaM2] show that CSG is DSPACE($2^{O(n)}$)-complete. They also show that CSG is efficiently reducible to the *polynomial ideal word problem* (PI) defined as the set of all tuples of polynomials $(p_0, p_1, \ldots, p_m) \in \mathbf{Q}[X]^{m+1}$, where X is a finite set of indeterminates, such that p_0 is in the ideal of $\mathbf{Q}[X]$ generated by $p_1, \ldots, p_m$. Therefore, PI is DSPACE($2^{O(n)}$)-hard. It is also noted in [MaM2] that work of G. Hermann implies that PI is elementary recursive.

Another result should be mentioned here since it holds the current record for the size of a lower bound established for a "natural" decidable problem. Notational variants of commutative semi-Thue systems, called *vector replacement systems* and *Petri nets*, have been used in computer science to model the behavior of asynchronous systems. Consider the *finite equality problem* (FEP) defined as the set of $(\alpha, \mathscr{P}, \alpha', \mathscr{P}')$ such that $\mathscr{P}$ and $\mathscr{P}'$ are commutative semi-Thue systems, $\mathrm{R}(\alpha, \mathscr{P})$ and $\mathrm{R}(\alpha', \mathscr{P}')$ are finite, and $\mathrm{R}(\alpha, \mathscr{P}) = \mathrm{R}(\alpha', \mathscr{P}')$. Decidability of FEP follows from an algorithm of Karp and Miller [KaM] which checks whether a reachability set is finite and generates it if it is finite. Mayr and Meyer [MaM1] show that FEP is not primitive recursive. (Without the finiteness condition on the reachability sets, the equality problem for commutative semi-Thue systems is undecidable, as shown by Rabin and Hack [Hac].)

Kozen [Ko1] shows that the uniform word problem for finitely presented nonassociative algebras in P-complete.

6.4. *Formal language theory.* Many of the early classification results concerned problems in formal language theory. We have noted in §4.4 that the first problem proved PSPACE-complete was the equivalence problem for Kleene regular expressions containing only the operations of union, concatenation and Kleene star [MeS]. Other results have been obtained for other combinations of operations. For example, Stockmeyer [Sto1] shows that the equivalence problem for *star-free expressions*, which contain only union, concatenation and complementation, is nonelementary. Fürer [Fu1] improves the lower bound of [Sto1] by showing that the equivalence problem for star-free expressions requires space $g(\ulcorner cn/(\log^* n)^2 \urcorner, 1)$ i.o., where $\log^* n$ is defined as the smallest j such that $g(j, 1) \geq n$. Hunt and Rosenkrantz [HuR1], [HuR2] give various complexity-theoretic analogues of Rice's theorem [Rog, §2.1], which show that other decision problems concerning regular expressions are at least as hard as the equivalence problem. Although the equivalence problem for context-free grammars is undecidable, it is decidable in special cases such as for grammars generating finite languages; Hunt, Rosenkrantz and Szymanski [HuRS] show that this special case requires nondeterministic exponential time i.o.

§7. Related issues

7.1. *Relativization.* For a complexity class $\mathscr{C}$ and a set A of words, let $\mathscr{C}^A$ denote the same complexity class defined in terms of oracle Turing machines with oracle A rather than in terms of ordinary Turing machines. For example, NP^A is the class of problems accepted in polynomial time by nondeterministic oracle machines with oracle A. Baker, Gill and Solovay [BaGS] show that there are recursive sets A and B such that $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$. One conclusion to be drawn from this result is that any proof of either P = NP or P ≠ NP cannot generalize to an arbitrary oracle, so in trying to settle the P vs. NP question we can rule out proof methods which relativize. The construction of an oracle B such that $\mathrm{P}^B \neq \mathrm{NP}^B$ was shown independently by Dekhtiar [Dek]. There are many other results showing that various complexity classes can be made equal or unequal relative to an oracle. Recently Yao [Ya], using a line of attack suggested by Furst, Saxe and Sipser [FuSS] and Sipser [Sip2], has shown that there is an oracle relative to which all classes of the polynomial-time hierarchy (§5.1) are distinct, thus settling a question which had remained open for over ten years. Yao's proof has been simplified by Hastad [Has]. Rackoff [Rac4] exhibits oracles D and E for which $\mathrm{P}^D \neq \mathrm{R}^D$ and $\mathrm{P}^E = \mathrm{R}^E$, where R is random polynomial time (§5.2). Bennett and Gill [BeG] show that for a *random* oracle A, $\mathrm{P}^A \neq \mathrm{NP}^A$ with probability 1 and $\mathrm{P}^A = \mathrm{R}^A$ with probability 1. Other open questions, besides questions about equality of complexity classes, have been studied in the relativized case. For example, Sipser [Sip1] shows that there are oracles A and B such that R^A and $\mathrm{NP}^B \cap \text{co-NP}^B$ do not have complete problems with respect to $\leq_{\mathrm{p}}$.

7.2. *Structural issues.* There has been a considerable amount of work on the structure of complete problems. We mention a few examples.

Define $A \equiv_{\mathrm{p}} B$ if $A \leq_{\mathrm{p}} B$ and $B \leq_{\mathrm{p}} A$. Each equivalence class under $\equiv_{\mathrm{p}}$ is called a *p-degree*. For example, the class of NP-complete problems is a p-degree. Ladner [Lad1] shows that if P ≠ NP then the ordering of the p-degrees within NP is dense. In particular, there is a problem in NP − P which is not NP-complete. This is significant since there are natural problems in NP which are neither known to be in P nor known to be NP-complete; one interesting example is the problem of checking whether two given undirected graphs are isomorphic. A recent paper of Ambos-Spies [Amb] gives results on the structure of p-degrees and references to other work on this subject.

In recursive function theory it is known that all r.e.-complete sets under many-one reducibility are recursively isomorphic. In analogy, Berman and Hartmanis [BerH] conjecture that all NP-complete problems are p-isomorphic. Sets $A \subseteq \Sigma^*$ and $B \subseteq \Delta^*$ are *p-isomorphic* if there is a bijection $f\colon \Sigma^* \to \Delta^*$ such that f and f^{-1} are computable in polynomial time, $A \leq_{\mathrm{p}} B$ via f and $B \leq_{\mathrm{p}} A$ via f^{-1}. This conjecture implies P ≠ NP, since if P = NP then the finite sets $\{0\}$ and $\{0, 1\}$ are NP-complete but they are obviously not p-isomorphic. Nevertheless, machinery is developed in [BerH] which can be applied to show that many of the known NP-complete problems are p-isomorphic. Mahaney [Mah1] shows that, within the class of NP-complete problems, the number of p-isomorphism classes is either one or infinite.

A question raised in [BerH] is whether there are any sparse NP-complete problems. A set A of words is *sparse* if there is a polynomial $p(n)$ such that, for all n, A contains at most $p(n)$ words of length n. The existence of a sparse NP-complete problem would provide a counterexample to the conjecture of Berman and Hartmanis, since a nonsparse NP-complete problem such as SAT cannot be p-isomorphic to a sparse problem. The question remained open a few years but was finally settled by Mahaney [Mah2], who showed that if $P \neq NP$ then there are no sparse NP-complete problems. The history of the solution is described by Hartmanis and Mahaney [HarM].

Lynch [Ly] shows that for every problem A which is not in P there is an infinite recursive set X such that, given any DTM M which accepts A and any polynomial $p(n)$, there is an integer n_0 such that M runs for more than $p(|x|)$ steps on input x for all $x \in X$ with $|x| \geq n_0$. We can think of X as a "hard-core" for A, since it is a set of inputs which are hard for all acceptors of A. General hard-core theorems for other complexity classes are proved by Even, Selman and Yacobi [EvSY].

7.3. *Languages which capture complexity classes.* So far, logic has entered the discussion mainly as a source of decision problems whose complexities have been successfully classified. Logic can also be used to give alternate definitions of complexity classes, and in some cases questions about computational complexity can be rephrased as questions about logic. One of the first examples of this was work of Jones and Selman [JoS] and Fagin [Fa] on spectra. Given a first-order sentence ϕ with equality, the *spectrum* of ϕ is the set of cardinalities of finite models of ϕ. Spectra were first considered by Scholz [Scho], who raised the question of characterizing spectra. Jones and Selman [JoS] show that a set S of nonnegative integers is a spectrum of some first-order sentence iff the set of binary representations of integers in S belongs to $\mathrm{NTIME}(2^{O(n)})$. Similarly, S is a spectrum iff the set of unary representations of integers in S belongs to NP (the unary representation of the nonnegative integer z is the word $111 \cdots 1$ (z times)). It follows that the following two questions are equivalent:

(1) Is the complement of every spectrum also a spectrum?

(2) Does NP = co-NP when both classes are restricted to subsets of $\{1\}^*$?

Question (1) was posed by Asser [As]. In particular, if Asser's question has a negative answer then NP $\neq$ co-NP, which in turn implies that $P \neq NP$.

Fagin [Fa] characterizes NP in terms of generalized second-order spectra. Consider a second-order formula $\phi(R_1, \ldots, R_m)$ where some predicates $R_1, \ldots, R_m$ occur free in ϕ and where all second-order quantifiers are existential. The *generalized spectrum* of ϕ is the set of finite models of ϕ. Fagin shows that a set of finite models, suitably encoded as a set of words, is a generalized spectrum of some ϕ iff it belongs to NP. For example, consider the set of undirected 3-colorable graphs (an NP-complete problem). A graph can be represented by its edge relation E. It is straightforward to write a formula containing three existentially quantified unary predictates R, B and G (the three colors) and the free binary predicate E which states that each vertex is colored with exactly one color (i.e., $\forall u$ exactly one of $R(u)$, $B(u)$ or $G(u)$ is true) and that if $E(u,v)$ then u and v are colored differently. Given Fagin's characterization of NP, it is easy to see that the classes of the polynomial-time hierarchy can be similarly characterized by allowing a bounded number of alternations of second-order quantifiers in ϕ. Börger [Borg] gives a historical discussion of the connection between complexity theory and spectra.

Immerman [Im1] and Vardi [Var] characterize P in terms of fixed point logic over linearly ordered finite domains. Sentences in this logic are written using $<$, first-order quantifiers, and the least fixed point operator (LFP). The least fixed point operator has been extensively studied in mathematical logic (see [Mos]), although

Chandra and Harel [ChH] initiated the study of its expressive power in the context of finite structures. It is defined as follows. Given two k-ary relations R and R' defined on some domain U, write $R \subseteq R'$ if $R(u_1,\ldots,u_k) \to R'(u_1,\ldots,u_k)$ for all $u_1,\ldots,u_k \in U$. Let $\phi(u_1,\ldots,u_k,R)$ be a first-order formula, where R is a k-ary relation symbol and where $u_1,\ldots,u_k$ and R occur free in ϕ. For each interpretation of R, ϕ defines a k-ary relation which we denote $\phi(R)$. We say that ϕ is *monotone* if $R \subseteq R'$ implies $\phi(R) \subseteq \phi(R')$. For a monotone ϕ define

$$\mathrm{LFP}(\phi) = \min\{R \mid \phi(R) = R\}.$$

(For a monotone ϕ, it is known that LFP(ϕ) exists and can be computed in deterministic time polynomial in the cardinality of the domain.) Unaware of the results concerning LFP, Livchak [Liv] independently defined another extension of first-order logic which, together with order, captures P. It is an open question whether there is a natural extention of first-order logic which captures P without the use of order (without order, first-order logic with LFP cannot express "the cardinality of the domain is even" [ChH]). Immerman [Im2] characterizes other complexity classes in terms of logical languages and also summarizes previous work. Two articles of Gurevich, [Gu2] and [Gu3], contain more discussion about the subject of capturing P and the connections between logic and theoretical computer science in general.

Gurevich [Gu1] shows that if primitive recursive definitions are interpreted over finite domains instead of over $\mathbf{N}$, then the class of functions which can be defined is precisely the class of functions computable in deterministic space $\log n$. For recursive definitions over finite domains, precisely the class of polynomial time computable functions can be defined.

7.4. *Complexity of finite problems.* An objection both to undecidability results and to lower bounds such as Theorem 1.1 is that in the real world one is not interested in solving all instances of a problem but only a finite collection of them. In the case of Th($\mathbf{R}$, +), for example, one might be uninterested in all sentences of length greater than 1000. Thus, the value of c in the lower bound c^n and the minimum length of sentences for which the lower bound takes effect become germane. Estimates of these values are implicit in the proofs of lower bounds obtained as in §3.2. However, in the literature on lower bounds these values are usually not estimated, since this would be tedious and the estimates would not be very useful in practice. However, one example of a finite decision problem has been studied in detail. Consider sentences of WS1S (see §6.1) where order relations such as "$x < y$" and "$x \geq y$" are allowed as primitives, and integer constants can be written in decimal notation. Let $\mathscr{P}$ denote the set of sentences of length 616 or less. It turns out that the alphabet used in writing sentences has 63 symbols, so each symbol can be uniquely encoded as a binary word of length 6. In this way, each sentence in $\mathscr{P}$ is encoded as a binary word of length $6 \cdot 616 = 3696$. Turing machine time is not sufficient to measure the complexity of finite decision problems, since any finite problem can be decided in linear time by building a table of all the answers into the finite state control of the Turing machine. Thus, for assessing the complexity of finite problems, account must be taken of the size of the device performing an algorithm as well as the time required by the algorithm. One quite general way to do this is to measure the number of logical operations or "gates", *and*, *or*, *exclusive-or*, etc., in an acyclic logical network which is capable of solving the finite decision problem. In the case of deciding WS1S on sentences of length 616 or less, the network would have 3696 input lines and a single output line which should carry the value 1 iff the values on the input lines are the code of a true sentence of length at most 616. Formally, the network can be defined as a sequence of equations as in §4.2, except that the 3696 variables corresponding to input lines do not appear as the left side of any equation, and X_m is the output line. The number of gates is the number of equations of the form $X_i = X_j \ @ \ X_k$.

THEOREM (MEYER AND STOCKMEYER [Sto1]). *Any network which decides truth of sentences of length* 616 *in* WS1S *contains at least* 10^{123} *gates.*

Even if gates were the size of a proton and were connected by infinitely thin wires, the network would densely fill the known universe.

Having brought up the subject of logical networks, we should mention that interest in the complexity of problems as measured by the size of logical networks is much wider than the example above. For each (infinite) problem A, one can define the function $C_A(n)$, called the *combinational complexity* of A, which maps n to the minimum number of gates in a network which decides membership in A restricted to words of length n. It is easy to see that if $A \in \mathrm{DTIME}(T(n))$ then $C_A(n)$ is at most polynomial in $T(n)$; Pippenger and Fischer [PiF] sharpen this to $C_A(n) = O(T(n) \log T(n))$. There is no relationship in the other direction since the network model is *nonuniform*, i.e., we can choose a different network for each input length n. In particular, it is easy to see that $C_A(n)$ is $2^{O(n)}$ for any A (even nonrecursive A). However, if Turing machines are made nonuniform by giving them access to arbitrary oracle sets $D \subseteq \{1\}^*$ or if the network model is made uniform by requiring that the networks be efficiently constructable, then deterministic Turing machine time complexity and combinational complexity are polynomially related; see, for example, Schnorr [Schn]. Other relationships between Turing machine complexity and network complexity are proved by Borodin [Boro].

As mentioned in the Introduction, Ehrenfeucht [Ehr] has shown that the combinational complexity of the first-order theory of $\mathbf{N}$ with addition and multiplication grows faster than any polynomial in n, even if the theory is made decidable by bounding all quantifiers. A. Meyer has observed that a similar proof can be done for any problem which is hard for the class of problems accepted by ATM's which accept simultaneously within time $2^{O(n)}$ and within $O(1)$ alternations. For example, if A is $\mathrm{Th}(\mathbf{R}, +)$ or $\mathrm{Th}(\mathbf{N}, +)$, then $C_A(n) > c^n$ for some constant $c > 1$ and infinitely many n.

§8. Conclusion. The main message of this paper is that after a problem has been proved decidable, there is still the opportunity to obtain much more precise information about the problem's computational complexity. Precise classifications are often challenging mathematical questions. In addition, for problems with practical applications such information can be useful. If a problem requires exponential time i.o. or even if it is NP-complete, it suggests that one should not waste time looking for an algorithm which is always fast and always correct. A more reasonable approach for such problems might be to apply heuristic methods which are sometimes slow or sometimes incorrect (for example, see [GaJ, Chapter 6] for a discussion of "approximation" algorithms for certain NP-complete optimization problems) or to abandon the worst-case notion of complexity and seek algorithms which do well on the average (for example, see [Joh2] for a survey).

Efficient reducibility, completeness, and explicit lower bounds are now tools of the trade in the theoretical computer science community. These methods should be useful in other areas of mathematics where decision problems arise.

Acknowledgements. I would like to thank Ron Fagin, Yuri Gurevich, Joe Halpern, Albert Meyer, and Moshe Vardi for many helpful comments on early drafts of this paper.

REFERENCES

[AHU] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, ***The design and analysis of computer algorithms***, Addison-Wesley, Reading, Massachusetts, 1974.

[Amb] K. Ambos-Spies, *Three theorems on polynomial degrees of NP-sets*, ***Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*** (1985), pp. 51–55.

[As] G. Asser, *Das Repräsentantenproblem im Prädikatenkalkül der ersten Stufe mit Identität*, ***Zeitschrift für Mathematische Logik und Grundlagen der Mathematik***, vol. 1 (1955), pp. 252–263.

[Ba] L. Babai, *Trading group theory for randomness*, ***Proceedings of the 17th ACM Symposium on Theory of Computing*** (1985), pp. 421–429.

[BaGS] T. Baker, J. Gill, and R. Solovay, *Relativizations of the* $P = ?\ NP$ *question*, ***SIAM Journal on Computing***, vol. 4 (1975), pp. 431–442.

[BeKR] M. Ben-Or, D. Kozen, and J. Reif, *The complexity of elementary algebra and geometry*, ***Journal of Computer and System Sciences***, vol. 32 (1986), pp. 251–264.

[BeG] C. H. Bennett and J. Gill, *Relative to a random oracle A*, $P^A \neq NP^A \neq co\text{-}NP^A$ *with probability* 1, ***SIAM Journal on Computing***, vol. 10 (1981), pp. 96–113.

[Berl] E. R. Berlekamp, *Factoring polynomials over large finite fields*, ***Mathematics of Computation***, vol. 24 (1970), pp. 713–735.

[Ber1] L. Berman, *On the structure of complete sets: almost everywhere complexity and infinitely often speedup*, ***Proceedings of the 17th IEEE Symposium on Foundations of Computer Science*** (1976), 76–80.

[Ber2] ———, *The complexity of logical theories*, ***Theoretical Computer Science***, vol. 11 (1980), pp. 71–77.

[BerH] L. Berman and J. Hartmanis, *On isomorphisms and density of NP and other complete sets*, ***SIAM Journal on Computing***, vol. 6 (1977), pp. 305–322.

[Blu1] M. Blum, *A machine independent theory of the complexity of recursive functions*, ***Journal of the Association for Computing Machinery***, vol. 14 (1967), pp. 322–336.

[Blu2] ———, *On effective procedures for speeding up algorithms*, ***Journal of the Association for Computing Machinery***, vol. 18 (1971), pp. 290–305.

[Blu3] N. Blum, *A note on the parallel computation thesis*, ***Information Processing Letters***, vol. 17 (1983), pp. 203–205.

[Bo] R. V. Book, *Translational lemmas, polynomial time, and* $(\log n)^j$*-space*, ***Theoretical Computer Science***, vol. 1 (1976), pp. 215–226.

[Borg] E. Börger, *Spektralproblem and completeness of logical decision problems*, ***Logic and machines: decision problems and complexity***, Lecture Notes in Computer Science, vol. 171, Springer-Verlag, New York, 1984, pp. 333–356.

[Boro] A. Borodin, *On relating time and space to size and depth*, ***SIAM Journal on Computing***, vol. 6 (1977), pp. 733–744.

[BrM] A. R. Bruss and A. R. Meyer, *On time-space classes and their relation to the theory of real addition*. ***Theoretical Computer Science***, vol. 11 (1980), pp. 59–69.

[Bu] J. R. Büchi, *Weak second order arithmetic and finite automata*, ***Zeitschrift für Mathematische Logik und Grundlagen der Mathematik***, vol. 6 (1960), pp. 66–92.

[ChH] A. K. CHANDRA and D. HAREL, *Structure and complexity of relational queries*, ***Journal of Computer and System Sciences***, vol. 25 (1982), pp. 99–128.

[ChKS] A. K. CHANDRA, D. C. KOZEN, and L. J. STOCKMEYER, *Alternation*, ***Journal of the Association for Computing Machinery***, vol. 28 (1981), pp. 114–133.

[ChS] A. K. CHANDRA and L. J. STOCKMEYER, *Alternation*, ***Proceedings of the 17th IEEE Symposium on Foundations of Computer Science*** (1976), pp. 98–108.

[Cob] A. COBHAM, *The intrinsic computational difficulty of functions*, ***Proceedings of the 1964 international congress for logic, methodology and philosophy of science*** (Y. Bar-Hillel, editor), North-Holland, Amsterdam, 1965, pp. 24–30.

[Col] G. E. COLLINS, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition*, ***Second GI conference on automata theory and formal languages***, Lecture Notes in Computer Science, vol. 33, Springer-Verlag, New York, 1975, pp. 134–183.

[CoH] K. COMPTON and C. W. HENSON, *A new method for proving lower bounds on the computational complexity of first-order theories*, manuscript in preparation. 1984.

[CoL] A. CONDON and R. LADNER, *Probabilistic game automata*, ***Proceedings of the structure in complexity theory conference*** (Berkeley, California, 1986), Lecture Notes in Computer Science, vol. 223, Springer-Verlag, New York, 1986, pp. 144–162.

[Co1] S. A. COOK, *The complexity of theorem proving procedures*, ***Proceedings of the 3rd ACM Symposium on Theory of Computing*** (1971), pp. 151–158.

[Co2] ———, *A hierarchy for nondeterministic time complexity*, ***Journal of Computer and System Sciences***, vol. 7 (1973), pp. 343–353.

[Co3]———, *An observation on time-storage trade off*, ***Journal of Computer and System Sciences***, vol. 9 (1974), pp. 308–316.

[Co4] ———, *Towards a complexity theory of synchronous parallel computation*, ***L'Enseignement Mathématique***, vol. 27 (1981), pp. 99–124; also, Technical Report #141/80, Department of Computer Science, University of Toronto, 1980.

[Co5] ———, *A taxonomy of problems with fast parallel algorithms*, ***Information and Control***, vol. 64 (1985), pp. 2–22.

[Co6] ———, *An overview of computational complexity*, ***Communications of the ACM***, vol. 26 (1983), pp. 400–408.

[CoR] S. A. COOK and R. A. RECKHOW, *Time bounded random access machines*, ***Journal of Computer and System Sciences***, vol. 7 (1973), pp. 354–375.

[Cs] L. CSANKY, *Fast parallel matrix inversion algorithms*, ***SIAM Journal on Computing***, vol. 5 (1976), pp. 618–623.

[Dav] M. DAVIS, ***The undecidable***, Raven Press, Hewlett, New York, 1965.

[Dek] M. DEKHTIAR, *On the impossibility of eliminating complete enumeration in computing a function relative to its graph*, ***Doklady Akademii Nauk SSSR***, vol. 189 (1969), pp. 748–751 (in Russian); English translation, ***Soviet Physics Doklady***, vol. 14 (1969/70), pp. 1146–1148.

[DoLR] D. DOBKIN, R. J. LIPTON, and S. REISS, *Linear programming is* log-*space hard for P*, ***Information Processing Letters***, vol. 8 (1979), pp. 96–97.

[DKM] C. DWORK, P. C. KANELLAKIS, and J. C. MITCHELL, *On the sequential nature of unification*, ***Journal of Logic Programming***, vol. 1 (1984), pp. 35–50.

[Edm] J. EDMONDS, *Paths, trees and flowers*, ***Canadian Journal of Mathematics***, vol. 17 (1965), pp. 449–467.

[Ehr] A. EHRENFEUCHT, *Practical decidability*, ***Journal of Computer and System Sciences***, vol. 11 (1975), pp. 392–396.

[Elg] C. C. ELGOT, *Decision problems of finite automata design and related arithmetics*, ***Transactions of the American Mathematical Society***, vol. 98 (1961), pp. 21–51.

[EvSY] S. EVEN, A. L. SELMAN, and Y. YACOBI, *Hard-core theorems for complexity classes*, ***Journal of the Association for Computing Machinery***, vol. 32 (1985), pp. 205–217.

[Fa] R. FAGIN, *Generalized first-order spectra and polynomial-time recognizable sets*, ***Complexity of computation*** (R. Karp, editor), SIAM-AMS Proceedings, vol. 7, American Mathematical Society, Providence, Rhode Island, 1974, pp. 43–73.

[Fe] J. FERRANTE, ***Some upper and lower bounds on decision procedures in logic***, Doctoral Thesis, Department of Mathematics, M.I.T., Cambridge, Massachusetts, 1974; also Report TR-139, M.I.T. Laboratory for Computer Science.

[FeR1] J. FERRANTE and C. RACKOFF, *A decision procedure for the first-order theory of real addition with order*, ***SIAM Journal on Computing***, vol. 4 (1975), pp. 69–76.

[FeR2] ———, ***The computational complexity of logical theories***, Lecture Notes in Mathematics, vol. 718, Springer-Verlag, New York, 1979.

[FiL] M. J. FISCHER and R. E. LADNER, *Propositional dynamic logic of regular programs*, ***Journal of Computer and System Sciences***, vol. 18 (1979), pp. 194–211.

[FiR] M. J. FISCHER and M. O. RABIN, *Super-exponential complexity of Presburger arithmetic*, ***Complexity of computation*** (R. Karp, editor), SIAM-AMS Proceedings, vol. 7, American Mathematical Society, Providence, Rhode Island, 1974, pp. 27–42.

[FoW] S. FORTUNE and J. WYLLIE, *Parallelism in random access machines*, ***Proceedings of the 10th ACM Symposium on Theory of Computing*** (1978), pp. 114–118.

[FrL] A. S. FRAENKEL and D. LICHTENSTEIN, *Computing a perfect strategy for n × n chess requires time exponential in n*, ***Journal of Combinatorial Theory A***, vol. 31 (1981), pp. 199–214.

[Fu1] M. FÜRER, ***Nicht-elementare untere Schranken in der Automaten-theorie***, Doctoral Thesis, ETH, Zürich, 1978.

[Fu2] ———, *The complexity of Presburger arithmetic with bounded quantifier alternation depth*, ***Theoretical Computer Science***, vol. 18 (1982), pp. 105–111.

[FuSS] M. FURST, J. B. SAXE, and M. SIPSER, *Parity, circuits, and the polynomial-time hierarchy*, ***Mathematical Systems Theory***, vol. 17 (1984), pp. 13–27.

[GaJ] M. R. GAREY and D. S. JOHNSON, ***Computers and intractability: a guide to the theory of NP-completeness***, Freeman, San Francisco, California, 1979.

[Gi] J. GILL, *Computational complexity of probabilistic Turing machines*, ***SIAM Journal on Computing***, vol. 6 (1977), pp. 675–695.

[Go1] L. M. GOLDSCHLAGER, *The monotone and planar circuit value problems are log space complete for P*, ***SIGACT News***, vol. 9 (1977), no. 2, pp. 25–29.

[Go2] ———, *A universal interconnection pattern for parallel computers*, ***Journal of the Association for Computing Machinery***, vol. 29 (1982), pp. 1073–1086.

[GoSS] L. M. GOLDSCHLAGER, R. A. SHAW, and J. STAPLES, *The maximum flow problem is log space complete for P*, ***Theoretical Computer Science***, vol. 21 (1982), pp. 105–111.

[GoSi] S. GOLDWASSER and M. SIPSER, *Private coins versus public coins in interactive proof systems*, ***Proceedings of the 18th ACM Symposium on Theory of Computing*** (1986), pp. 59–68.

[Gu1] Y. GUREVICH, *Algebras of feasible functions*, ***Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*** (1983), pp. 210–214.

[Gu2] ———, *Toward logic tailored for computational complexity*, ***Computation and proof theory*** (M. M. Richter et. al., editors), Lecture Notes in Mathematics, vol. 1104, Springer-Verlag, New York, 1984, pp. 175–216.

[Gu3] ———, ***Logic and the challenge of computer science***, Report CRL-TR-10-85, Computing Research Laboratory, University of Michigan, Ann Arbor, Michigan, 1985.

[Hac] M. HACK, *The equality problem for vector addition systems is undecidable*, ***Theoretical Computer Science***, vol. 2 (1976), pp. 77–96.

[HalM] J. Y. HALPERN and Y. MOSES, *A guide to the modal logics of knowledge and belief*, ***Proceedings of the international joint conference on artificial intelligence*** (1985), American Association for Artificial Intelligence, Menlo Park, California, 1985, pp. 480–490.

[HalR] J. Y. HALPERN and J. REIF, *The propositional dynamic logic of deterministic well-structured programs*, ***Theoretical Computer Science***, vol. 27 (1983), pp. 127–165.

[HalV] J. Y. HALPERN and M. VARDI, *The complexity of reasoning about knowledge and time*, ***Proceedings of the 18th ACM Symposium on Theory of Computing*** (1986), pp. 304–315.

[Har] J. HARTMANIS, *Observations about the development of theoretical computer science*, ***Annals of the History of Computing***, vol. 3 (1981), pp. 42–51.

[HarM] J. HARTMANIS and S. R. MAHANEY, *An essay about research on sparse complete sets*, ***Proceedings of the 9th symposium on mathematical foundations of computer science*** (1980), Lecture Notes in Computer Science, vol. 88, Springer-Verlag, New York, pp. 40–57.

[HarS] J. HARTMANIS and R. E. STEARNS, *On the computational complexity of algorithms*, ***Transactions of the American Mathematical Society***, vol. 117 (1965), pp. 285–306.

[Has] J. HASTAD, *Almost optimal lower bounds for small depth circuits*, ***Proceedings of the 18th ACM Symposium on Theory of Computing*** (1986), pp. 6–20.

[HooR] H. J. Hoover and W. L. Ruzzo, ***A compendium of problems complete for P***, Technical Report, Department of Computer Science, University of Washington, Seattle, Washington (to appear).

[HoPV] J. Hopcroft, W. Paul, and L. Valiant, *On time versus space*, ***Journal of the Association for Computing Machinery***, vol. 24 (1977), pp. 332–337.

[HoU] J. E. Hopcroft and J. D. Ullman, ***Introduction to automata theory, languages, and computation***, Addison-Wesley, Reading, Massachusetts, 1979.

[HuR1] H. B. Hunt, III, and D. J. Rosenkrantz, *On equivalence and containment problems for formal languages*, ***Journal of the Association for Computing Machinery***, vol. 24 (1977), pp. 387–396.

[HuR2] ———, *Computational parallels between the regular and context-free languages*, ***SIAM Journal on Computing***, vol. 7 (1978), pp. 99–114.

[HuRS] H. B. Hunt, III, D. J. Rosenkrantz, and T. G. Szymanski, *On the equivalence, containment, and covering problems for the regular and context-free languages*, ***Journal of Computer and System Sciences***, vol. 12 (1976), pp. 222–268.

[Huy] D. T. Huynh, *Deciding the inequivalence of context-free grammars with 1-letter terminal alphabet is Σ_2^p-complete*, ***Theoretical Computer Science***, vol. 33 (1984), pp. 305–326.

[Ib] O. H. Ibarra, *A note concerning nondeterministic tape complexities*, ***Journal of the Association for Computing Machinery***, vol. 19 (1972), pp. 608–612.

[IbM] O. H. Ibarra and S. Moran, *Probabilistic algorithms for deciding equivalence of straight-line programs*, ***Journal of the Association for Computing Machinery***, vol. 30 (1983), pp. 217–228.

[Im1] N. Immerman, Relational queries computable in polynomial time, ***Proceedings of the 14th ACM Symposium on Theory of Computing*** (1982), 147–152.

[Im2] ———, *Languages which capture complexity classes*, ***Proceedings of the 15th ACM Symposium on Theory of Computing*** (1983), 347–354.

[Je] R. G. Jeroslow, ***The polynomial hierarchy and a simple model for competitive analysis***, Report No. 83272-OR, Institut für Ökonometrie und Operations Research, Rheinische Friedrich-Wilhelms-Universität, Bonn, 1983.

[Joh1] D. S. Johnson, *The NP-completeness column: an ongoing guide*, ***Journal of Algorithms***, vol. 4 (1983), pp. 397–411.

[Joh2] ———, *The NP-completeness column: an ongoing guide*, ***Journal of Algorithms***, vol. 5 (1984), pp. 284–299.

[Joh3] ———, *The NP-completeness column: an ongoing guide*, ***Journal of Algorithms***, vol. 5 (1984), pp. 433–447.

[Joh4] ———, *The NP-completeness column: an ongoing guide*, ***Journal of Algorithms***, vol. 6 (1985), pp. 291–305.

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

[JoL] N. D. Jones and W. T. Laaser, *Complete problems for deterministic polynomial time*, ***Theoretical Computer Science***, vol. 3 (1977), pp. 105–117.

[JoLL] N. D. Jones, Y. E. Lien, and W. T. Laaser, *New problems complete for nondeterministic log space*, ***Mathematical Systems Theory***, vol. 10 (1976), pp. 1–18.

[JoS] N. D. Jones and A. L. Selman, *Turing machines and the spectra of first-order formulas*, this Journal, vol. 39 (1974), pp. 139–150.

[Ka1] R. M. Karp, *Reducibility among combinatorial problems*, ***Complexity of computer computations*** (R. E. Miller and J. W. Thatcher, editors), Plenum Press, New York, 1972, pp. 85–104.

[Ka2] ———, *Combinatorics, complexity, and randomness*, ***Communications of the ACM***, vol. 29 (1986), pp. 98–109.

[KaM] R. M. Karp and R. E. Miller, *Parallel program schemata*, ***Journal of Computer and System Sciences***, vol. 3 (1969), pp. 147–195.

[Kh] L. G. Khachiyan, *A polynomial algorithm for linear programming*, ***Doklady Akademii Nauk SSSR***, vol. 244 (1979), pp. 1093–1096 (in Russian); English translation, ***Soviet Mathematics Doklady***, vol. 20 (1979), pp. 191–194.

[Ko1] D. Kozen, *Complexity of finitely presented algebras*, ***Proceedings of the 9th ACM Symposium on Theory of Computing*** (1977), pp. 164–177.

[Ko2] ———, *Complexity of Boolean algebras*, ***Theoretical Computer Science***, vol. 10 (1980), pp. 221–248.

[Lad1] R. E. LADNER, *On the structure of polynomial time reducibility*, ***Journal of the Association for Computing Machinery***, vol. 22 (1975), pp. 155–171.

[Lad2] ———, *The circuit value problem is log space complete for P*, ***SIGACT News***, vol. 7 (1975), no. 1, pp. 18–20.

[Lad3] ———, *The computational complexity of provability in systems of modal propositional logic*, ***SIAM Journal on Computing***, vol. 6 (1977), pp. 467–480.

[LaLS] R. E. LADNER, N. A. LYNCH, and A. L. SELMAN, *A comparison of polynomial time reducibilities*, ***Theoretical Computer Science***, vol. 1 (1975), pp. 103–123.

[LaN] R. E. LADNER and J. K. NORMAN, *Solitaire automata*, ***Journal of Computer and System Sciences***, vol. 30 (1985), pp. 116–129.

[Lau] C. LAUTEMANN, *BPP and the polynomial hierarchy*, ***Information Processing Letters***, vol. 17 (1983), pp. 215–217.

[Lev] L. A. LEVIN, *Universal sorting problems*, ***Problemy Peredachi Informatsii***, vol. 9 (1973), no. 3, pp. 115–116 (in Russian); English translation, ***Problems of Information Transmission***, vol. 9 (1973), pp. 265–266.

[Lew] H. R. LEWIS, *Complexity results for classes of quantificational formulas*, ***Journal of Computer and System Sciences***, vol. 21 (1980), pp. 317–353.

[Liv] A. B. LIVCHAK, *The relational model for process control*, ***Nauchno-Tekhnicheskaya Informatsiya***, ser. 2, 1983, no. 4, pp. 27–29 (in Russian); English translation in ***Automatic Documentation and Mathematical Linguistics***, vol. 17 (1983), pp. 105–110.

[Lo] L. LO, ***On the computational complexity of the theory of abelian groups***, Ph.D. Thesis, University of Michigan, Ann Arbor, Michigan, 1984.

[Ly] N. LYNCH, *On reducibility to complex or sparse sets*, ***Journal of the Association for Computing Machinery***, vol. 22 (1975), pp. 341–345.

[Mah1] S. R. MAHANEY, *On the number of p-isomorphism classes of NP-complete sets*, ***Proceedings of the 22nd IEEE Symposium on Foundations of Computer Science*** (1981), pp. 271–278.

[Mah2] ———, *Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis*, ***Journal of Computer and System Sciences***, vol. 25 (1982), pp. 130–143.

[MaM1] E. MAYR and A. R. MEYER, *The complexity of the finite containment problem for Petri nets*, ***Journal of the Association for Computing Machinery***, vol. 28 (1981), pp. 561–576.

[MaM2] ———, *The complexity of the word problems for commutative semigroups and polynomial ideals*, ***Advances in Mathematics***, vol. 46 (1982), pp. 305–329.

[Me1] A. R. MEYER, *Weak monadic second-order theory of successor is not elementary-recursive*, ***Logic colloquium: symposium on logic held at Boston 1972–73*** (R. Parikh, editor), Lecture Notes in Mathematics, vol. 453, Springer-Verlag, New York, 1975, pp. 132–154.

[Me2] ———, *The inherent computational complexity of theories of ordered sets*, ***Proceedings of the International Congress of Mathematicians (Vancouver, 1974)***, Vol. 2, Canadian Mathematical Congress, Montréal, 1975, pp. 477–482.

[MeS] A. R. MEYER and L. J. STOCKMEYER, *The equivalence problem for regular expressions with squaring requires exponential space*, ***Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*** (1972), pp. 125–129.

[Mi] G. L. MILLER, *Riemann's hypothesis and tests for primality*, ***Journal of Computer and System Sciences***, vol. 13 (1976), pp. 300–317.

[Mo] L. MONK, ***Elementary recursive decision procedures***, Ph.D. Thesis, University of California, Berkeley, California, 1975.

[Mos] Y. N. MOSCHOVAKIS, ***Elementary induction on abstract structures***, North-Holland, Amsterdam, 1974.

[Pa1] C. H. PAPADIMITRIOU, *On the complexity of unique solutions*, ***Journal of the Association for Computing Machinery***, vol. 31 (1984), pp. 392–400.

[Pa2] ———, *Games against nature*, ***Journal of Computer and System Sciences***, vol. 31 (1985), pp. 288–301.

[PaS] C. H. PAPADIMITRIOU and K. STEIGLITZ, ***Combinatorial optimization: algorithms and complexity***, Prentice-Hall, Englewood Cliffs, N.J., 1982.

[PaW] C. H. PAPADIMITRIOU and D. WOLFE, *The complexity of facets resolved*, ***Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*** (1985), pp. 74–78.

[PaY] C. H. PAPADIMITRIOU and M. YANNAKAKIS, *The complexity of facets (and some facets of complexity)*, ***Journal of Computer and System Sciences***, vol. 28 (1984), pp. 244–259.

[PPST] W. J. PAUL, N. PIPPENGER, E. SZEMERÉDI, and W. T. TROTTER, *On determinism versus nondeterminism and related problems*, ***Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*** (1983), pp. 429–438.

[Pi] N. PIPPENGER, *On simultaneous resource bounds*, ***Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*** (1979), pp. 307–311.

[PiF] N. PIPPENGER and M. J. FISCHER, *Relations among complexity measures*, ***Journal of the Association for Computing Machinery***, vol. 26 (1979), pp. 361–381.

[Pl] D. A. PLAISTED, *Complete problems in the first-order predicate calculus*, ***Journal of Computer and System Sciences***, vol. 29 (1984), pp. 8–35.

[Pn] A. PNUELI, *The temporal logic of programs*, ***Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*** (1977), pp. 46–57.

[Pr1] V. R. PRATT, *Semantical considerations on Floyd-Hoare logic*, ***Proceedings of the 17th IEEE Symposium on Foundations of Computer Science*** (1976), pp. 109–121.

[Pr2] ———, Models of program logics, ***Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*** (1979), pp. 115–122.

[PrS] V. R. PRATT and L. J. STOCKMEYER, *A characterization of the power of vector machines*, ***Journal of Computer and System Sciences***, vol. 12 (1976), pp. 198–221.

[Rab1] M. O. RABIN, ***Degree of difficulty of computing a function and a partial ordering of recursive sets***, Technical Report 2, Hebrew University, Jerusalem, 1960.

[Rab2] ———, *A simple method for undecidability proofs and some applications*, ***Proceedings of the 1964 international congress for logic, methodology and philosophy of science*** (Y. Bar-Hillel, editor), North-Holland, Amsterdam, 1965, pp. 58–68.

[Rab3] ———, *Decidability of second-order theories and automata on infinite trees*, ***Transactions of the American Mathematical Society***, vol. 141 (1969), pp. 1–35.

[Rab4] ———, *Theoretical impediments to artificial intelligence*, ***Information Processing 74*** (J. Rosenfeld, editor), North-Holland, Amsterdam, 1974, pp. 615–619.

[Rab5] ———, *Probabilistic algorithm for testing primality*, ***Journal of Number Theory***, vol. 12 (1980), pp. 128–138.

[Rab6] ———, *Probabilistic algorithms in finite fields*, ***SIAM Journal on Computing***, vol. 9 (1980), pp. 273–280.

[Rac1] C. W. RACKOFF, ***The computational complexity of some logical theories***, Doctoral Thesis, Department of Electrical Engineering, M.I.T., Cambridge, Massachusetts, 1974; also Report TR-144, M.I.T. Laboratory for Computer Science.

[Rac2] ———, ***The complexity of theories of the monadic predicate calculus***, Research Report #136, IRIA-LABORIA, France, October 1975.

[Rac3] ———, *On the complexity of the theories of weak direct powers*, this JOURNAL, vol. 41 (1976), pp. 561–573.

[Rac4] ———, *Relativized questions involving probabilistic algorithms*, ***Journal of the Association for Computing Machinery***, vol. 29 (1982), pp. 261–268.

[ReL] C. R. REDDY and D. W. LOVELAND, *Presburger arithmetic with bounded quantifier alternation*, ***Proceedings of the 10th ACM Symposium on Theory of Computing*** (1978), pp. 320–325.

[Reif] J. REIF, *The complexity of two-player games of incomplete information*, ***Journal of Computer and System Sciences***, vol. 29 (1984), pp. 274–301.

[Reis] S. REISCH, *Hex ist* PSPACE*-vollständig*, ***Acta Informatica***, vol. 15 (1981), pp. 167–191.

[Ro] E. L. ROBERTSON, *Structure of complexity in the weak monadic second-order theories of the natural numbers*, ***Proceedings of the 6th ACM Symposium on Theory of Computing*** (1974), pp. 161–171.

[Rob1] J. M. ROBSON, *N by N checkers is exptime complete*, ***SIAM Journal on Computing***, vol. 13 (1984), pp. 252–267.

[Rob2] ———, *The complexity of go*, ***Information Processing 83 (Proceedings of the ninth IFIP world computer congress, Paris, 1983***; R. E. A. Mason, editor), North-Holland, Amsterdam, 1983, pp. 413–418.

[Rog] H. ROGERS, JR., ***Theory of recursive functions and effective computability***, McGraw-Hill, New York, 1967.

[Ruz] W. L. RUZZO, *On uniform circuit complexity*, ***Journal of Computer and System Sciences***, vol. 22 (1981), pp. 365–383.

[SaY] Y. SAGIV and M. YANNAKAKIS, *Equivalences among relational expressions with the union and difference operators*, ***Journal of the Association for Computing Machinery***, vol. 27 (1980), pp. 633–655.

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

[Scha] T. J. SCHAEFER, *On the complexity of some two-person perfect-information games*, ***Journal of Computer and System Sciences***, vol. 16 (1978), pp. 185–225.

[Schn] C. P. SCHNORR, *The network complexity and the Turing machine complexity of finite functions*, ***Acta Informatica***, vol. 7 (1976), pp. 95–107.

[Scho] H. SCHOLZ, *Ein ungelöstes Problem in der symbolischen Logik*, this JOURNAL, vol. 17 (1952), p. 160.

[Schw] J. T. SCHWARTZ, *Fast probabilistic algorithms for verification of polynomial identities*, ***Journal of the Association for Computing Machinery***, vol. 27 (1980), pp. 701–717.

[Se] J. I. SEIFERAS, *Relating refined space complexity classes*, ***Journal of Computer and System Sciences***, vol. 14 (1977), pp. 100–129.

[SeFM] J. I. SEIFERAS, M. J. FISCHER, and A. R. MEYER, *Separating nondeterministic time complexity classes*, ***Journal of the Association for Computing Machinery***, vol. 25 (1978), pp. 146–167.

[Sip1] M. SIPSER, *On relativization and the existence of complete sets*, ***Automata, languages and programming (Aarhus, 1982)***, Lecture Notes in Computer Science, vol. 140, Springer-Verlag, New York, 1982, pp. 523–531.

[Sip2] ———, *Borel sets and circuit complexity*, ***Proceedings of the 15th ACM Symposium on Theory of Computing*** (1983), pp. 61–69.

[Sip3] ———, *A complexity theoretic approach to randomness*, ***Proceedings of the 15th ACM Symposium on Theory of Computing*** (1983), pp. 330–335.

[SisC] A. P. SISTLA and E. M. CLARKE, *The complexity of propositional linear temporal logics*, ***Journal of the Association for Computing Machinery***, vol. 32 (1985), pp.733–749.

[SiVW] A. P. SISTLA, M. Y. VARDI, and P. WOLPER, *The complementation problem for Büchi automata with applications to temporal logic*, ***Proceedings of the 12th international colloquium on automata, languages and programming***, Lecture Notes in Computer Science, vol. 194, Springer-Verlag, New York, 1985, pp. 465–474.

[SoS] R. SOLOVAY and V. STRASSEN, *A fast Monte-Carlo test for primality*, ***SIAM Journal on Computing***, vol. 6 (1977), pp. 84–85; erratum, vol. 7 (1978), p. 118.

[StHL] R. E. STEARNS, J. HARTMANIS, and P. M. LEWIS, II, *Hierarchies of memory limited computations*, ***Proceedings of the 6th IEEE Symposium on Switching Circuit Theory and Logical Design*** (1965), pp. 179–190.

[Sto1] L. J. STOCKMEYER, ***The complexity of decision problems in automata theory and logic***, Doctoral Thesis, Department of Electrical Engineering, M.I.T., Cambridge, Massachusetts, 1974; also Report TR-133, M.I.T. Laboratory for Computer Science.

[Sto2] ———, *The polynomial-time hierarchy*, ***Theoretical Computer Science***, vol. 3 (1977), pp. 1–22.

[StoC] L. J. STOCKMEYER and A. K. CHANDRA, *Provably difficult combinatorial games*, ***SIAM Journal on Computing***, vol. 8 (1979), pp. 151–174.

[StoM] L. J. STOCKMEYER and A. R. MEYER, *Word problems requiring exponential time: preliminary report*, ***Proceedings of the 5th ACM Symposium on Theory of Computing*** (1973), pp. 1–9.

[Sud] I. H. SUDBOROUGH, *A note on tape-bounded complexity classes and linear context-free languages*, ***Journal of the Association for Computing Machinery***, vol. 22 (1975), pp. 499–500.

[Tarj] R. E. TARJAN, ***Data structures and network algorithms***, CBMS-NSF Regional Conference Series in Applied Mathematics, no. 44, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1983.

[Tars] A. TARSKI, ***A decision method for elementary algebra and geometry***, 2nd ed., University of California Press, Berkeley, California, 1951.

[Va1] L. G. VALIANT, *The complexity of computing the permanent*, ***Theoretical Computer Science***, vol. 8 (1979), pp. 189–202.

[Va2] ———, *The complexity of enumeration and reliability problems*, ***SIAM Journal on Computing***, vol. 8 (1979), pp. 410–421.

[VaV] L. G. VALIANT and V. V. VAZIRANI, *NP is as easy as detecting unique solutions*, ***Proceedings of the 17th ACM Symposium on Theory of Computing*** (1985), pp. 458–463.

[Var] M. Y. VARDI, *The complexity of relational query languages*, ***Proceedings of the 14th ACM Symposium on Theory of Computing*** (1982), pp. 137–146.

[VarS] M. Y. VARDI and L. STOCKMEYER, *Improved upper and lower bounds for modal logics of programs: preliminary report*, ***Proceedings of the 17th ACM Symposium on Theory of Computing*** (1985), pp. 240–251.

[Wr] C. WRATHALL, *Complete sets and the polynomial-time hierarchy*, ***Theoretical Computer Science***, vol. 3 (1977), pp. 23–33.

[Ya] A. C. YAO, *Separating the polynomial-time hierarchy by oracles*, ***Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*** (1985), pp. 1–10.

IBM ALMADEN RESEARCH CENTER
SAN JOSE, CALIFORNIA 95120