CHAPTER 2

A Catalog of Complexity Classes

David S. JOHNSON

AT&T Bell Laboratories, Murray Hill, NJ 07974, USA

Contents

HANDBOOK OF THEORETICAL COMPUTER SCIENCE
Edited by J. van Leeuwen
© Elsevier Science Publishers B.V., 1990

1. Preliminaries

One of the goals of complexity theory is to classify problems as to their intrinsic computational difficulty. Given a problem, how much computing power and/or resources do we need in order to solve it? To date, we have not made much progress toward finding precise answers to such questions. We have, however, made a great deal of progress in classifying problems into general "complexity classes", which characterize, at least in a rough way, something of their inherent difficulties. This chapter will survey the most popular such classes, and the types of problems they contain.

This first section contains the preliminary definitions and concepts needed for defining the complexity classes. Sections 2 through 5 can be viewed as a catalog of complexity classes, with Section 6 devoted to last-minute developments, and tables and figures for use as ready-reference material. In addition to providing definitions of the classes, we will discuss what is known about the relationships between them. We shall also, where possible, provide examples of typical problems in each class, and describe something of what is known of the "structure" of the class.

1.1. Problems and instances

We begin by defining our terms. For our purposes, a "problem" is a total relation on strings, say over the alphabet $\{0,1\}$. More precisely, we have the following definition (where "$\{0,1\}^*$" represents the set of all finite strings made up of 0s and 1s).

Definition. A *problem* is a set X of ordered pairs (I, A) of strings in $\{0,1\}^*$, where I is called the *instance*. A is called an *answer* for that instance, and every string in $\{0,1\}^*$ occurs as the first component of at least one pair.

This abstract and technical definition is required because the complexity classes we shall be discussing are all defined in terms of machine models like those covered in [153, 246]. Such machines are only equipped to handle inputs and outputs that are strings, rather than the more interesting combinatorial objects that we normally think of as the subjects of "problems", such as graphs, equations, or logical expressions.

Note, however, that such combinatorial objects are not all that different from strings, at least in the ways we use them. As a cogent example, consider the following problem.

GRAPH ISOMORPHISM

Instance: Two undirected graphs $G=(V,E)$ and $G'=(V',E')$, where V and V' are finite sets of vertices, and E and E' are finite sets of edges (unordered pairs of vertices from V and V' respectively).

Answer: "Yes" if there is a one-one onto function $f: V \rightarrow V'$ such that for all pairs $\{u,v\} \subseteq V$, $\{u,v\} \in E$ if and only if $\{f(u),f(v)\} \in E'$. Otherwise, "no".

As defined, this problem makes sense only as a question about *descriptions* of graphs, asking in effect whether two descriptions describe the same graph. In other words, it is a question about symbolic representations of the combinatorial objects called graphs, rather than the graphs themselves. Typically, we will present such symbolic representations in a linear fashion, i.e., as strings. It is thus not too much of a jump to think of the GRAPH ISOMORPHISM problem as a string relation. Let a_Y and a_N be strings chosen to represent "yes" and "no" respectively. The string relation would then be

$$\{(x, a_Y)\text{: string } x \text{ consists of two representations of the same graph } G\}$$
$$\cup\{(x, a_N)\text{: string } x \text{ does not consist of two representations of the same graph } G\}.$$

Similar translations will work for any other problem we might consider solving by digital computer, since we must have some way to represent instances and answers as strings if we hope to represent them in computer memory, which itself can be viewed as a string.

The above definition of the string relation corresponding to the GRAPH ISOMORPHISM problem is of course incomplete, for we have failed to specify precisely how strings are to represent graphs. As pointed out in [247], various schemes are available, from adjacency lists to incidence matrices. And even if we choose one basic scheme, say adjacency lists, there are still many fine details of punctuation and syntax to be filled in. Each choice would give a different string relation.

Fortunately, all the "reasonable" representation schemes are relatively interchangeable. Given one form of representation, one can quickly translate to any other. As we shall explain more fully in Section 1.5, it is normally safe to describe a problem in abstract terms, leaving the representational details to be filled in in the standard ways. The results we quote will hold no matter which string relation would result, assuming the details are "reasonable". When representational issues do make a difference, we shall say so.

We conclude this section with several technical points. First, recall that we require that a string relation be total if it is to be considered a "problem". Thus in our formulation of GRAPH ISOMORPHISM as a string relation, the answer is "no" both when x consists of representations of different graphs and when x does not represent graphs at all. Alternatively, one might wish to have a third "answer string" a_M ("M" for "meaningless input"). In either case, our definition requires that the problem be completely specified, even for meaningless input.

Second, note that our definition of problem does not require that the relation be a function, i.e., there can be more than one pair in X that have the same first component. For instance, in the GRAPH ISOMORPHISM problem, each pair (x, a_Y) could be replaced by the set of pairs of the form (x, w), where w is the representation of a function f that provides an isomorphism between the two graph representations included in x (there can be more than one such f). In this new problem, *any* w such that $(x, w) \in X$ would be a satisfactory answer for instance x. If one desired *all* such w, one could define a different problem X' where $(x, w) \in X'$ if x consists of two graph representations and w is the (possible empty) list contaning all the possible isomorphisms between the two graphs represented by x.

As the above example illustrates, our definition of problem is sufficiently general to meet most needs. To emphasize this generality, string relations are sometimes referred to as generalized "search problems" [87]. This is in opposition to such important special cases as *functions*, *decision problems*, and *counting problems*, defined below.

DEFINITION. A *function* is a string relation in which each string $x \in \{0, 1\}^*$ is the first component of precisely one pair.

DEFINITION. A *decision probem* is a function in which the only possible answers are "yes" and "no".

DEFINITION. A *counting problem* is a function in which all answers are nonnegative integers.

Decision problems are a particularly important special case. Most of the complexity classes we shall be considering are restricted to such problems, or, more precisely, to the *languages* derived from them.

DEFINITION. A *language* is any subset of $\{0, 1\}^*$.

There is a natural correspondence between languages and decision problems:

DEFINITION. If L is a language, then the decision problem R_L corresponding to L is $\{(x, \text{yes}): x \in L\} \cup \{(x, \text{no}): x \notin L\}$.

DEFINITION. Given a decision problem R, the language $L(R)$ corresponding to it is simply $L(R) = \{x \in \{0, 1\}^*: (x, \text{yes}) \in R\}$.

Note that there is an asymmetry between "yes" and "no" in this latter definition (i.e., "no" does not appear although "yes" does). Interchanging "yes" and "no" would yield a different and "complementary" language, defined as follows.

DEFINITION. If L is a language, then its *complementary language* is co-$L = \{0, 1\}^* - L$.

This asymmetry in the definition of $L(R)$ is of substantial theoretical importance. As we shall see in the next section, many of our models of computation, in particular the nondeterministic ones, are similarly asymmetrical, and hence $L(R)$ and co-$L(R)$ need not always belong to the same complexity class. Indeed, a common question we shall be asking is whether a given complexity class C is "closed under complement", i.e., whether $L \in C$ implies co-$L \in C$ for all languages L.

1.2. How to solve a problem

In everyday speech we often talk about "solving" particular instances of problems, as in solving a crossword puzzle. Complexity theorists, however, do not consider a problem X solved unless they have a general method that will work for *any* instance (assuming enough time, memory, and other resources are provided). In practice, such methods may be usable only for a small finite set of instances, given physical bounds on the resources available. Because the methods are general, however, they will automatically let us solve larger instances should the amount of available resources ever increase. The key question about such methods is thus how their resource requirements increase with instance size. As we shall see, there are many different notions of what a "method" can be, some rather abstract, but the following definitions will be relevant to all.

DEFINITION. The *size* of an instance I, written $|I|$, is its length, i.e., the number of symbols it contains. (Recall that formally our problems are string relations and our instances are strings.)

DEFINITION. If M is a method for solving problem X and R is a resource used by that method, then $R_M : Z^+ \rightarrow Z^+$ is the function defined by letting $R_M(n)$ be the maximum, over all strings x of length n, of the amount of resource R used when M is applied to input x.

Note that $R_M(n)$ is a worst-case measure. For a particular instance x with $|x| = n$, and even for most such instances, M may use far less than $R_M(n)$ of resource R. However, $R_M(n)$ provides the best possible overall guarantee, and thus provides a certainty that an average-case measure cannot.

For the purpose of defining complexity classes, it is useful to be able to talk about the resource requirements of problems themselves, rather than just of the particular methods used to solve them. It is difficult to define this notion of "requirement" directly, however, due to the fact that there may be no "best" method for a given problem X. For example, it could be the case that there are methods $M[\alpha]$ with $R_{M[\alpha]}(n) = n^{1+\alpha}$ for all $\alpha > 0$, but no method M with $R_M(n) = O(n)$. We shall thus settle for an indirect definition in terms of upper and lower bounds. Suppose X is a problem, R is a resource, and C is a class of methods.

DEFINITION. The requirements of problem X for resource R under methods from C is *upper bounded by* $T(n)$ if and only if there is a method $M \in C$ for solving X that has $R_M(n) = O(T(n))$.

DEFINITION. The requirement of problem X for resource R under methods from C is *lower bounded by* $T(n)$ if and only if all methods $M \in C$ for solving X have $R_M(n) = \Omega(T(n))$.

The above definitions are in terms of "classes" of methods. So far the term "method" has been used to represent the thing that does the "solving", but has otherwise been left unspecified. There are many notions of "method" available, each with its own uses and applicability. The goal of any method that solves problem X is, given an instance x, to produce an a such that $(x, a) \in X$. How that a is produced, however, can vary substantially. Depending on the class of methods in use and the type of problem, the answer a might be represented by anything from the contents of an output tape to the result of applying some logical or arithmetic function to the structure of an abstract computation tree.

Our choice of "method class" can also yield very different notions of the resource requirements for a problem. This choice, and the bounds we place on the relevant resource usage functions, are what defines a complexity class. In the next section, we survey the main classes of methods that we shall consider.

1.3. Machine models

Each class of methods that we discuss is most easily described in terms of a model of computation. A standard variant on the deterministic Turing machine can serve as our first model and a natural point of reference.

Deterministic Turning machines (DTMs)

For a general introduction to Turing machines, see [246]. In our variant, the machine has three tapes, one semi-infinite read-only tape for input, one semi-infinite write-only tape for output, and a read-write worktape. Such a machine "solves" a problem, if, whenever it is started with a string written in the leftmost cells of its input tape (all other cells blank), it eventually halts with an acceptable answer written in the leftmost cells of the output tape (all other cells blank). We will be principally interested in two resources for the basic machine, "time" and "space". The *time* for a computation is simply the number of steps made before the machine halts. The *space* is the number of cells of the worktape that ever were visited by the worktape head during the computation. Note that, for this machine, the space used can be much smaller than the input size (and we shall see complexity classes defined by placing such a "small" bound on the space usage allowed). The running time, on the other hand, must be at least as large as the input size unless answers depend only on some initial substring of the input.

Nondeterministic Turing machines (NDTMs)

This is a variant in which at each step the Turing machine has several choices as to its next move. The set of all possible computations can thus be viewed as a tree, with each reachable configuration of the Turing machine and its tapes having as its children those configurations that can be reached from it in one legal move. This machine can only yield the answers "yes" and "no", and hence is applicable only to decision problems. It answers "yes" if the tree of reachable configurations contains *any* configuration in which the machine is halted and the output tape contains the string representing yes. The answer is "no" if no such configuration is reachable. For the kinds of problems and algorithms in which we are interested, we may assume that the machine is designed in such a way that the tree is finite and all halting configurations (leaves) are at the same depth in the tree. In this case, the time used is the depth of the tree, and the space is the maximum, over all configurations in the tree, of the number of worktape cells in use.

Alternating Turing machines (ATMs)

This variant, proposed in [52], is again restricted to decision problems. In it, we again have a finite tree of computations with all halting configurations at the same level. Now, however, each configuration in the tree is labelled as either *universal* (∀) or as *existential* (∃). We inductively determine if the answer is "yes" as follows: A halting configuration is a yes-configuration if the output tape contains the string representing yes. A nonhalting ∃-configuration is a yes-configuration if at least one of its children is a yes-configuration. A nonhalting ∀-configuration is a yes-configuration if *all* its children are yes-configurations. The answer is "yes" if and only if the initial configuration is a yes-configuration.

The resources of time and space are defind as for NDTMs, but now we have an additional resource, which we call "alternations". This is the maximum, over all paths from an initial configuration to a final one, of the number of times a configuration has a different label from its parent (the root has no parent, but is assumed to contribute the first alternation). Thus, for instance, NDTMs can be viewed as ATMs with one alternation (and all configurations labelled by ∃).

Turing machines with other acceptance criteria

A variety of other output conventions besides those given above are possible for Turing machines that produce computation trees. A *probabilistic Turing machine* yields answer "yes" if and only if more than half the halting configurations are accepting configurations. A *random Turing machine* says "yes" if at least half the halting configurations are yes-configurations, says "no" if none of the halting configurations are yes-configurations, and otherwise says nothing. (It solves a decision problem only if it always says something, and that "something" is the correct answer.) A *counting Turing machine* yields as answer the *number* of halting configurations that are yes-configurations (and hence is applicable only to counting problems). Still other options are possible, but these will be discussed in the context of the complexity classes they are used to define (see Section 4).

Oracle Turing machines (OTMs)

These are Turing machines (DTMs, NDTMs, ATMs) with an additional semi-infinite "oracle tape", which alternates between write-only and read-only modes. Associated with any oracle machine is a particular problem (string relation) Y which the oracle can solve for free. While the oracle tape is in write-only mode, the DTM can at any time enter a query state, in which case at the next step the contents y of the oracle tape will be automatically replaced by a string b such that $(y, b) \in Y$, and the tape will become read-only. (Once the result has been read, the machine can in one step erase the oracle tape and return it to write-only mode.) Querying the oracle is thus like invoking a subroutine for solving Y, except that we do not count the time required by the subroutine.

Time and space requirements are the same as before. There are technical questions having to do with whether the number of cells used on the oracle tape should be counted (as with input and output tapes, this number can greatly exceed the number needed for the worktape). Unless otherwise stated, we shall assume that oracle tape space *is* counted in determining total space usage. One additional resource we may wish to count is oracle queries, the maximum, over all computation paths, of the number of times the machine enters its query state.

Parallel random access machines (PRAMs)

For a general introduction to random access machines and parallel random access machines, see [153, 246]. We need only consider the parallel version here, first introduced in [82]. We shall assume that the machine, in addition to having an arbitrary number of processors, is equipped with three sets of memory registers, one of read-only cells for input, one of write-only cells for output, and one of read-write memory cells for work. The input-output requirements are analogous to those for DTMs. For the sake of specificity, we assume that PRAM programs satisfy the concurrent-read, exclusive-write (CREW) restriction. (Two processors can read the contents of the same cell at the same time, but they cannot both write something in the cell at the same time.)

The space resource is measured in the standard way: the memory cells are indexed in sequence $c_1, c_2, \ldots$, and the space used equals the maximum k such that cell c_k is accessed by some processor during the computation. Time is a bit more complicated. We assume that all the processors work synchronously, one step at a time. The time for a given step is the maximum, over all processors, of the "semilogarithmic" cost measure for the operation performed by that processor. (The semilogarithmic cost for an operation that places a number N in register i when the input is of size n is $\lceil 1 + (\log N + \log i)/\log n \rceil$). The third resource is the number of processors used. (The familiar "random access machine" or RAM is simply a PRAM wth only one processor.)

Families of Boolean circuits

Again, for a general introduction, see [153]. A family F of Boolean circuits is an infinite collection of acyclic Boolean circuits $\{B_1, B_2, \ldots\}$, made up of AND, OR, and NOT gates. Such families are applicable only to problems X satisfying the property that for any two pairs $(x, a), (y, b)$ in X, $|x| = |y|$ implies $|a| = |b|$. (Decision problems obey this requirement, and most other search problems can be reformulated as string relations satisfying that property.) In a family that solves a problem X, circuit B_n has an ordered sequence of n inputs and an ordered sequence of outputs whose number equals the length of the answers for instances of size n. Given the bits of a string x of length n as input, it produces the bits of an answer for x at its outputs.

Families of circuits constitute what is called a "nonuniform" method for solving a problem. There need be no commonality between the circuits, no way to build circuit B_n given n. A *uniform* circuit family is one in which B_n can be generated automatically, given n (say by an appropriately resource-bounded DTM).

As with PRAMs, circuit families can be viewed as a model for parallel computation. For circuit families (uniform or nonuniform), the resource analogous to (parallel) time is *depth*, the number of gates in the longest path from an input to an output in a circuit. The resource analogous to sequential time is *size*, the total number of gates in the circuit. The resource most closely analogous to space is *width*. To define this, we assign levels to all the gates (with input gates having level 0 and each other gate having a level one greater than the maximum level among those gates that provide its input). We call a gate "live at level i" if the level of the gate is i or less, and an output wire from the gate is an input wire for some gate at level greater than i. The width of a circuit is the maximum, over all i, of the number of gates (other than input gates) that are live at level i. Two final resources, these having no obvious correlate among the Turing machine resources, are *fan-in*, the maximum number of inputs an AND or OR gate can have, and *fan-out*, the maximum number of outputs a gate can have.

1.4. *Resource bounds*

In the previous section, we saw a wide variety of resources. In restricting these resources, we shall normally choose from a short list of types of bounds:

- *Constant*: there exists a constant k such that $R_M(n) \leqslant k$ for all $n \geqslant 0$.
- *Logarithmic*: $R_M(n) = O(\log n)$.
- *Polylogarithmic*: there exists a constant k such that $R_M(n) = O(\log^k n)$.
- *Linear*: $R_M(n) = O(n)$.
- *Polynomial*: there exists a constant k such that $R_M(n) = O(n^k)$.
- *Exponential*: there exists a constant k such that $R_M(n) = O(2^{n^k})$.
- *Unbounded*: no constant at all is imposed on $R_M(n)$, beyond the fact that only a finite amount of resource R is used for any particular instance.

Note that these alternatives do not allow us to make as precise distinctions as are theoretically possible. In particular, the following "hierarchy" theorems show that very fine distinctions can be made. We state them here since, as a corollary, they also imply that each of the above restrictions is stronger than its successors, at least as far as DTM and NDTM space and time are concerned.

H1. THEOREM (Hartmanis and Stearns [113]). *If $F_1(n)$ and $F_2(n)$ are "time constructible functions" and if*

$$\liminf_{n \to \infty} \frac{F_1(n) \log_2(F_2(n))}{F_2(n)} = 0,$$

then there exists a language L that can be recognized by a DTM in time bounded by $F_2(n)$, but not by any DTM with time bounded by $F_1(n)$.

H2. THEOREM (Hartmanis, Lewis and Stearns [112]). *If $F_1(n)$ and $F_2(n)$ are "space constructible functions" with $F_2(n) \geqslant \log_2 n$ for all $n \geqslant 1$, and if*

$$\liminf_{n \to \infty} \frac{F_1(n)}{F_2(n)} = 0,$$

then there exists a language L that can be recognized by a DTM in space bounded by $F_2(n)$, but not by any DTM with space bounded by $F_1(n)$.

H3. THEOREM (Seiferas, Fischer and Meyer [223]). *If $F_1(n)$ and $F_2(n)$ are "time constructible functions" and if*

$$\liminf_{n \to \infty} \frac{F_1(n)}{F_2(n)} = 0,$$

then there exists a language L that can be recognized by an NDTM with time complexity bounded by $F_2(n)$, but not by any NDTM with time complexity bounded by $F_1(n)$.

(Separation for NDTM space classes, at least the coarse separation we require, follows from Theorem H2 and a result of [216] that we shall discuss in more detail in Section 2.6. More refined separations of NDTM space classes can be found, for example in [222].)

The reason we settle for coarse distinctions, rather than the fine ones offered by Theorems H1 through H3, is that, currently, coarse distinctions are the only ones we know how to make. (And even when we make them, their validity may depend on unproven conjectures.) Theorems H1 through H3 are proved using diagonalization arguments that construct problems with the desired properties, but give no insight into how a natural problem might be shown to have them. Thus such important algorithmic questions as whether, for example, the bipartite matching problem can be solved in time $O(n^2)$ as well as $O(n^{2.5})$ remain out of reach.

Fortunately, there is a theoretical advantage to the consideration of such coarse bounds: They often free us from concern about the precise details of our models of computation. Similarly, they allow us much latitude in the way we represent problems as string relations, as we shall see in the next section.

1.5. *A first example: the class P (and the class FP)*

As an illustrative example, let us consider the perhaps most famous of all complexity classes, the class "P" of decision problems solvable (or languages recognizable) by DTMs obeying a polynomial bound on running time. Informally, one often sees "P" used to refer to the class of *all* search problems solvable in polynomial time, but we shall use the notation "FP" to denote this more general class. This is not a standard usage (there is none), but will help clarify the issues in what follows. By the simulation results reported in [246], we know that the class P (and the class FP) will not be altered if we add extra worktapes to our machine model, or even if we replace the DTM by a RAM. Furthermore, on the assumption that any "reasonable" representation of a problem as a string relation can be translated into any other in polynomial time, P (and FP) can be viewed as representation-independent. (For example, note that the adjacency list representation for a graph can be translated into an adjacency matrix representation in $O(n^2)$ time if n is the length of the former representation.)

The significance of P (and FP) was first pointed out in [56, 72]. In particular, Edmonds was the first to propose polynomial-time solvability as a theoretical equivalent to the informal notion of "efficiently solvable", an identification that has held up over the years, despite its obvious exceptions. (For more on the motivation of this identification and its drawbacks, see for example [87, 135].)

A wide range of problems are known to be in P (and FP), and researchers are continually attempting to identify more members. Perhaps the most significant addition to the membership list in the 1980s is LINEAR PROGRAMMING [149, 157]. This is formulated as a decision problem as follows.

LINEAR PROGRAMMING

Instance: Integer valued vectors $V_i = (v_i[1], \ldots, v_i[n])$, $1 \leqslant i \leqslant m$, and $D = (d[1], \ldots, d[m])$, and $C = (c[1], \ldots, c[n])$, and an integer B.

Answer: "Yes" if and only if there is a vector $X = (x[1], \ldots, x[n])$ of rational numbers such that $C \cdot X \geqslant B$ and $V_i \cdot X \leqslant d[i]$ for all i, $1 \leqslant i \leqslant m$.

LINEAR PROGRAMMING is of particular significance for P, for, in a sense to be explained in Section 1.7, it is an example of the *hardest* kind of problem in the class.

1.6. *Reductions*

In the main body of this chapter, we shall be defining complexity classes and illustrating them by giving examples of problems they do and do not contain. To show that a particular problem is in a given class, we need only exhibit a "method" for solving the problem that meets the requirements of the class's definition (as to machine model and resource bounds). Proofs of non-membership must be more indirect. One key technique, however, is applicable to both sorts of proof: the "reduction".

One common programming trick that can be incorporated into most of our models of computation is the "subroutine". For instance, an algorithm for solving the network flow problem (see [247]) might proceed by solving a sequence of shortest path problems, i.e., by using a *subroutine* for the shortest path problem. If we restrict ourselves to Turing machine models, the concept of subroutine can be formalized in terms of oracles. Suppose for example that we have a polynomial-time deterministic oracle Turing machine that solves problem X under the assumption that the oracle solves problem Y. In this case, the oracle acts just like a subroutine for solving Y. If it should turn out that problem Y is in P, one could replace the oracle by an appropriate version of a DTM program for solving Y (the subroutine), thus obtaining a polynomial-time oracle-free DTM that solves X. (We use here the fact that if p and q are polynomials, then so is the composition $p \circ q$ of p and q.)

DEFINITION. If X and Y are problems (string relations), a (*Turing*) *reduction* from X to Y is any OTM that solves X given an oracle for Y.

When restricting attention to decision problems, we often do not need to use the full power of this definiton, with its ability to ask an unbounded number of queries. Indeed, the following much restricted version often suffices.

DEFINITION. If X and Y are decision problems, a *transformation* from X to Y is a (DTM-computable) function $f: \{0,1\}^* \to \{0,1\}^*$ such that x has answer "yes" under X if and only if $f(x)$ has answer "yes" under Y.

Note that this is the same thing as a Turing machine reduction in which the OTM can ask but one query of its oracle, and must give as is own answer the answer it receives from the oracle. This type of reduction is also called a "many-one" reduction in the literature, given that it can conceivably map many x to the same string y.

As indicated above, reductions become useful tools in dealing with complexity classes when they themselves obey resource bounds. If R is a restricted class of reduction, let us write "$X \leqslant_R Y$" to signify that there is a reduction in R from X to Y.

Definition. If C is a complexity class and R is a class of resource bounded reductions, we say that R is *compatible* with C if for any problems X and Y, $X \leqslant_R Y$ and $Y \in C$ imply that $X \in C$.

For the class P, the most general compatible class of reductions that is commonly considered consists of Turing reductions that obey polynomial time bounds (*polynomial-time Turing reductions*, sometimes abbreviated as "$\leqslant_T$" reductions). Also compatible, of course, is the more restricted class of *polynomial transformations* (transformations f with $f \in \mathrm{FP}$, abbreviated as "$\leqslant_p$" reductions). Another popular class of reductions is obtained by a further restriction, this time to *log-space transformations* (or "$\leqslant_{\text{log-space}}$" reductions), i.e., those transformations that are computable in logarithmic space. (Note that such transformations are also polynomial transformations, since the logarithmic bound on workspace means that there are only a polynomial number of distinct states for the worktape). In addition to being compatible with P, these three classes of reductions all have one more important property: they are "transitive".

Definition. A class R of reductions is *transitive* if for all problems A, B, and C, $A \leqslant_R B$ and $B \leqslant_R C$ together imply $A \leqslant_R C$.

(Proving that log-space transformations are transitive is an interesting exercise, as we do not necessarily have enough space on our logarithmic worktape to write down the entire output of the first transformation, and so must generate it bit by bit, as needed.)

In Section 4.4, we shall introduce some additional forms of reduction, more complex than those given here, such as "γ-reductions" and "random" reductions of various types. In the meantime, however, the ones given here will suffice for our discussions.

1.7. Completeness

Reductions can clearly simplify the job of proving membership in a class. More importantly, however, they can also be used to prove non-membership. Suppose we know that problem X is not in class C. Then if we can exhibit a reduction from X to Y that is compatible with C, we will know that Y cannot be in C either, by the definition of "compatible". The difficult task, of course, is finding that first problem X that is not in C. Reduction can help here too.

We know by Theorems H1 and H3 of Section 1.4 that certain class containments $C' \subseteq C$ are proper, i.e., although every X in C' is in C, there is an X in C that is not in C'. As already pointed out, the diagonalization arguments used to prove H1 through H3 unfortunately do not make examples of problems in $C - C'$ readily available to us. Suppose, however, that we could show, for some class R of reductions compatible with

C', that *all* problems in C were reducible to a given problem X. Then if X were in C', we would have $C \subseteq C'$, a contradiction. This motivates the following definition.

DEFINITION. Suppose X is a problem, C is a class of problems, and R is a class of reductions. If $Y \leqslant_R X$ for all $Y \in C$, then we say that X is *hard* for C (*under R-reductions*), or simply *R-hard for C*. If also $X \in C$, then we say that X is *complete for C* (*under R-reductions*), or *R-complete for C*.

Note that if X is R-complete for a class C, it can be viewed as typical of the "hardest" problems in C (at least from the viewpoint of any class $C' \subseteq C$ with which R is compatible). As an example, consider the LINEAR PROGRAMMING problem introduced in Section 1.6. When we said there that this was in a sense the "hardest problem" in P, we were referring to the fact that LINEAR PROGRAMMING is complete for P under log-space reductions ("log-space complete for P") [68]. Log-space reductions are compatible not only with P, but also with the class $\mathrm{L} \subseteq \mathrm{P}$ of decision problems solvable by DTMs obeying a logarithmic space bound. Since LINEAR PROGRAMMING is complete for P under a reduction that is compatible with L, we conclude that it is in L if and only if $\mathrm{P} = \mathrm{L}$.

The situation here is somewhat different from that envisioned above, however. Although we strongly suspect that L is properly contained in P, we do not yet know for sure that this is the case. Thus we cannot at this point prove that a problem is in $\mathrm{P} - \mathrm{L}$ simply by showing that it is log-space complete for P; all that we can currently conclude from such a result is that the problem is in $\mathrm{P} - \mathrm{L}$ unless a widely believed conjecture is false. Such conditional complexity results abound in the current theory of complexity classes, as so many of the relations between classes are still unresolved.

There remains the question of how we show a problem X to be hard for a class under a given type of reduction. Given that the classes of interest are all of infinite size, we clearly cannot exhibit distinct reductions to X from each member. For many of our complexity classes, however, there will be one "generic" problem for which all these reductions follow almost by definition. In the case of the class P and log-space transformations, that problem is as follows.

DTM ACCEPTANCE
Instance: Description of a DTM M, string x, and an integer n written in unary.
Answer: "Yes" if and only if M, when started with input x, halts with answer "yes" in n or fewer steps.

Note that if a problem Y is in P, there must be a DTM M that accepts the set of strings representing its yes-instances, and integers c and k such that M never takes more than $c|y|^k$ steps on any input y. If one has such a c, k, and M in mind, one can, given any instance y of problem Y, construct an instance of DTM ACCEPTANCE that has the same answer using only logarithmic space. Simply write down the description of M (constant effort, independent of y), copy y (requires constant workspace), and compute $n = c|y|^k$ (logarithmic space). Thus DTM ACCEPTANCE is trivially log-space hard for P.

In fact, DTM ACCEPTANCE is log-space *complete* for P, as it is easy to see that it is itself in P: we can simply simulate the running of the Turing machine M on the input x, and the number of steps n that we have to simulate is polynomially bounded in the size of the input. (Note that here our requirement that n be written in unary notation is crucial; if n were written in the more standard binary notation, the simulation might take exponential time in the "size" of the input. Representing numbers in unary fashion is not "reasonable" in any ordinary sense, but it has its technical uses, and we shall see more of them in Section 2.1.)

Once we have our first R-complete problem Y for a class C, obtaining others is conceptually much more straightforward, assuming our class of reductions is transitive. All we need do to show that problem X is complete is prove $Y \leqslant_R X$; the transitivity of R does the rest. In the descriptions of complexity classes that follow, we shall when possible give examples of complete problems for each that are a bit more interesting than the generic one. We shall not, however, give the reductions used to prove completeness. Readers unfamiliar with the techniques involved in constructing such reductions are referred to [87] or any of the standard textbooks on algorithms and complexity, such as [5, 192].

1.8. *Relativized worlds*

We will be discussing many open problems concerning the relationship between classes in what follows. One common measure of the potential obstacles to resolving such problems is derived by considering how they "relativize". If X is a class defined in a particular way, and A is any subset of $\{0, 1\}^*$, then X^A is the analogous class defined using the same resource bounds but augmenting the machine model with a (perhaps additional) oracle tape for asking questions about membership in A. For instance, P^A would be the set of all languages that can be recognized in polynomial time by an oracle Turing machine with oracle A.

The question of whether class X equals class Y can usually be judged "difficult" if it relativizes both ways, i.e., if there exists an oracle A such that $X^A = Y^A$ and an oracle B such that $X^B \neq Y^B$. This is because the standard proof techniques for proving classes equal or unequal, such as simulation and diagonalization, continue to work even if relativized. If X is shown to equal Y by one of these techniques, then X^A will equal Y^A for all A, and similarly if we can use them to show $X \neq Y$, then we will have $X^A \neq Y^A$ for all A. Hence if a question relativizes both ways, the standard techniques cannot be applied to it. Unfortunately, most of the interesting open questions have this property.

A few closed problems also are known to have the property, but these tend to be of three well understood basic types, and do not appear relevant to the types of open questions we shall be discussing:

(1) Questions defined in terms of specially constructed oracles, so that their "relativized" versions deal with *doubly* relativized classes (i.e., classes defined in terms of machines with two separate oracle tapes and two separate oracles). For example, one can construct languages A, B, and C such that $\mathrm{P}^A = \mathrm{NP}^A$, and yet both $(\mathrm{P}^A)^B = (\mathrm{NP}^A)^B$, and $(\mathrm{P}^A)^C \neq (\mathrm{NP}^A)^C$ [104, 126].

(2) Problems discussing running time distinctions that are so fine as to be machine-dependent, such as the result of [197] concerning deterministic and nondeterministic linear time on a particular Turing machine model.

(3) Problems concerning sublinear space bounds, when the oracle tape is not required to obey the space bound. These last will be discussed further in Section 5.1, but see also [107].

Sometimes, even though a particular open question relativizes both ways, there is an additional property that at least hints at what the correct answer might be. Each oracle set A can be viewed as establishing an "alternate universe", with its own properties and relation between classes. For some questions, even though there are alternate universes where each possible outcome occurs, one of those outcomes is vastly preferred. More precisely, if one takes the natural measure over the space of all subsets of $\{0, 1\}^*$, the set of all oracles A for which a given outcome occurs has measure 1, i.e., the outcome occurs for almost all A. We then say that the given outcome holds "for a random oracle". There is unfortunately no a priori reason why this should imply that the outcome occurs for the particular oracle $A=\emptyset$, i.e., the one which yields the base case and hence the world in which we are normally most interested. (A "random oracle hypothesis", asserting that if a property holds for a random oracle then it holds for the empty oracle, was wishfully proposed in [29], but counterexamples, albeit technical ones, were quickly found [165].) Nevertheless, such results at least add an extra aura of believability to conjectures to which we already subscribe for other reasons, and we shall mention them where they are known. (For more on relativization, see Section 6.1.)

1.9. The organization of the catalog

The next four sections constitute a "catalog" of complexity classes. We have attempted to be moderately complete, covering all the major and many of the minor classes. Inevitably, however, some of the more obscure classes had to be omitted for space reasons. Normally, however, papers that discuss such classes will begin by relating them to more famous classes, so even in such cases the current survey may help in putting things in perspective. In discussing each class, we try to summarize some of the most important and interesting results about its structure and its relation to other classes. Here our choice of what to present is of necessity somewhat more selective, but we have attempted to present those results that the non-specialist might find the most intriguing.

Note that while the presentation must be sequential, the relationships between classes are not. Thus the presentation will unavoidably contain many pointers back and forth. If a property of class X does not make sense until we have discussed class Y, it cannot be fully explained until that later class has been introduced, and hence can only be alluded to by a forward reference when X is discussed. Our intention is that the non-expert should be able to read the presentation straight through as a tutorial, and subsequently be able to use it as a reference, with the aid of the tables and figures in Section 6 and the pointers in the text.

One additional note about the presentation. It is not our purpose here to present an historical picture of the development of the field, but rather to survey the current "state of the art" for complexity classes. We shall, of course, attempt to cite all the fundamental papers in the field. There will be cases, however, where a paper that was instrumental in pointing the way to a particular result will be ignored in favor of the paper that actually first presented the result itself. For readers interested in the development of ideas, the papers we do cite should provide an adequate point of departure for the needed literature search. Other general surveys and tutorials related to the subject of complexity classes can be found in [20, 87, 224, 236].

2. Presumably intractable problems

2.1. *The class NP, NP-complete problems, and structural issues*

The class "NP" is defined to be the set of all decision problems solvable (languages recognizable) by NDTMs in polynomial-bounded time. It trivially contains P, since DTMs are special cases of NDTMs. It would appear to contain much more, however, as nondeterminism seems to add significant power to time-bounded computations.

To see this, it is perhaps easier to consider a more restricted, though no less powerful version of the polynomial-time NDTM: the polynomial time "guess-and-check" procedure for verifying yes-answers (language membership). On input x, such a procedure first (nondeterministically) guesses a string y such that $|y| \leqslant p(|x|)$ for some fixed polynomial p, and then runs a polynomial-time algorithm on input (x, y). The answer is "yes" (x is in L) if and only if there is a guessable y such that the algorithm will answer "yes". It is easy to see that a decision problem (language) is in NP if and only if such a scheme exists.

Moreover, it is easy to see that a wide variety of seemingly intractable problems are susceptible to polynomial-time guess-and-check procedures, with the "guess" offering a shortcut to what would otherwise seem to be an unavoidably exponential-time exhaustive search. For instance, consider the following famous problem.

SATISFIABILITY

Instance: List of *literals* $U = (u_1, \bar{u}_1, u_2, \bar{u}_2, \ldots, u_n, \bar{u}_n)$, sequence of *clauses* $C = (c_1, c_2, \ldots, c_m)$, where each clause c_i is a subset of U.

Answer: "Yes" if there is a *truth assignment* for the variables $u_1, \ldots, u_n$ that satisfies all the clauses in C, i.e., a subset $U' \subseteq U$ such that $|U' \cap \{u_i, \bar{u}_i\}| = 1$, $1 \leqslant i \leqslant n$, and such that $|U' \cap c_i| \geqslant 1$, $1 \leqslant i \leqslant m$.

Note that there are 2^n possible truth assignments, and at present we know of no foolproof way to determine in subexponential time whether there exists one that satisfies all the clauses. A guess-and-check algorithm, however, need only guess a satisfying truth assignment (if one exists); verifying that all the clauses are satisfied is then trivial.

A second example is the following problem.

CLIQUE

Instance: Graph $G = (V, E)$, positive integer K.

Answer: "Yes" if G contains a complete subgraph of size K, i.e., if there is a subset $V' \subseteq V$ with $|V| = K$ such that for all $u, v \in V', u \neq v$, the pair $\{u, v\}$ is an edge in E.

Note that there are roughly $|V|^K$ possible subsets, and at present we know of no way to determine if one with the desired property exists in subexponential time. A guess-and-check algorithm, however, need only guess V' and verify that all the required edges are present in E.

A third example is the following.

TRAVELLING SALESMAN PROBLEM (TSP)

Instance: List $c_1, c_2, \ldots, c_n$ of *cities*, a positive integer *distance* $d(c_i, c_j) = d(c_j, c_i)$ for each pair $\{c_i, c_j\}$ of distinct cities, and an integer bound B.

Answer: "Yes" if there is a *tour* through all the cities of total length B or less, i.e., a permutation π of indices $1, \ldots, n$ such that

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}) \leqslant B.$$

Note that there are $n! = 2^{\Theta(n \log n)}$ possible tours, and again there is no known way of finding the answer deterministically in subexponential time. A guess-and-check algorithm, however, need only guess the desired tour and verify its length.

These three examples, and many more like them, have led most theoretical computer scientists to the belief that P is a strict subclass of NP, i.e., that $\mathrm{P} \neq \mathrm{NP}$. Moreover, although the question of P versus NP relativizes both ways (there exist oracles A and B such that $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$ [17]), the two classes are unequal for a "random" oracle [29].

Given the likelihood that $\mathrm{P} \neq \mathrm{NP}$, there can be considerable practical significance in identifying a problem as "complete" for NP under an appropriate notion of reducibility. If the reducibility is compatible with P in the sense described in Section 1.6, such problems will be polynomial-time solvable if and only if $\mathrm{P} = \mathrm{NP}$. Such a result thus provides strong theoretical support for the belief that a problem is intractable, and can direct us to more productive approaches to the problem, such as settling for near-optimal rather than optimal solutions in the case of optimization problems.

Section 1.6 listed three types of reduction that were compatible with P. In order of apparently decreasing power, they were polynomial-time Turing reductions, polynomial transformations, and log-space transformations. By tradition, we reserve the unadorned adjective "NP-complete" for just one of these:

DEFINITION. A problem (language) is NP-*complete* if it is complete for NP under polynomial transformations.

If completeness under a different type of reduction is intended, most authors will normally point this out explicitly, as in "complete for NP under polynomial-time Turing reductions". It is not yet clear whether such distinctions are meaningful, however. Although polynomial-time Turing reductions are known to be more powerful than polynomial transformations for classes larger than NP [170], this distinction is not known to hold within NP. Indeed, it can only hold there if $\mathrm{P} \neq \mathrm{NP}$, which remains to be proved. Similary, polynomial-time transformations cannot be more powerful than log-space transformations unless $P \neq L$, another unproved conjecture. All three problems introduced above as members of NP are complete for NP under log-space reductions, the weakest of the three, and there are few, if any, examples of problems that are complete for NP but are not known to be complete under such reductions.

The first NP-complete problem to be identified (other than the generic problem of "NDTM ACCEPTANCE", defined analogously to the DTM ACCEPTANCE problem of Section 1.8) was SATISFIABILITY. The result that identified it is now called "Cook's Theorem" in honor of its author, Steven Cook.

1. THEOREM (Cook [58]). SATISFIABILITY *is* NP-*complete.*

In the same paper, Cook also proved that CLIQUE was NP-complete. (At roughly the same time, Leonid Levin in the Soviet Union independently came up with an idea equivalent to NP-completeness and proved a version of Theorem 1 in which SATISFIABILITY was replaced by a variant on the problem of "TILING" that we will cover in Section 3.2 [174].) Shortly thereafter, the class of known NP-complete problems was expanded to include the TSP and a wide variety of other problems by Richard Karp in the landmark paper [150] (see also [151]). By 1979, over 300 problems could be listed [87]. NP-complete problems are now known to permeate all areas of computer science, operations research, and mathematics, and are not unknown in such disparate areas as biology, physics, and political science. Readers are referred to the above book to obtain a better feel for the variety of results that have been obtained, as well as a much more thorough treatment of the subject and its history. More recent surveys can be found in the current author's "*NP-completeness Column*", starting with the first edition in [129].

An interesting observation is that it often takes very little to change a problem from polynomial-time solvable to NP-complete. Table 1 lists several pairs of such related problems, one in P and one NP-complete. (All the NP-completeness results in Table 1 are from [150], except the result for QUADRATIC DIOPHANTINE EQUATIONS, which is from [182].

The distinction illustrated in the last row of Table 1 is worthy of special note. Many a false proof that $P = NP$ has been based on the mistaken impression that UNARY PARTITION is NP-complete, whereas in fact it can be solved in polynomial time by a straightforward dynamic programming algorithm. Such an algorithm requires time exponential in the input size for BINARY PARTITION, even though it is polynomial in the perhaps exponentially larger) size for the corresponding UNARY PARTITION. This distinction is worth formalizing.

DEFINITION. A *pseudopolynomial-time* algorithm is one whose running time would be polynomial if all input numbers were expressed in unary notation.

The existence of a pseudopolynomial-time algorithm for a given NP-complete problem may mean that the problem is not so "intractable" after all. (Pseudopolynomial time becomes polynomial time if one restricts attention to instances in which the maximum number obeys a constant or polynomial bound in terms of the input size, and such restrictions may well hold in practice.) Thus it may be important to determine

Table 1
Problems on the frontier

POLYNOMIAL TIME	NP-COMPLETE
EDGE COVER *Instance*: Graph $G=(V,E)$, integer k. *Answer*: Yes if there is a subset $E' \subseteq E$ with $\lvert E'\rvert \leqslant k$ such that every vertex is the endpoint of an edge in E'.	VERTEX COVER *Instance*: Graph $G=(V,E)$, integer k. *Answer*: Yes if there is a subset $V' \subseteq V$ with $\lvert V'\rvert \leqslant k$ such that every edge has an endpoint in V'.
FEEDBACK EDGE SET *Instance*: Graph $G=(V,E)$ integer k. *Answer*: Yes if there is a subset $E' \subseteq E$ with $\lvert E'\rvert \leqslant k$ such that every cycle in G contains an edge in E'.	FEEDBACK ARC SET *Instance*: Directed graph $G=(V,A)$ integer k. *Answer*: Yes if there is a subset $A' \subseteq A$ with $\lvert A'\rvert \leqslant k$ such that every (directed) cycle in G contains an arc in E'.
EULER CYCLE *Instance*: Graph $G=(V,E)$ with $m=\lvert E\rvert$. *Answer*: Yes if there is an ordering $e_1, e_2, \ldots, e_m$ such that e_1 and e_m share an endpoint, as do all pairs $\{e_i, e_{i+1}\}$, $1 \leqslant i < m$.	HAMILTONIAN CYCLE *Instance*: Graph $G=(V,E)$ with $n=\lvert V\rvert$. *Answer*: Yes if there is an ordering $v_1, v_2, \ldots, v_n$ such that $\{v_1, v_n\}$ forms an edge, as do all pairs $\{v_i, v_{i+1}\}$, $1 \leqslant i < n$.
2-SATISFIABILITY (2-SAT) *Instance*: Instance of SATISFIABILITY in which no clause contains more than 2 literals. *Answer*: Yes if the clauses are satisfiable.	3-SATISFIABILITY (3-SAT) *Instance*: Instance of SATISFIABILITY in which no clause contains more than 3 literals. *Answer*: Yes if the clauses are satisfiable.
LINEAR DIOPHANTINE EQUATIONS *Instance*: Positive integers a, b, and c *Answer*: Yes if there are positive integers x and y such that $ax+by=c$.	QUADRATIC DIOPHANTINE EQUATIONS *Instance*: Positive integers a, b, and c. *Answer*: Yes if there are positive integers x and y such that $ax^2+by=c$.
UNARY PARTITION *Instance*: Set $A=\{a_1, \ldots, a_n\}$ of integers written in unary notation. *Answer*: Yes if there is a subset $A' \subset A$ such that $\Sigma_{a\in A'} a = \Sigma_{a \in A-A'} a$.	BINARY PARTITION *Instance*: Set $A=\{a_1, \ldots, a_n\}$ of integers written in binary notation. *Answer*: Yes if there is a subset $A' \subset A$ such that $\Sigma_{a\in A'} a = \Sigma_{a \in A-A'} a$.

whether a problem is merely NP-complete in the ordinary sense, or whether it has the following stronger property.

DEFINITION. A problem is said to be NP-*complete in the strong sense* if the variant of it in which all input numbers are written in unary notation is NP-complete.

Note that problems like CLIQUE are trivially NP-complete in the strong sense, since the only number in the input is by definition bounded by the input size (in this case, the input bound K is bounded by the number of vertices $\lvert V\rvert$). A more meaningful example of a "strongly NP-complete" problem is the following variant of the "PARTITION" problems of Table 1.

3-PARTITION

Instance: Sequence a_i, $1 \leqslant i \leqslant 3n$, of positive integers (in binary notation).

Answer: "Yes" if there is a partition of these integers into disjoint 3-element sets A_j, $1 \leqslant j \leqslant n$, such that all n sets A_i have exactly the same sum.

This problem is proved NP-complete in the strong sense by exhibiting a polynomial p and showing that the problem remains NP-complete (in the ordinary sense) even if we require $a_i \leqslant p(n)$ for $1 \leqslant i \leqslant 3n$. (See [86] for the proof and a more rigorous development of these concepts.)

Distinctions like that provided by "strong NP-completeness" are only possible if one imposes some form of semantics on the language in NP, i.e., identifies certain parts of the input string as "numbers", "graphs", etc. There is much one can say, however, without at all considering what the languages in NP "mean", but by simply examining their structural, set-theoretic properties. A first question along these lines concerns what are called "sparse" languages, defined as follows.

DEFINITION. A language L is *sparse* if there is a polynomial p such that L contains no more than $p(n)$ strings of length n for all $n > 1$.

As simple examples of sparse languages, consider languages over a one-letter alphabet, i.e., subsets of $\{1\}^*$. These are sometimes called "tally languages". For such languages, there can be at most one string of length n for any n.

A natural "structural" question to ask is whether there can be a sparse NP-complete language. Unfortunately, this possibility seems ruled out. In [179] it was proved that a sparse language can be NP-complete under polynomial transformations only if P = NP. Weaker, but not more believable conclusions follow if a sparse language is complete for NP under polynomial-time Turing reductions [146, 179] (see Section 2.4). Indeed, there are interesting consequences if there is a sparse language anywhere in NP − P [111] (see Section 3.2). (For more detailed tutorials on results concerning sparsity, see [105, 180].

Thus, assuming P ≠ NP, all NP-complete sets have a certain gross structural similarity. Could the similarity actually be much closer? Could they, for instance, all be isomorphic?

DEFINITION. A polynomial transformation is a *polynomial-time isomorphism* if it is a one-one onto function and its inverse is also polynomial-time computable.

In [31] Berman and Hartmanis was conjectured that if a problem is NP-complete under polynomial-time transformations then it is polynomial-time isomorphic to SATISFIABILITY (and hence all NP-complete problems are isomorphic in this sense). Berman and Hartmanis were able to show that this was true for all the NP-complete problems known to them at the time, although of course this did not constitute a proof of the conjecture. (Indeed a complete proof would have implied that P ≠ NP: If P = NP, then all languages in NP would be NP-complete, including finite and infinite ones, and no finite language can be isomorphic to an infinite one.)

A further obstacle to proving this "isomorphism conjecture" is the fact that it may well be false even if $P \neq NP$. In particular, a class of specially constructed "k-creative" languages, introduced in [145, 258], seems to be rich source of candidates for NP-complete languages *not* isomorphic to the more normal NP-complete languages or to each other. In [166] it is shown that the desired nonisomorphic NP-complete k-creative languages exist so long as a particular type of polynomial-time "scrambling" function exists, and that, although no such function has yet been found, they do exist with respect to a random oracle. Thus the isomorphism conjecture fails with respect to a random oracle (and no oracle is yet known for which the conjecture succeeds; the best we have is an oracle for which the analogous conjecture succeeds for the larger class Σ_2^P, to be defined in Section 2.5 [120]).

In this light, it is interesting to note that if there exist nonisomorphic NP-complete languages, then there exist infinitely many distinct isomorphism classes, and these have a rich structure, as shown in [181]. A similar statement can be made about the structure of $NP - P$ under polynomial transformations, assuming $P \neq NP$. In [167], it is shown that under this assumption and these reductions, $NP - P$ must consist of an infinite collection of equivalence classes, of which the NP-complete problems are only one, albeit the hardest. Two famous problems have long been considered candidates for membership in such intermediate equivalence classes: GRAPH ISOMORPHISM (as described in Section 1.1) and COMPOSITE NUMBER (Given an integer n written in binary notation, are there positive integers p, q, $1 < p, q < n$, such that $n = pq$?). We shall have more to say about these two in later sections (COMPOSITE NUMBER in Sections 2.2 and 4.3, GRAPH ISOMORPHISM in Sections 2.5 ad 4.1). As we shall see, there are strong arguments why neither is likely to be NP-complete.

The equivalence class containing GRAPH ISOMORPHISM, in particular, has been so well-studied that a name for it (or rather its generalization to arbitrary search problems) has been introduced:

DEFINITION. The class of *GRAPH ISOMORPHISM-complete* problems consists of all those search problems X such that X and GRAPH ISOMORPHISM are polynomial-time Turing reducible to each other.

Included in this class are various special cases of graph isomorphism (e.g., GRAPH ISOMORPHISM restricted to regular graphs [38]), as well as related problems (e.g. "Given G, what is the order of its automorphism group?" [183]) and at first glance unrelated ones (e.g., a special case of CLIQUE [162]). These are to be contrasted with the many special cases of GRAPH ISOMORPHISM that are now known to be in P, for surveys of which, see for instance [87, 129].

We conclude this section with a more fundamental issue. So far, in considering the question of P versus NP, we have assumed that there were only two possible answers: $P = NP$ and $P \neq NP$. There is the possibility, however, that the question is *independent* of the standard proof systems on which we rely. This would mean that there exist models for both possible outcomes, with neither model offering an affront to our fundamental assumptions. This possibility was first raised explicitly in [109], which showed that for any formal system F there exists an oracle A such that $P^A = NP^A$ is independent of F.

The result as proved in [109] was limited, since it held only for a particular description A. That is, independence was proved only for the statement "$\mathrm{P}^{L(M)} = \mathrm{NP}^{L(M)}$", where M was a particular DTM and "$L(M)$" denotes the language accepted by M. (Moreover, in this case, $L(M)$ was in fact the empty set, and it was only the opaque choice for its representation that yielded the independence result.) Subsequently, however, the restriction to a particular representation has been removed. In particular, given F, there exists a recursive set A such that for *any* provably total DTM M that recognizes A, $\mathrm{P}^{L(M)} = \mathrm{NP}^{L(M)}$ is independent of F[103, 204]. (A DTM is "total" if it halts on all inputs.)

The question thus arises, could this more powerful conclusion hold for the empty oracle and for a proof system we commonly use, such as Peano Arithmetic? Certain computer science questions about the termination of programs in particular typed languages *have* been proved to be independent of Peano Arithmetic (even of second-order Peano Arithmetic) in [81]. It does not appear, however, that the techniques used could be applied to the P versus NP question. So far, the only results for P = NP have been consistency results for substantially weaker systems, as in [67]. Unfortunately, these systems are so weak that many standard, known results cannot be proved in them either [141, 142, 143, 173], and so independence of such systems tells us little. Moreover, it can be argued that even Peano Arithmetic is too weak a theory for independence of it to be significant, and that at the very least one would want the question to be independent of full Zermelo–Fraenkel set theory before one would be willing to give up the quest for an answer. Although such independence is believed unlikely, it is a possibility that cannot be totally ignored. We will nonetheless ignore it for the remainder of this chapter. Readers wishing more background on the issue are referred to [144]. For a detailed discussion of the logical theories mentioned above, see for instance [25].

2.2. *Co-NP, NP∩co-NP, and nondeterministic reducibilities*

One aspect of NP and the NP-complete problems that we have failed so far to emphasize is their one-sidedness. Consider the complementary problem to SATISFIABILITY, in which we are asked if it is the case that every truth assignment fails to satisfy at least one clause. This is simply SATISFIABILITY with the answers "yes" and "no" reversed, and hence in practical terms is computationally equivalent to SATISFIABILITY. However, it is not at all clear that this "co-SATISFIABILITY" problem is NP-complete or even in NP. (What would be a "short proof" that all truth assignments have the desired property?)

In general, if we let "co-NP" denote the set of all languages $\{0, 1\}^* - L$, where $L \in \mathrm{NP}$, we are left with the open question of whether NP = co-NP, and the suspicion that it is not. This suspicion is supported by the fact that the two classes are distinct for a random oracle [29], although there do exist oracles A such that $\mathrm{NP}^A = \text{co-NP}^A$ even though $\mathrm{P}^A \neq \mathrm{NP}^A$ [17], and so the question seems unlikely to be resolved soon. Note that the two classes *must* be identical if P = NP, and indeed that P = NP if and only if P = co-NP. Note also that NP must equal co-NP if any NP-complete problem, such as SATISFIABILITY, should prove to be in co-NP, and that if NP ≠ co-NP, then neither set can contain the other.

Assuming that NP and co-NP are distinct, one wonders if there is any structural, rather than computational, way in which they differ. If one takes "structural property" in a suitably broad sense, there is at least one such difference, albeit a technical one. In [111] it is shown that there is an oracle for which co-NP $-$ P contains a sparse language but no tally (i.e., one-symbol) languages, in contrast to the fact that for all oracles, NP $-$ P contains a sparse language if and only if it contains a tally language.

A more important consequence of the presumed inequality of NP and co-NP is that it would provide us with yet another potentially interesting class, NP$\cap$co-NP. This class would be a proper subclass of both and would contain P, although not necessarily properly. The question then becomes, does P $=$ NP$\cap$co-NP? This question relativizes both ways for oracles that yield P $\neq$ NP [17], although it is not currently known whether either result is preferred by a random oracle (inequality holds with probability 1 for a "random permutation") [29]. The one current candidate of importance for NP$\cap$co-NP $-$ P is the above-mentioned COMPOSITE NUMBER problem. As remarked in the previous secton, this is in NP. By a result of [199] that there exist short proofs of primality, it is also in co-NP. Unfortunately, the status of COMPOSITE NUMBER as a candidate is somewhat weakened by the fact that many researchers think it is in P, and that it is only a matter of time before the required polynomial-time algorithm is discovered. For instance, it is shown in [188] that COMPOSITE NUMBER is in P if the Extended Riemann Hypothesis is true. A better candidate would be a problem that is complete for NP$\cap$co-NP, but unfortunately, no such problem is known. Indeed, it is strongly expected that none exists, since completeness results proved by known methods relativize and there exist oracles A such that NP$^A\cap$co-NP^A has no complete sets under polynomial transformations [228]. Moreover, it will be no easier finding complete sets under the more general notion of polynomial-time Turing reducibility: it is shown in [110] that either both types of complete sets exist, or neither.

The class NP$\cap$co-NP may still have its uses, however. Consider the concept of "γ-reducibility," introduced in [3]. This reducibility, while not necessarily compatible with P, is compatible with NP$\cap$co-NP. Thus, although a problem that is complete for NP under γ-reductions could conceivably be in P even if P $\neq$ NP, it could not be in P (or even in co-NP) if NP $\neq$ co-NP. A γ-reduction is an example of a "nondeterministic" reduction, in that it is defined in terms of an NDTM rather than a DTM:

DEFINITION. A language X is *γ-reducible* to a problem Y if there is a polynomial-time NDTM M such that, on any input string x, M has at least one accepting computation, and such that, if y is the contents of the tape at the end of any such accepting computation, then $x \in X$ if and only if $y \in Y$.

Several problems are proved to be complete for NP under γ-reductions in [3], none of which has yet been proved NP-complete under the more restrictive reductions of the previous section. All were of a number-theoretic nature, with a typical one being LINEAR DIVISIBILITY (Given positive integers a and c in binary representation, is there a positive integer x such that $ax+1$ divides c?).

This notion of nondeterministic reduction was modified and generalized in [177] to what is called "strong nondeterministic polynomial-time Turing reducibility".

Definition. A language X is *strongly nondeterministically polynomial-time Turing reducible* to a problem Y if there is a nondeterministic polynomial-time OTM M with Y as oracle such that

(i) all computations halt with one of the three outcomes {yes, no, don't know},
(ii) for any string x, M has at least one computation that yields the correct answer, and
(iii) all computations that do not result in "don't know" yield the correct answer.

If a problem Y is complete for NP under this more general notion of reducibility, it can be shown that $Y \in$ co-NP if and only if NP = co-NP, the same conclusion that holds for γ-reducibility. In [55] strong nondeterministic Turing reducibility is used to prove the (presumed) intractability of a broad class of network testing problems, only a few of which are currently known to be NP-complete.

If one drops from the above definition the requirement that (ii) hold for non-members of X, one obtains (ordinary) polynomial-time nondeterministic Turing reductions, as introduced in [187]. Although these do not have the nice sorts of consequences discussed here, they have their uses, e.g., see [17, 170, 187]. We shall be introducing still further types of reducibilities and of "completeness" for NP once we introduce the concept of "randomized" computation in Section 4.

2.3. *NP-hard, NP-easy, and NP-equivalent problems (the class $F\Delta_2^P$)*

In the previous section we mentioned that NP was limited as a class because of its one-sided nature. A second apparent limitation is due to the fact that NP is restricted to decision problems, whereas in practice the problems that interest us are often function computations or more general search problems. For instance, the TRAVELLING SALESMAN (TSP) decision problem defined above is not really the problem one wants to solve in practice. In practice, one wants to find a tour (permutation) that has the shortest possible length, not simply to tell whether a tour exists that obeys a given length bound. The latter problem was simply chosen as a decision problem in NP that could serve as a "stand-in" for its optimization counterpart, based on the observation that the optimization problem could be no easier than its stand-in. (Finding an optimal tour would trivially enable us to tell whether any tour exists whose length obeys the given bound.) The term "NP-hard" has been introduced for use in this context.

Definition. A search problem X is NP-*hard* if for some NP-complete problem Y there is a polynomial-time Turing reduction from Y to X.

Note the obvious consequence that an NP-hard problem cannot be solvable in polynomial time unless P = NP.

The optimization version of the TSP is clearly NP-hard, as are all other search problems that contain NP-complete problems as special cases. Moreover, the term "NP-hard" serves a valuable function even in the realm of decision problems, where it provides a simple way of announcing the presumed intractability of problems that may not be in NP, but are still no easier than the NP-complete problems. For instance, the complement co-X of an NP-complete problem X, although just as intractable as X, will not be NP-complete unless NP = co-NP. It can be called NP-hard, however, if all we want to do is emphasize its presumed intractability. (When more precise language is required, the term "co-NP-complete" is available for this case.)

Unlike the NP-complete problems, the NP-hard problems do not form an equivalence class. Although no NP-hard problem can be in P unless P = NP, the converse need not be true. (Even undecidable problems can be NP-hard.) To get an equivalence class, we need to be able to impose an upper bound on the complexity of the problems. For this purpose, we introduce the following definition.

DEFINITION. A search problem X is NP-*easy* if for some problem Y in NP there is a polynomial-time Turing reduction from X to Y.

Note that all NP-easy problems will be solvable in polynomial time if P = NP.

Perhaps surprisingly, the optimization problem version of the TSP is NP-easy as well as NP-hard. It can be solved by binary search using an oracle for the following problem in NP: Given a sequence $c_1, c_2, \ldots, c_n$ of cities with specified intercity distances, a bound B, and an integer k, is there a tour that has length B or less whose first k cities are c_1 through c_k, in order? Similar arguments apply to many other search problems.

The set of search problems that are both NP-hard and NP-easy, an equivalence class under polynomial-time Turing reductions, thus constitutes a natural extension of the class of NP-complete problems to search problems in general, an extension that is captured in the following definition (from [87]).

DEFINITION. A search problem is NP-*equivalent* if it is both NP-hard and NP-easy.

Some additional notation will prove useful in what follows.

DEFINITION. The class $F\Delta_2^P(\Delta_2^P)$ consists of the set of all NP-easy search problems (decision problems).

Commonly used pseudonyms for $F\Delta_2^P$ and Δ_2^P that are perhaps more informative are "FP^{NP}" and "P^{NP}". The symbolism of the "Δ" notation will become clearer when we discuss the polynomial hierarchy in Section 2.5. Note that the NP-equivalent problems are simply those that are complete under polynomial-time Turing reductions for $F\Delta_2^P$.

2.4. *Between NP and Δ_2^P: the class D^P and the Boolean hierarchy*

For practical purposes there appears to be little point in distinguishing amongst the NP-equivalent problems. As with NP-complete problems, each can be solved in single exponential time ($O(2^{p(n)})$ for some polynomial p), but none can be solved in polynomial time unless P = NP (in which case they all can). Furthermore, if any problem in NP can be solved in time $O(n^{c \log n})$ for some c, so can all NP-equivalent problems, and similar statements hold for time $2^{c \log^k n}$ and any fixed k. Nevertheless, there are significant theoretical distinctions to be made, and interesting classes of decision problems that lie midway between NP and Δ_2^P, classes that are seemingly distinct from each other with respect to polynomial transformations (although not with respect to polynomial-time Turing reductions).

Of particular interest is the following class, introduced in [194].

DEFINITION. The class D^P consists of all those languages that can be expressed in the form $X \cap Y$, where $X \in$ NP and $Y \in$ co-NP.

Note that this is far different from the class NP$\cap$co-NP. It in fact contains all of NP$\cup$co-NP, and does not equal that union unless NP = co-NP. Indeed, the three statements "D^P = NP$\cup$co-NP", "Δ_2^P = NP$\cup$co-NP", and "NP = co-NP" are all equivalent [194]. As we have already seen, oracles exist for which both outcomes occur. A variety of interesting types of problems appear to be in D^P − (NP$\cup$co-NP).

A first example is the *exact answer* problem. For instance, in the exact answer version of the TSP, we are given a list of cities together with their intercity distances and a bound B, and are asked whether the optimal tour length is *precisely* B. This is the intersection of the TRAVELLING SALESMAN decision problem given in Section 2.1 (in NP) and the (co-NP) question that asks whether all tours are of length B or more. This problem is in fact complete for D^P under polynomial transformations (or "D^P-complete") [194] and hence cannot be in either NP or co-NP unless the two classes are equal. Another example of an exact answer problem that is complete for D^P is EXACT CLIQUE: Given a graph G and an integer K, is the maximum clique size precisely K? [194].

A second type of problem in D^P is the *criticality* problem. For instance, consider CRITICAL SATISFIABILITY: Given an instance of SATISFIABILITY, is it the case that the set of clauses is unsatisfiable, but deleting any single clause is enough to yield a subset that is satisfiable? Here the "unsatisfiability" restriction determines a co-NP language, and the m satisfiability restrictions, where m is the number of clauses, can be combined into a single instance of SATISFIABILITY, which is in NP. This problem is also D^P-complete [193].

A third type of problem in D^P is the *uniqueness* problem, as in UNIQUE SATISFIABILITY: Given an instance of SATISFIABILITY, is it the case that there is one, and only one, satisfying truth assignment? This is once again clearly in D^P and it is not difficult to show that it is NP-hard (e.g., see [33]). In this case, however, we do not know whether it is D^P-complete. There are oracles for which it is and for which it is not [33]. (D^P properly contains NP$\cup$co-NP for both oracles.) What we do know is that UNIQUE SATISFIABILITY cannot be in NP unless NP = co-NP [33]. The possibility that UNIQUE SATISFIABILITY is in co-NP has yet to be similarly limited. (For a slightly more thorough treatment of D^P, see [133].)

Related to the class D^P are two intermingled hierarchies of classes within Δ_2^P: the *Boolean hierarchy* and the *query hierarchy* (e.g., see [48, 49, 106, 250, 252]).

DEFINITION. The *Boolean hierarchy* consists of the classes BH_k, $k \geqslant 0$, as follows:

(1) $BH_0 = P$.

(2) If $k>0$, BH_k is the set of all languages expressible as $X-Y$, where $X \in NP$ and $Y \in BH_{k-1}$.

At present there is no consistently adopted notation for these classes; "BH(k)" is used for BH_k in [53, 147] and "NP(k)" is preferred in [48]. The notation used here is chosen for its mnemonic value, especially in relation to the following notation, which *is* consistently used.

Definition. The class BH is equal to the union $\bigcup_{k=0}^{\infty} BH_k$.

See Fig. 1 (Section 6) for a schematic view of this hierarchy and how it relates to the other classes we have seen so far. Note that $BH_1 = NP$ and $BH_2 = D^P$. There exist oracles for which this hierarchy is infinite, i.e., for which $BH_k \neq BH_{k+1}$ for all $k \geqslant 0$, as well as oracles for which it collapses, i.e., for which $BH = BH_k$ for some $k \geqslant 0$. Indeed, for any $k \geqslant 0$ there is an oracle such that all classes BH_i, $0 \leqslant i < k$, are distinct, but for which $BH_k = BH = \Delta_2^P$ (in fact, it will equal PSPACE, but we have not defined PSPACE yet; see Section 2.6) [48]. Note that it need not be the case that $BH = \Delta_2^P$. Indeed, there are oracles that separate the two classes. In particular, there are oracles for which BH does not have any complete problems under polynomial transformations, even though Δ_2^P always has such complete problems [48]. The individual classes BH_k also have complete problems, although for classes above $BH_2 = D^P$ the currently known complete problems are somewhat contrived. For instance, for each even $k>0$, the following problem is complete for BH_k: "Given a graph G, is $\chi(G)$ an odd number lying in the interval $[3k, 4k]$?" [48].

Let us now turn to the related *query hierarchy*. In defining it, we shall make use of the following formalism:

Definition. For each function $f: Z^+ \to Z^+$, the class $P^{NP[f(n)]}$ is the set of all languages that can be recognized by an oracle Turing machine with an oracle for SATISFIABILITY that makes no more than $f(n)$ queries of the oracle, where n is the input size.

(Note that $P^{NP} = \Delta_2^P$ is the same as $P^{NP[n^{O(1)}]} = \bigcup_{k=1}^{\infty} P^{NP[n^k]}$.)

Definition. The *query hierarchy* consists of the classes QH_k, $k \geqslant 0$, where for each such k, $QH_k = P^{NP[k]}$. (Here k represents the constant function $f(n) = k$.)

Definition. The class QH is equal to the union $\bigcup_{k=0}^{\infty} QH_k$.

It is easy to see that $BH_k \subseteq QH_k$; QH_k can in turn be shown to reside in a class of the Boolean hierarchy, albeit one that is somewhat higher up [50] (for a proof, see [250]). Thus $QH = BH$, and we can conclude that either both hierarchies are infinite, or both collapse to some finite level [106, 250]. If they do collapse, there are some interesting consequences for a more famous hierarchy, and we shall discuss these in the next section.

The above results do not carry over to the analogs of these hierarchies for arbitrary search problems. Let $\mathrm{FP}^{\mathrm{NP}[f(n)]}$ be the search problem analog of the class $\mathrm{P}^{\mathrm{NP}[f(n)]}$. The hierarchy of classes $\mathrm{FP}^{\mathrm{NP}[k]}$, $k \geqslant 0$, cannot collapse unless P = NP, a much stronger result than is known for the query hierarchy. Indeed, the precise number of queries asked makes a difference (assuming P ≠ NP) up to at least $\frac{1}{2}\log n$ queries [163].

This raises the question of classes defined using a nonconstant bound on the number of queries, in particular $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ and $\mathrm{FP}^{\mathrm{NP}[O(\log n)]}$. Note that the former contains (and may properly contain) all of QH = BH. With these more general classes, we once again can obtain interesting complete problems. The derivation is somewhat more straightforward in the case of the search problem classes. For instance, with an appropriate definition of reduction (the "metric" reduction of [163]), the problem of determining (rather than merely verifying) the size of the largest clique in a graph G is complete for the class $\mathrm{FP}^{\mathrm{NP}[O(\log n)]}$. The $O(\log n)$ here comes from the number of questions that need to be asked when performing a binary search to find the answer, given that the maximum clique size is no more than the number of vertices in G. If the optimal value one is asking to determine can be significantly larger than the input size, one can get much higher complexities. In particular, determining the length of an optimal travelling salesman tour is complete for all of $\mathrm{F}\Delta_2^{\mathrm{p}}$ [163].

Obtaining problems that are complete for the decision problem classes $\mathrm{P}^{\mathrm{PN}[O(\log n)]}$ and Δ_2^{p} under polynomial transformations is a bit more of a challenge. The natural plan of attack would be to find appropriate decision versions of the maximum clique search problem and the TSP, but note that our previous attempts to convert such problems to decision problems forced the problems to lose complexity. "Is there a tour of length B or less?" dropped the TSP from Δ_2^{p} to NP, and "Is the optimal tour length exactly B?" dropped it to D^{P}, almost as far. To obtain completeness for $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ and Δ_2^{p}, it turns out that one must find ways to provide instances with less in the way of hints as to the optimal clique size or tour length. Two questions that do the trick for the TSP and Δ_2^{p} are as follows: "Is there exactly one optimal tour?" [191] and "Is the optimal tour length divisible by k?" (given k as part of the input) [163]. Similarly, the problem of determining whether the maximum clique size for graph G is divisible by k (given G and k) is complete for $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ [163].

A final question to consider is whether $\mathrm{P}^{\mathrm{NP}[O(\log n)]} = \Delta_2^{\mathrm{p}}$ or $\mathrm{FP}^{\mathrm{NP}[O(\log n)]} = \mathrm{F}\Delta_2^{\mathrm{p}}$. Here it is once again easier to prove separation results for the function classes: It is shown in [163] that the $\mathrm{F}\Delta_2^{\mathrm{p}}$ equality can hold only if P = NP, whereas it is not known whether the consequences of the Δ_2^{p} equality would be so severe. Indeed, oracles exist for which the Δ_2^{p} equality holds but not the one for $\mathrm{F}\Delta_2^{\mathrm{p}}$ [163].

2.5. *The polynomial hierarchy*

The class Δ_2^{p} gets its name from its membership in a hierarchy that is far more famous than those of the previous section. The *polynomial hierarchy*, introduced in [187] as a computational analog to the Kleene arithmetic hierarchy of recursion theory, consists of classes Δ_k^{p}, Σ_k^{p}, and Π_k^{p}, $k \geqslant 0$, defined as follows:

$$\Delta_0^{\mathrm{p}} = \Sigma_0^{\mathrm{p}} = \Pi_0^{\mathrm{p}} = \mathrm{P},$$

and, for all $i \geqslant 0$,

$$\Delta^{\mathrm{P}}_{k+1} = \mathrm{P}^{\Sigma^{\mathrm{P}}_k}, \qquad \Sigma^{\mathrm{P}}_{k+1} = \mathrm{NP}^{\Sigma^{\mathrm{P}}_k}, \qquad \Pi^{\mathrm{P}}_{k+1} = \text{co-}\Sigma^{\mathrm{P}}_{k+1}.$$

In other words, $\Delta^{\mathrm{P}}_{k+1}$ ($\Sigma^{\mathrm{P}}_{k+1}$) is the set of all languages recognizable in polynomial time (nondeterministic polynomial time) with an oracle to a problem in Σ^{P}_k, and Π^{P}_{k+1} consists of the complements of all languages in $\Sigma^{\mathrm{P}}_{k+1}$. In particular, $\Delta^{\mathrm{P}}_1 = \mathrm{P}$, $\Sigma^{\mathrm{P}}_1 = \mathrm{NP}$, and $\Pi^{\mathrm{P}}_1 = \text{co-NP}$, while, as stated earlier, $\Delta^{\mathrm{P}}_2 = \mathrm{P}^{\mathrm{NP}}$. Note that $\Delta^{\mathrm{P}}_k \subseteq \Sigma^{\mathrm{P}}_k \cap \Pi^{\mathrm{P}}_k$. It is an open problem whether the containment is proper, just as it was open for the case of $k=1$ ($\mathrm{P} \subseteq \mathrm{NP} \cap \text{co-NP}$). Similarly, $\Delta^{\mathrm{P}}_{k+1}$ contains $\Sigma^{\mathrm{P}}_k \cup \Pi^{\mathrm{P}}_k$ and we do not know whether the containment is proper for any $k \geqslant 1$. Given these containment relationships, we can capture the set of all languages in the polynomial hierarchy with the following definition.

DEFINITION. The class PH is equal to the union $\bigcup_{k=0}^{\infty} \Sigma^{\mathrm{P}}_k$.

For a schematic illustration of PH and the classes of the polynomial hierarchy, see Fig. 2. There is an interesting alternative method for defining the classes Σ^{P}_k and Π^{P}_k, based on alternations of quantifiers. Note that we can extend the notion of "string relation" from the binary relations we discussed in Section 1.1 to k-ary relations for arbitrary $k \geqslant 2$. Σ^{P}_k is then the set of languages that can be expressed in the following format:

$$\begin{aligned}\{x\colon &(\exists y_1 \text{ with } |y_1| \leqslant p(|x|))\\ &\quad(\forall y_2 \text{ with } |y_2| \leqslant p(|x|))\\ &\quad\quad\cdots\\ &\quad\quad\quad(Q_k y_k \text{ with } |y_k| \leqslant p(|x|))\\ &\quad\quad\quad\quad[\langle y_1, y_2, \ldots, y_k, x\rangle \in R]\}\end{aligned}$$

where R is a polynomial-time recognizable k-ary relation, p is a polynomial, and Q_k is $\exists$ if k is odd, $\forall$ otherwise, and in general the quantifiers alternate [234]. (Note that, in the terminology of Section 1.3, this is equivalent to saying that Σ^{P}_k is the set of languages recognized by polynomial-time alternating Turing machines where the root of the computation tree is always labelled by "$\exists$" and the number of alternations is bounded by k.) The definition of Π^{P}_k is obtained by replacing all $\exists$s by $\forall$s, and vice versa.

Using this formalism, it is easier to identify problems as belonging in particular classes of the hierarchy. For instance, consider sentences of the form

$$(Q_1 x_1)(Q_2 x_2)\ldots(Q_j x_j)[F(x_1, x_2, \ldots, x_j)]$$

where each Q_i is a quantifier ($\exists$ or $\forall$) and F is a Boolean expression in the given variables. Let us say that such a sentence has a "quantifier alternation" for each $i > 1$ such that $Q_i \neq Q_{i-1}$ and for Q_1 itself. Then the set $\mathrm{QBF}_{k,\exists}$ of true sentences of this form with k quantifier alternations and with $Q_1 = \exists$ is a language in Σ^{P}_k. In fact, not surprisingly, it is complete for that class [255] under polynomial transformations. Similarly, the set $\mathrm{QBF}_{k,\forall}$ of true sentences with k quantifier alternations and with $Q_1 = \forall$ is complete for Π^{P}_k.

A more interesting example that is complete for a level of the polynomial hierarchy above Δ_2^P is the following problem, which is complete for Σ_2^P [237].

INTEGER EXPRESSION INEQUIVALENCE

Instance: Two *integer expressions* g and h, where the syntax and semantics of integer expressions are defined inductively as follows: the binary representation of a positive integer n is an integer expression representing the singleton set $\{n\}$; if e and f are integer expressions representing sets E and F, then $(e \cup f)$ is an expression representing the set $E \cup F$ and $(e+f)$ is an expression representing the set $\{m+n: m \in E \text{ and } n \in F\}$.

Answer: "Yes" if g and h represent different sets.

One level up, we have the following problem, complete for Δ_3^P [164].

DOUBLE KNAPSACK

Instance: Positive integers $x_1, \ldots, x_m, y_1, \ldots, y_n, N, k$.

Answer: "Yes" if the kth bit of the number M (to be defined below) is 1. To define M, let Z be the set of integers I such that (a) there exists an $S \subseteq \{1,2,\ldots,m\}$ with $\sum_{i \in S} x_i = I$, and (b) there is no $T \subseteq \{1,2,\ldots,n\}$ such that $\sum_{i \in T} y_i = I$. M is the largest element of $Z \cap \{1,2,\ldots,N\}$ (or 0, if the set is empty).

For other interesting natural problems that appear to be located above Δ_2^P in the hierarchy, see [172]. Note that a problem that is complete for Σ_k^P cannot be in a class that is lower in the hierarchy unless the two classes are equal. This raises the question of whether the polynomial hierarchy might collapse to some fixed level.

As a first and most important observation along this line, note that if P = NP, then $\Sigma_{k+1}^P = \Sigma_k^P = P$ for all $k \geq 0$, and so the hierarchy collapses to P. In fact P = NP if and only if P = PH. Observe that this in a sense makes the fine structure of the polynomial hierarchy about as "academic" as that of $\Delta_2^P - NP$. If $P \neq NP$, the hardest problems in PH will be intractable, but no more so than NP-complete problems seem to be (all problems in PH can be solved straightforwardly in single exponential time). If P = NP, all problems in PH will be in P. There might be a practical distinction, however, if NP ended up in some class intermediate between P and single exponential time. For instance, if the NP-complete problems could be solved in time $2^{c\log^2 n}$, then although this would extend to Δ_2^P, it would not seem to extend to all of PH. The best we can say a priori is that each problem in PH would be solvable in time $2^{c\log^k n}$ for some k, i.e., in time exponential in "polylog n". (The proof is by straightforward padding arguments.)

The fact that PH collapses to P if P = NP, i.e., if $\Sigma_0^P = \Sigma_1^P$, generalizes. It is not difficult to see that for any $k > 0$, $\Sigma_k^P = \Sigma_{k+1}^P$ implies $PH = \Sigma_k^P$. Similar statements hold for Δ_k^P and Π_k^P, as well as for $\Sigma_k^P = \Pi_k^P$ [234]. Another interesting and somewhat surprising result, alluded to in the previous section, links the collapse of the polynomial hierarchy to that of the Boolean and query hierarchies within Δ_2^P. In [53, 147] it is shown that if the Boolean hierarchy (or equivalently the query hierarchy) is finite, i.e. if it collapses to some fixed level, then $PH \subseteq \Delta_3^P$. A result to which we alluded in Section 2.1 is also relevant: If there is a sparse language that is complete for NP under Turing reductions, then PH collapses into Δ_2^P [179] (in fact to $P^{NP[O(\log n)]}$, as defined in the previous section

[146]). Finally, it has recently been shown that the polynomial hierarchy will collapse to Π_2^p if GRAPH ISOMORPHISM turns out to be NP-complete [39, 220].

Despite all these intriguing results, the consensus of the experts currently is that the hierarchy does not collapse. If so, the result is likely to be hard to prove, as each of the important alternatives for PH is now known to hold in at least one relativized world. Oracles for which PH is an infinite hierarchy and does not collapse at all are presented in [257] and Ko has recently shown that for each $k \geqslant 0$, there is an oracle for which $\Sigma_k^p \neq \Sigma_{k+1}^p = \text{PH}$ [159]. (With respect to a random oracle, all we currently know is that PH does not collapse below NP$\cup$co-NP [29].) The story continues in the next section.

2.6. *PSPACE and its subclasses*

The polynomial hierarchy consists of all those languages recognizable by polynomial-time bounded alternating Turing machines obeying fixed bounds on the number of alternations. One might reasonably ask what happens when the number of alternations is not so bounded. Let "APTIME" denote the class of languages accepted by polynomial-time alternating Turing machines (with no constraints on the number of alternations). We are thus asking whether PH = APTIME.

This is an even more interesting question than it might at first seem, for APTIME is a more interesting class than it might first seem. In fact it equals "PSPACE", the set of all languages recognizable by polynomial-*space* bounded (deterministic) Turing machines. This is a consequence of a more general result proved by Chandra, Kozen and Stockmeyer. Before stating that theorem, we introduce some notation that will be useful both here and later:

DEFINITION. If $T(n)$ is a function from the positive integers to themselves, then DTIME[$T(n)$] (NTIME[$T(n)$], ATIME[$T(n)$]) is the set of all languages recognized by DTMs (NDTMs, ATMs) in time bounded by $T(n)$, where n is the size of the input.

DEFINITION. If $S(n)$ is a function from the positive integers to themselves, then DSPACE[$S(n)$] (NSPACE[$S(n)$], ASPACE[$S(n)$]) is the set of all languages recognized by DTMs (NDTMs, ATMs) in space bounded by $S(n)$, where n is the size of the input.

2. THEOREM (Chandra, Kozen and Stockmeyer [52]). *For any function* $T(n) \geqslant n$, $\text{ATIME}[T(n)] \subseteq \text{DSPACE}[T(n)] \subseteq \bigcup_{c>0} \text{ATIME}[c \cdot T(n)^2]$.

Taking a union with $T(n)$ ranging over all polynomials yields the claimed result that $\text{APTIME} = \bigcup_{k>0} \text{ATIME}[n^k] = \bigcup_{k>0} \text{DSPACE}[n^k] = \text{PSPACE}$.

With PSPACE we reach our first complexity class containing P that is not known to collapse to P if P = NP. There are oracles such that P = NP and yet P $\neq$ PSPACE [159] (as well, of course, as oracles for which P = NP = PSPACE and such that P $\neq$ NP = PSPACE) [17]. Indeed, the range of oracular possibilities with respect to the entire polynomial hierarchy is now quite comprehensive. Among oracles for which PH does not collapse, there are ones both for PH = PSPACE and for which PH $\neq$ PSPACE [114, 115, 257]. Similarly, for any $k \geqslant 0$, among those oracles for which PH collapses to

Σ_k^P, there are also ones both for PH = PSPACE and for which PH $\neq$ PSPACE [159]. (With respect to a random oracle, however, PH $\neq$ PSPACE [47, 13].)

Thus it would seem that APTIME = PSPACE is a new and distinct class, and that a problem complete for it under polynomial transformations ("PSPACE-complete") is not likely to be in PH (much less P). There are interesting PSPACE-complete languages. An easy first example is the following generalization of the languages $\text{QBF}_{k,\exists}$ and $\text{QBF}_{k,\forall}$ as defined in the previous section.

QUANTIFIED BOOLEAN FORMULAS (QBF)

Instance: A sentence of the form $S=(Q_1x_1)(Q_2x_2)\dots(Q_jx_j)[F(x_1,x_2,\dots,x_j)]$ where each Q_i is a quantifier ($\exists$ or $\forall$) and F is a Boolean expression in the given variables. (Note that there is no restriction on the number of alternations.)

Answer: "Yes" if S is a true sentence.

Starting with QBF, researchers have identified a wide variety of other PSPACE-complete problems. First among them are problems about games. Note that the alternation inherent in the APTIME definition of PSPACE is analogous to the thought processes one must go through in determining whether one has a forced win in a two-person game. ("Do I have a move such that for all possible next moves of my opponent there is a second move for me such that for all second moves of my opponent there is a third move for me such that...") Thus it is perhaps not surprising that many problems about games are PSPACE-complete. Here is a simple one from [219].

GENERALIZED GEOGRAPHY

Instance: Directed graph $G=(V,A)$, specified vertex v_0.

Answer: "Yes" if the Player 1 has a forced win in the following game. Players alternate choosing new arcs from the set A. Player 1 starts by choosing an arc whose tail is v_0, and thereafter each player must choose an arc whose tail equals the head of the previously chosen arc. The first player unable to choose a new arc loses.

Note that this generalizes the "geography" game in which players alternate choosing the names of countries, with the first letter of each name having to agree with the last letter of its predecessor. Other PSPACE-complete games include a generalization of the game Hex from the Hex board to an arbitrary graph [74] and the endgame problem for a generalization of the game GO to arbitrarily large grids [176]. (For surveys of further recreational examples, see [87, 131].)

There are also many PSPACE-complete problems in which the alternation is more circumspect, such as FINITE-STATE AUTOMATA EQUIVALENCE (given the state diagrams of two nondeterministic finite automata, do they recognize the same language? [237]), REGISTER MINIMIZATION (given a straight-line program P and a bound B, can the output of P be computed using just B registers if recomputation is allowed? [89]), and the GENERALIZED MOVER'S PROBLEM (given a collection of three-dimensional "jointed robots", represented by polyhedra joined at their vertices, an assignment of locations to these robots in a 3-dimensional space with fixed polyhedral obstacles, and a set of desired final locations for the robots, can they be moved to their desired locations without ever causing an overlap between separate robots or a robot and an obstacle? [206]).

With PSPACE, we have reached the top of the tower of classes for which the question of containment in P remains open. The obvious candidate for a next step, nondeterministic polynomial space or "NPSPACE", has a rather serious drawback: like APTIME, it equals PSPACE itself. This follows from "Savitch's Theorem", proved in 1970.

3. THEOREM (Savitch [216]). *For any "space constructible" function* $T(n) \geqslant \log_2 n$, $\mathrm{NSPACE}[T(n)] \subseteq \mathrm{PSPACE}[T(n)^2]$.

Thus any dream of an alternating hierarchy above PSPACE in analogy to the polynomial hierarchy above P collapses back into PSPACE itself.

Savitch's result does not, however, dispose of the question of nondeterminism for the interesting subclass of those problems solvable in *linear* space.

DEFINITION. The class LIN-SPACE (NLIN-SPACE) equals $\bigcup_{c>0} \mathrm{DSPACE}[cn]$ ($\bigcup_{c>0} \mathrm{NSPACE}[cn]$).

These classes are especially interesting since NLIN-SPACE consists precisely of those languages that can be generated by "context-sensitive grammars" or, equivalently, recognized by "(nondeterministic) linear bounded automata" or "LBAs" (see [122, 214]). The question of whether LIN-SPACE = NLIN-SPACE is thus equivalent to the long open problem of whether deterministic LBAs are as powerful as general (nondeterministic) ones. Savitch's Theorem does not resolve the question, since it only shows that NLIN-SPACE $\subseteq \bigcup_{c>0} \mathrm{DSPACE}[cn^2]$.

It should be noted that, as complexity classes, LIN-SPACE and NLIN-SPACE differ somewhat from the ones that we have seen so far. They are machine-independent in much the same sense that P and PSPACE are, but there is one valuable property of P and PSPACE that they do not share: they are *not* closed under polynomial transformations (as can be shown by padding arguments using Theorem H2 of Section 1.4 [35]). One consequence of this is that membership in LIN-SPACE (NLIN-SPACE) can be strongly dependent on the way in which problem instances are represented. A graph problem that is in LIN-SPACE if instances are given by adjacency matrices, might conceivably fail to be in it if instances are given by adjacency lists, even though both representations are "reasonable" in the sense of Section 1.1. A second consequence concerns the relationship of the two classes to P and NP.

By Theorem H2 of Section 1.4, LIN-SPACE and NLIN-SPACE are proper subclasses of PSPACE. Unlike PSPACE, however, they are not known to contain NP, or even P. Indeed, all we know for certain about their relationships to the latter two classes is that LIN-SPACE (NLIN-SPACE) does not equal either P or NP. (LIN-SPACE and NLIN-SPACE are not closed under polynomial transformations whereas P and NP both are.) The most natural assumption is that LIN-SPACE and NLIN-SPACE are incomparable to P and NP, although strict containment one way or the other is not ruled out. For instance, NLIN-SPACE$\subset$P if P$=$PSPACE. This is actually an "if and only" statement, since there are PSPACE-complete problems in NLIN-SPACE, for instance, CONTEXT-SENSITIVE LANGUAGE MEMBERSHIP: given a context-sensitive grammar G and a string s, is s in the language generated by G? Indeed, there is a *fixed* context-sensitive language whose membership problem is PSPACE-complete [37].

Although the question of LIN-SPACE versus NLIN-SPACE remains open, a related problem of almost equal antiquity has recently been resolved. This one involves co-NLIN-SPACE, the set of all languages whose complements are in NLIN-SPACE, and its resolution puts an end to any hopes of building an alternation hierarchy above LIN-SPACE. The key result here is the following result.

4. THEOREM (Immerman [125] and Szelepcsényi [239]). *For any function* $S(n)\geqslant\log n$, NSPACE$[S(n)]=$co-NSPACE$[S(n)]$.

As a consequence, NLIN-SPACE$=$co-NLIN-SPACE, and the alternation hierarchy above LIN-SPACE collapses to NLIN-SPACE, even if we assume LIN-SPACE$\neq$ NLIN-SPACE.

We shall see further applications of Theorems 3 and 4 later in this catalog.

3. Provably intractable problems

None of the complexity classes considered in the previous section contains a problem that has been *proved* to be intractable, no matter how hard some of those problems may have looked. If P$=$PSPACE, a possibility that has not yet been ruled out, then all problems in the classes discussed would be solvable in polynomial time. In this section we consider complexity classes that *are* known to contain intractable problems. A schema of the classes to be covered is given in Fig. 3.

First, it should be noted that for any well-behaved functional bound f that grows faster than any polynomial (for instance, $f(n)=n^{\log\log\log\log n}$), the class DTIME$(f(n))$ of decision problems solvable by $f(n)$ time-bounded DTMs must, by Theorem H1 of Section 1.4, contain problems that are not in P and hence are intractable by our definition. We shall not be interested in such "subexponential" classes however, as all the interesting theoretical work has concerned classes with exponential (or worse) resource bounds, and these are the ones we shall discuss.

3.1. *The class EXPTIME and its variants*

There are currently two distinct notions of "exponential time" in use. In what follows, we shall introduce a terminological distinction between the two notions, but the reader should be aware that the terminology we use is not completely standard. In perusing the literature, one should thus take care to ascertain the definition being used. The first and more natural notion of exponential time gives rise to the following class.

DEFINITION. The class EXPTIME equals $\bigcup_{k>0}\mathrm{DTIME}[2^{n^k}]$, i.e., is the set of all decision problems solvable in time bounded by $2^{p(n)}$, where p is a polynomial.

As defined, EXPTIME equals the class APSPACE of all languages recognizable by alternating Turing machines (ATMs) using polynomial-bounded space. This equivalence follows from a theorem of Chandra, Kozen and Stockmeyer that complements Theorem 2 above.

5. THEOREM (Chandra, Kozen and Stockmeyer [52]). *For any function* $S(n) \geqslant \log n$, $\mathrm{ASPACE}[S(n)] = \bigcup_{c>0}\mathrm{DTIME}[2^{c\cdot S(n)}]$.

EXPTIME thus contains both PSPACE and NP. (There are oracles that separate EXPTIME from PSPACE, as well as oracles for which NP = PSPACE = EXPTIME [66].)

The second notion of "exponential time" restricts attention to linear exponents and gives rise to the following class:

DEFINITION. The class ETIME equals $\bigcup_{c>0}\mathrm{DTIME}[2^{cn}]$.

Note that, by Theorem H1, ETIME is properly contained in EXPTIME. Like EXPTIME, it is independent of machine model (since a function $p(2^{cn})$, p a polynomial, is simply $O(2^{c'n})$ for a possibly larger constant c'). ETIME has several theoretical drawbacks, however. Like LIN-SPACE, it is not closed under polynomial transformations, and so a problem that is in ETIME under one choice of "reasonable" input representation could conceivably *not* be in ETIME if another "reasonable" choice was made. Furthermore, the computational dominance of ETIME over P is perhaps not quite as strong as one would like. In particular, $\mathrm{ETIME} \neq \mathrm{P}^{\mathrm{ETIME}}$, i.e., one can do more in polynomial time with an oracle for a problem in ETIME than one can do in ETIME alone. This is bcause $\mathrm{P}^{\mathrm{ETIME}} = \mathrm{EXPTIME}$, as can be shown by simple padding arguments (e.g., see [246]). In contrast, note that $\mathrm{EXPTIME} = \mathrm{P}^{\mathrm{EXPTIME}}$. Finally, unlike EXPTIME, ETIME is not known to contain NP, although it is known to be unequal to NP (because it is not closed under polynomial transformations [34, 35]). There exist oracles yielding each of the three possibilities (ETIME incomparable to NP, $\mathrm{ETIME} \subset \mathrm{NP}$, and $\mathrm{NP} \subset \mathrm{ETIME}$) [66].

Because EXPTIME contains problems that require exponential time, any problem that can be shown to be complete for EXPTIME under polynomial transformations, i.e., "EXPTIME-complete", is provably intractable. (By padding arguments we can show that any problem that is ETIME-complete is EXPTIME-complete, so we restrict ourselves to the latter definition.) Some examples make explicit use of the aternation implicit in the definition of EXPTIME as APSPACE, such as GENERALIZED CHECKERS and GENERALIZED CHESS (given an endgame position in one of these games, as generalized from the standard 8×8 board to an $N \times N$ one, does white have a forced win?) [83, 210]. Others are less obvious, such as determining whether a given attribute grammar has the "circularity" property [128].

3.2. *The class NEXPTIME*

The nondeterministic analogs of EXPTIME and ETIME are as follows.

Definition. The class NEXPTIME equals $\bigcup_{k>0}\text{NTIME}[2^{n^k}]$.

Definition. The class NETIME equals $\bigcup_{c>0}\text{NTIME}[2^{cn}]$.

It is not difficult to see that P = NP implies both that EXPTIME = NEXPTIME and that ETIME = NETIME, and that ETIME = NETIME implies EXPTIME = NEXPTIME. None of the three converses is known to hold, however, and there exist oracles for which each is violated [66]. (There also exist oracles such that EXPTIME ≠ NEXPTIME, such that PSPACE ≠ EXPTIME, and such that PSPACE = EXPTIME = NEXPTIME [66].) Although P ≠ NP does not by itself seem to imply ETIME ≠ NETIME, we can get that conclusion if we also assume something about the structure of NP − P. In particular, ETIME ≠ NETIME if and only if NP − P contains a sparse language [111]. Whether this argues that NP − P contains sparse languages or that ETIME = NETIME is left to the biases of the reader. A final connection between NETIME and the P-versus-NP question (also of unclear significance) is the fact that $P^{\text{NETIME}} = NP^{\text{NETIME}}$ [116].

Since Theorem H3 of Section 1.4 implies that NEXPTIME must properly contain NP, any problem that is NEXPTIME-complete (under polynomial transformations) is not only not in P, it is not even in NP. Thus such results not only prove intractability, they prove the nonexistence of "short proofs" for yes-answers. (Once again, NETIME-completeness implies NEXPTIME-completeness and so can be ignored.) Examples of NEXPTIME-complete problems come from a variety of fields. A first example is worth highlighting, as we shall see variants on it in later sections. It is due to Stockmeyer and Meyer [237] and is reminiscent of the integer expression inequivalence problem of Section 2.5, although this time we are talking about *regular* expressions and different operators.

inequivalence of regular expressions over $(\cup, \cdot, {}^2)$

Instance: Two regular expressions e_1 and e_2 involving only the operations of union, composition, and squaring, where such an expression has the following syntax and semantics: A single symbol (0 or 1) is an expression representing the set consisting of the 1-character string (0 or 1). If e and f are expressions representing sets E and F, then "$e \cup f$" is an expression representing $E \cup F$, "$e \cdot f$" is an expression representing $\{xy: x \in E \text{ and } y \in F\}$, and e^2 is an expression that serves as a shorthand representation for the expression $e \cdot e$.

Answer: "Yes" if e_1 and e_2 represent different sets.

As a second example of a NEXPTIME-complete problem, consider the following tiling problem. A *tile* is an ordered 4-tuple $T = (N_T, S_T, E_T, W_T)$ of integers. Suppose we are given a finite collection C of tiles, a specific sequence $T_1, \ldots, T_m$ of tiles, satisfying $E_{Ti} = W_{T_{i+1}}$, $1 \leqslant i < m$, and an integer n (written in binary notation). The question we ask is whether there is a tiling of the $n \times n$ square, i.e., a mapping $f:\{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\} \to C$, such that $f(1, i) = T_i$, $1 \leqslant i \leqslant m$, and such that, for all i, j, $1 \leqslant i, j \leqslant n$,

(a) $j < n$ implies that $E_{f(i,j)} = W_{f(i,j+1)}$, and

(b) $i < n$ implies that $N_{f(i,j)} = S_{f(i+1,j)}$.

(Note the key fact that n can be exponentially larger than m and $|C|$. If we restrict ourselves to instances with $m = n$, the problem is only NP-complete [174].)

The idea of converting NP-complete problems into NEXPTIME-complete problems by using succinct representations of instances is also the key to our final example. Let us say that a Boolean circuit C *represents* a graph $G = (V, E)$ if, when given as input two integers i, j, $1 \leqslant i < j \leqslant |V|$, C outputs 1 if the graph contains an edge between vertex i and vertex j, and outputs 0 otherwise. Not all graphs have exponentially succinct Boolean circuit representations, but enough of them do that the following problem is NEXPTIME-complete [195]: given the Boolean circuit representation of a graph $G = (V, E)$, does G contain a clique of size $|V|/2$? (Note that for ordinary graph representations, this problem is "only" NP-complete.) Similar tricks can be pulled with other graph problems that are NP-complete under the standard representation [195]. Moreover, even if one starts with a problem that is trivial under the standard representation, conversion to the Boolean circuit representation is often enough at least to yield NP-completeness [85].

3.3. *On beyond NEXPTIME*

In this section we complete our climb up the ladder of intractability, with brief stops along the way to look at a few more interesting classes. A likely choice for the first stop would be the following.

DEFINITION. The class EXPSPACE equals $\bigcup_{k>0}$DSPACE$[2^{n^k}]$.

There is, however, an intermediate class between NEXPTIME and EXPSPACE that captures the precise complexity of an interesting problem. This class is defined by simultaneously bounding both time and alternations, rather than just a single one as in the definitions of NEXPTIME and EXPSPACE. (Note that by Theorem 2, EXPSPACE $= \bigcup_{k>0}$ATIME$[2^{n^k}]$.) To facilitate definitions based on simultaneous bounds, we introduce some additional notation.

DEFINITION. The class TA$[t(n), a(n)]$ is the set of all problems solvable by ATMs using at most $t(n)$ time and $a(n)$ alternations on inputs of length n.

Note that we have already seen classes for which this type of notation would have been appropriate. For example, $\Sigma_2^{\mathrm{P}} \cup \Pi_2^{\mathrm{P}} = \bigcup_{k>0}TA[n^k, 2]$, and the polynomial hierarchy PH $= \bigcup_{k>0, j>0}$TA$[n^k, j]$. An "exponential hierarchy" (EH), including EXPTIME and NEXPTIME as its first two classes, can be obtained by analogy, simply by replacing "n^k" by "2^{n^k}" in the above formulation for PH. The class of interest here, however, is not EH but a class that appears to lie somewhere between EH and EXPSPACE. Specifically, it is $\bigcup_{k>0}\mathrm{TA}[2^{n^k}, n]$. (No less formal name has yet been proposed.) Note that this class is presumably larger than EH, but (also presumably) smaller than EXPSPACE (for which $a(n)$ should be 2^{n^k} rather than simply n).

The problem whose complexity is captured by $\bigcup_{k>0}\mathrm{TA}[2^{n^k}, n]$ is the THEORY OF REAL ADDITION: given a first-order sentence S involving the logical connectives, the "$<$" and "$=$" relational symbols, binary constants for all the integers, and variables (with binary indices) representing real numbers, is S true? The completeness of this problem for $\bigcup_{k>0}\mathrm{TA}[2^{n^k}, n]$ follows from results in [30, 76]. (Note that, by being complete for this class, the problem is at least as hard as hard as any problem in NEXPTIME. Thus, for any axiomatization of the theory of real addition, there must be true theorems whose proofs are exponentially long.)

Stepping up the rest of the way to EXPSPACE, we can get even stronger intractability results, as problems complete for this class provably require exponential space as well as time. To obtain an example of an EXPSPACE-complete problem, all we need do is generalize the problem highlighted in the previous section, obtaining INEQUIVALENCE OF REGULAR EXPRESSIONS OVER $(\cup, \cdot, ^2, ^*)$ [237]. This is the version of the problem in which the Kleene "*" operator is added, i.e., if e is an expression representing the set E, we also allow the expression "e^*", which represents the set of all strings of the form $x_1 x_2 \ldots x_n$, where $n \geqslant 0$ ($n = 0$ means we have the empty string) and $x_i \in E$, $1 \leqslant i \leqslant n$.

As might be expected from the alternative definition of EXPSPACE in terms of alternating exponential time, there are also games that are complete for EXPSPACE. These are quite complicated however, involving teams of players, incomplete information, and rules sufficiently convoluted that it is perhaps best simply to refer the interested reader to the original source [205] and to [131].

Next up the ladder we come to what we shall denote as "2-EXPTIME" and "2-NEXPTIME", the classes of decision problems solvable by DTMs (NDTMs) operating in time bounded by $2^{2^{p(n)}}$ for some polynomial p. However, we must go a bit further (halfway to 2-EXPSPACE) to obtain our next interesting completeness result. The following problem is complete for the class $\bigcup_{k>0}\mathrm{TA}[2^{2^{n^k}}, n]$: PRESBURGER ARITHMETIC (the analog of the theory of real addition in which the variables are to be interpreted as natural numbers rather than arbitrary reals) [76, 30]. Thus, contrary to what we learned in school, integers are more difficult than real numbers (for which the theory of addition was exponentially easier, as mentioned above).

Continuing up the ladder we now have 2-EXPSPACE, and then k-EXPTIME, k-NEXPTIME, and k-NEXPSPACE, $k \geqslant 3$, where k refers to the number of levels of exponentiation in the appropriate resource bound. The next class of real interest, however, is the union of this hierarchy: the class of "elementary" decision problems.

DEFINITION. The class ELEMENTARY $= \bigcup_{k>1} k$-EXPTIME.

Even those unimpressed with the difficulty of problems in 2- or 3-EXPTIME will have to admit that if a problem is decidable but not in ELEMENTARY, it might as well not be decidable at all. Examples of such nonelementary problems are the "weak monadic second-order theory of successor" [186] and yet another generalization of the regular expression inequivalence problem, this time to INEQUIVALENCE OF REGULAR EXPRESSIONS OVER $(\cup, \cdot, \neg)$. Here we replace "squaring" and the Kleene "*" operation by negation: if e is an expression representing the set E, then "$\neg e$" is an expression representing $\{0,1\}^* - E$. Although determining whether two such expressions are equivalent is decidable, it requires running time that grows with a tower of 2s that is at least $\log n$ levels tall [237]. (The currently best upper bound involves a tower that is n levels tall [237].)

One could of course define complexity classes that are even larger than the ones so far discussed, but at this point it seems best to skip ahead to the class representing the supreme form of intractability: the undecidable (or nonrecursive) problems. These are traditionally the domain of recursion theorists, and can themselves be grouped into more and more complex classes, but we shall leave a detailed discussion of these to the Handbook on Logic [65] and other surveys, such as those contained in [102, 122]. Suffice it to point out that it is almost as easy to change a problem from decidable to undecidable as it is to change one from polynomial-time solvable to NP-complete. For example, if in the TILING problem mentioned above one asks that the tiling extend to all of $Z \times Z$ instead of a simple $n \times n$ subsquare, the problem becomes undecidable [251], as does PRESBURGER ARITHMETIC if one augments it to include multiplication [184].

4. Classes that count

In this section we return to the world of the only *presumably* intractable, to examine complexity classes that do not fit in the standard format of the previous sections. The classes we shall study are defined by variants of the NDTM in which the answer depends, not just on the existence or absence of computations ending in accept states, but on the *number* of such computations. We begin with classes in which the *precise* number is significant, and conclude with a whole panoply of "probabilistic" classes where the important criterion is that the number exceed a given threshold.

4.1. *Counting Turing machines and the class #P*

The first class we consider is restricted to functions $f: \{0,1\}^* \to Z_0^+$, i.e., from strings to the nonnegative integers. It was introduced by Valiant in [243], and is defined in terms of what Valiant calls *counting Turing machines*:

DEFINITION. A *counting Turing machine* (*CTM*) is an NDTM whose "output" for a given input string x is the number of accepting computations for that input.

DEFINITION. The class #P is the set of all functions that are computable by polynomial-time CTMs.

There is some debate about how "#P" should be pronounced, as the "#"-sign variously abbreviates "sharp", "pound", and "number", depending on context. The context here would seem to favor the last of the three alternatives, and that was advocated by [87] although many authors prefer the first.

A typical member of #P is the following HAMILTONIAN CIRCUIT ENUMERATION problem: given a graph G, how many Hamiltonian circuits does it contain? Note that this problem is NP-hard in the sense of Section 1.3, since we can use an oracle for it to solve the NP-complete HAMILTONIAN CIRCUIT problem. (Trivially, G has such a circuit if and only if the number of such circuits that it has is greater than 0.) Note, however, that counting the number of Hamiltonian circuits might well be harder than telling if one exists. To begin with, the decision problem that simply asks, given G and k, whether G has k or more Hamiltonian circuits, is not known to be in NP. The number of Hamiltonian circuits that a graph with n vertices can have need not be polynomially bounded in n, and so the approach of simply guessing the requisite number of circuits and checking their Hamiltonicity may well take more than polynomial time. No clever tricks for avoiding this difficulty are known. Indeed, this decision problem is not known to lie anywhere within the polynomial hierarchy, although it is clearly in PSPACE.

Based on the presumed unlikelihood that $\#\mathrm{P} \subseteq \mathrm{FP}$, Valiant [243] introduced a new class of (presumably) intractable problems: those that are "#P-complete".

DEFINITION. A problem X is #P-*hard* if there are polynomial-time Turing reductions to it from all problems in #P. If in addition $X \in \#\mathrm{P}$, we say that X is #P-*complete.*

Note the corollary that a #P-hard problem can be in FP only if $\#\mathrm{P} \subseteq \mathrm{FP}$. In practice, it turns out that a more restricted form of reduction, first introduced in [225] often suffices to prove #P-hardness, especially for enumeration problems whose underlying existence question is in NP.

DEFINITION. A *parsimonious transformation* is a polynomial transformation f from problem X to problem Y such that, if $\#(X,x)$ is defined to be the number of solutions that instance x has in problem X, then $\#(X,x) = \#(Y,f(x))$.

By appropriately modifying the standard transformations for showing NP-completeness, one can often obtain transformations that are parsimonious, or at least "weakly parsimonious" ($\#(X,x)$ can be computed in polynomial time from x and $\#(Y,f(x))$). In particular, Cook's Theorem can be modified to show that "#SAT", the problem of counting the number of satisfying truth assignments for an instance of SATISFIABILITY, is #P-complete. From this it follows that the enumeration versions of most NP-complete problems, including HAMILTONIAN CIRCUIT ENUMERATION, are #P-complete.

What is perhaps surprising is that enumeration versions of problems in P can also be #P-complete. The following is the most famous such #P-complete problem, first identified in [243].

PERMANENT COMPUTATION

Instance: An $n \times n$ 0-1 matrix A.

Answer: The value $perm(A)=\sum_\sigma \prod_{i=1}^n A_{i,\sigma(i)}$ of the *permanent* of A, where the summation is over all $n!$ permutations σ of $\{1,2,\ldots,n\}$.

Note that the permanent of a matrix is simply the variant on the determinant in which all summands are given positive signs. The analogous DETERMINANT COMPUTATION problem appears to be much easier however, since the determinant of a matrix A is computable in polynomial time, even when the entries of A are allowed to be arbitrary rationals (e.g., see [5]).

To see that PERMANENT COMPUTATION can be viewed as an enumeration problem, simply observe that $\prod_{i=1}^n A_{i,\sigma(i)}$ must be either 0 or 1. Hence $perm(A)$ is simply the number of permutations σ for which this product equals 1. Moreover, note that the problem of whether there *exists* a σ for which the product equals 1 is in P. Indeed, it is equivalent to the well-known polynomial-time solvable problem of telling whether a bipartite graph contains a perfect matching. Recall that for a given graph G, a *perfect matching* is a set M of edges such that every vertex of G is an endpoint of precisely one edge in M. Suppose we consider A to be the adjacency matrix for bipartite graph G_A on $2n$ vertices $v_1,\ldots,v_n$ and $u_1,\ldots,u_n$, with (u_i, v_j) an edge of G_A if and only if $A_{ij}=1$. Then it is easy to verify that $\prod_{i=1}^n A_{i,\sigma(i)}=1$ if and only if $M=\{(u_i, v_{\sigma(i)})\colon 1\leqslant i\leqslant n\}$ is a perfect matching for G_A. Thus $perm(A)$ is simply the number of distinct perfect matchings in G_A. Consequently, counting the number of perfect matchings in a bipartite graph is #P-complete, in direct contrast to such positive results as the fact that, given a graph G, one can in polynomial time determine both the number of distinct spanning trees it contains and the number of Euler circuits it contains [101].

For more examples of #P-complete problems, see [70, 200, 244]. One particularly interesting type of problem covered by these references is the *reliability* problem. As an example, consider the following problem.

GRAPH RELIABILITY

Instance: Directed graph $G(V, A)$ with specified source and sink vertices s and t. For each arc a, a rational number $p(a)\in[0,1]$.

Answer: The probability that there exists a path in G from s to t consisting entirely of "nonfailed" arcs, given that an arc a fails with probability $p(a)$, independently for all a.

Although not formally in #P, this problem is essentially the same as the problem of counting the number of distinct subgraphs of a certain type, and the latter problem *can* be shown to be #P-complete. (Technically, we should say that the reliability problems are #P-*equivalent*, in analogy with the NP-equivalent problems, as defined in Section 2.3. That is, they are #P-hard, and yet solvable in polynomial time with an oracle to a problem in #P, i.e., they are in the class $\mathrm{FP}^{\#\mathrm{P}}$. Another interesting #P-equivalent problem is that of computing the volume of the convex hull of a set of rational coordinate points in Euclidean n-space [70].)

One important enumeration problem that appears *not* to be #P-complete is the problem, given two graphs G and H, of counting the number of distinct isomorphisms between G and H. For this problem, the enumeration problem is polynomial-equivalent to the decision problem, and indeed, verifying the number of isomorphisms is known to be in NP, even though there may be an exponential number of them [183].

Another interesting set of enumeration problems that are not known to be #P-hard are such problems as, given an integer n (written in unary so that "polynomial time" means "polynomial in n"), how many distinct labelled graphs with n vertices contain Hamiltonian circuits? This is the form for many classical enumeration problems in combinatorial mathematics. Some such problems are known to be in P; for instance the much simpler problem, given n, of determining how many distinct labelled graphs with n vertices exist (there are precisely $2^{n(n-1)/2}$). Others, like the former, remain open. For problems like these, a new class becomes relevant.

DEFINITION. The class $\#\mathrm{P}_1$ consists of all those problems in #P whose instances are restricted to strings over a single-letter alphabet.

The question of whether $\#\mathrm{P}_1 \subseteq \mathrm{FP}$ is addressed in [244]. It is perhaps easier than #P versus FP, but no less open. Furthermore, there are graph problems, albeit convoluted ones, that are $\#\mathrm{P}_1$-complete under parsimonious transformations and hence in FP if and only if $\#\mathrm{P}_1 \subseteq \mathrm{FP}$. For details, see [244].

Returning to the class #P, let us briefly consider its relationship to some of the other classes we have been considering, in particular the polynomial hierarchy. In discussing this relationship, we must first deal with the fact that #P is a class of functions, while PH and its subclasses are classes of languages. One option here is to consider a functional analog of PH. Suppose we replace $\Delta_1^{\mathrm{P}} = \mathrm{P}$ by FP and, for $k > 1$, replace $\Delta_k^{\mathrm{P}} = \mathrm{P}^{\Sigma_{k-1}^{\mathrm{P}}}$ by $\mathrm{F}\Delta_k^{\mathrm{P}} = \mathrm{FP}^{\Sigma_{k-1}^{\mathrm{P}}}$, the set of all functions computable in polynomial time with an oracle for a set in $\Sigma_{k-1}^{\mathrm{P}}$. We could then ask whether there exists a k such that $\#\mathrm{P} \subseteq \mathrm{F}\Delta_k^{\mathrm{P}}$, or more generally, whether $\#\mathrm{P} \subseteq \mathrm{FPH} = \bigcup_{k=1}^{\infty} \mathrm{F}\Delta_k^{\mathrm{P}}$?

This problem remains open, although it is known that there exist oracles for which #P is not contained in $\mathrm{F}\Delta_2^{\mathrm{P}}$[235]. Indeed, something much stronger can be said: there are relativized worlds in which there exist problems in #P that cannot even be *well-approximated* within $\mathrm{F}\Delta_2^{\mathrm{P}}$, in the sense of being computed to within a constant factor r for some r. This result does not extend any farther up in the hierarchy however. For any fixed ε and d, and any function f in #P, there is a function in $\mathrm{F}\Delta_3^{\mathrm{P}}$ that approximates f to within a factor of $1 + \varepsilon|x|^{-d}$, where $|x|$ is the input length [235]. Note, however, that this is a far cry from saying that #P is actually contained in $\mathrm{F}\Delta_3^{\mathrm{P}}$, and current betting is that in fact #P is not even contained in FPH.

The other possibility, that #P in a sense dominates the polynomial hierarchy, is more appealing, and a recent result indicates a precise sense in which it is true. This result, due to Toda [240] concerns languages rather than functions, and so requires an appropriate stand-in for #P rather than one for PH. The stand-in chosen is the natural one of the "#P-easy" languages, i.e., the class $\mathrm{P}^{\#\mathrm{P}}$ of languages recognizable in polynomial time with an oracle to a problem in #P.

6. THEOREM (Toda [240]). $\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$.

Equality between PH and $\mathrm{P}^{\#\mathrm{P}}$ seems unlikely. The two classes are unequal in almost all relativized worlds, as a technical consequence of the proofs mentioned in Section 2.6

that PH $\neq$ PSPACE relative to a random oracle [13, 47]. Furthermore, an immediate consequence of Theorem 6 is that $P^{\#P}$ cannot be contained in PH unless the polynomial hierarchy collapses. The same statement holds for an apparently much simpler class than #P, independently introduced by [93] and [196] and defined as follows.

DEFINITION. A *parity Turing machine* ($\oplus$TM) is an NDTM that accepts a given input string x if and only if the number of accepting computations for that input is odd.

DEFINITION. The class $\oplus$P ("*parity*-P") is the set of all languages accepted by polynomial-time $\oplus$TMs.

As one might expect, a typical example of a $\oplus$P-complete problem is $\oplus$SAT (given an instance I of satisfiability, is the number of satisfying truth assignments for I even?). Superficially, this would seem possibly to be a simpler problem than actually counting the number of satisfying assignments (#SAT), and so we might assume that $\oplus$P is a less powerful class than #P. However, just as determining the parity of the optimal travelling salesman tour's length proved to be just as hard as computing that length (both problems are complete for $F\Delta_2^P$, as seen in Section 2.4), here too one does not lose much by settling for the parity of x rather than x itself. In [240], it is shown that there is a randomized sense in which PH is $\oplus$P-easy, and that consequently $\oplus$P cannot be in PH unless PH collapses. We shall cover the concept of randomized reduction in Section 4.4, and will have more to say about $\oplus$P both there and in Section 4.2.

We conclude this section with a brief discussion of two more classes whose definitions can be related to that of #P. Suppose our NDTMs are augmented with output tapes, the contents of which are interpreted as (binary) integers. Such machines can be interpreted as computing functions in a variety of ways. It is not difficult to see that #P is simply the set of functions computable by taking the sum of the output values over all accepting computations of a polynomial-time NDTM. Two other natural options are to take the maximum value or to take the number of distinct values. These give rise to the following two classes, defined respectively in [163] and [161].

DEFINITION. The class OptP is the set of all functions computable by taking the maximum of the output values over all accepting computations of a polynomial-time NDTM.

DEFINITION. The class span-P is the set of all functions computable as $|S|$, where S is the set of output values generated by the accepting computations of a polynomial-time NDTM.

It is not difficult to see that $\text{OptP} \subseteq \text{FP}^{\#P}$ (and indeed that $\text{OptP} \subseteq \text{FP}^{\text{NP}} = F\Delta_2^P$), given that the maximum can be determined by binary search. Thus it is likely that OptP is strictly weaker than #P. It does, however, capture essentially the full power of $F\Delta_2^P$. Although it is a proper subclass, given that it can only contain functions rather than arbitrary search problems, any problem complete for OptP (under the "metric reductions" to which we alluded in Section 2.4) is complete for all of $F\Delta_2^P$ [163]. An example (also mentioned in Section 2.4) is the problem of computing the length of an optimal travelling salesman tour.

Whereas OptP seems to be less powerful than #P, this is not the case for span-P. It is not difficult to show that both #P and OptP are contained in span-P[161]. Moreover, there is evidence supporting the contention that span-P *properly* contains #P, but for that we will need the definitions of the next section.

4.2. *Unambiguous Turing machines and the classes FUP and UP*

In this section we consider yet another way in which NDTMs with output tapes can be considered to compute functions. Rather than worry about how to combine the results of multiple accepting computations, let us require that there be only one! The following definition is from [242].

DEFINITION. An *unambiguous Turing machine* (UTM) is an NDTM that, for each possible input string, has at most one accepting computation.

Such a machine, if allowed an output tape, provides a unique output for every string it accepts, and hence can be viewed as computing a function over the domain of the language that it recognizes.

DEFINITION. FUP (UP) is the class of all partial functions computable (all languages recognizable) by polynomial-time UTMs.

(This class is called "UPSV" in [99].) There are a variety of functions that, although not at present known to be in FP, are known to be in FUP. For a primary example, consider the *discrete logarithm*. In its simplest formulation, this problem can be given as follows.

DISCRETE LOG

Instance: Given a prime p, a primitive root a modulo p, and an integer b, $0<b<p$.

Answer: The discrete logarithm of b with respect to p and a, i.e., the (unique) integer c, $0 \leqslant c < p$, such that $a^c = b \bmod p$.

This problem, as stated, is almost, but not quite, in FUP. The answer is unique, but unfortunately the domain is not known to be in UP (which is required by definition). Although one can guess short proofs that substantiate primality and primitivity [199], no way of guessing *unique* proofs is known. Fortunately, the following augmented version of DISCRETE LOG, also not known to be in FP, *can* be computed by a polynomial-time UTM: given p, a, and b as before, together with short proofs that p is prime and a is a primitive root, compute the discrete log of b with respect to p and a.

Functions like (augmented) DISCRETE LOG that potentially lie in FUP$-$FP have cryptographic significance. The existence of such functions can be shown to be equivalent to that holy grail of modern cryptography theory, the "one-way function".

There are a variety of definitions of this concept, some stronger than others. The following captures about the minimum one would wish to have.

DEFINITION. A (partial) function f is (*weakly*) *one-way* if it satisfies the following three properties:

(1) f is "honest" (i.e., for all x in the domain of f, the string $f(x)$ can be no more than polynomially smaller than the string x),
(2) $f \in \mathrm{FP}$, and
(3) f^{-1} is not in FP.

Note that this definition of "one-way" requires only that the inverse be difficult in the worst-case. We have introduced the qualifier "weakly" so as to contrast this notion with the stronger one of [256] that we shall discuss briefly at the end of Section 4.3.

The discrete logarithm (or more precisely, its inverse) is currently one of the more popular candidates for the role of one-way function (according to both weak and strong definitions). Many current cryptographic schemes are based on presumed difficulty of the problem of computing c given p, a, and b (see [209]). At present, however, the only evidence we have of this difficulty is the fact that no one has as yet found an efficient algorithm for the problem.

A proof that a one-way function exists would be a major event, of course, as it would imply $\mathrm{P} \neq \mathrm{NP}$. The question on one-way function existence, however, can be even more tightly tied to a simple question about UTMs. Note that by definition $\mathrm{P} \subseteq \mathrm{UP} \subseteq \mathrm{NP}$, and it is not difficult to see that $\mathrm{FUP} = \mathrm{FP}$ if and only if $\mathrm{UP} = \mathrm{P}$. It has been shown in [99] that one-way functions exist if and only if $\mathrm{P} \neq \mathrm{UP}$. (One-way functions whose range is in P exist if and only if $\mathrm{P} \neq \mathrm{UP} \cap \text{co-UP}$.) Thus we might still hope to get strong candidates for one-way functions if we could identify complete problems for UP. Unfortunately, this seems unlikely: there are oracles for which UP has no such complete problems, either under polynomial transformations [108] or the more general polynomial-time Turing reductions [117]. The question of whether $\mathrm{UP} = \mathrm{NP}$ also has interesting connections to other areas, in this case to the classes of the previous section. In [161] it is shown that $\text{span-P} = \#\mathrm{P}$ if and only if $\mathrm{UP} = \mathrm{NP}$.

As might be expected, there exist oracles for all four possible relations between P, UP and NP: $\mathrm{P} = \mathrm{UP} = \mathrm{NP}$, $\mathrm{P} \neq \mathrm{UP} \neq \mathrm{NP}$, $\mathrm{P} = \mathrm{UP} \neq \mathrm{NP}$, and $\mathrm{P} \neq \mathrm{UP} = \mathrm{NP}$ [202, 88]. There is also an oracle such that $\mathrm{P} \neq \mathrm{UP} \cap \text{co-UP}$ [108]. Relative to a random oracle the containments $\mathrm{P} \subseteq \mathrm{UP} \subseteq \mathrm{NP}$ are all strict [28, 211].

Reference [28] actually is addressed toward more detailed considerations. It addresses the interesting question of whether one gets a bigger class of languages every time one increases the allowable number of accepting computations.

DEFINITION. For each $k > 0$, the class UP_k is the set of languages recognizable by NDTMs that always have k or fewer accepting computations.

Note that $\mathrm{UP} = \mathrm{UP}_1$. It is not known whether there is any k such that UP_k is strictly contained in UP_{k+1}. (If there were, then we would have $\mathrm{P} \neq \mathrm{NP}$.) It is shown in [28] however, that with respect to a random oracle, strict containment holds for all $k \geqslant 1$.

Moreover, for random oracles we also have that $\bigcup_{k\geq 1} \mathrm{UP}_k$ is strictly contained in the following natural class, first introduced, under a slightly different name, in [9].

DEFINITION. The class FewP is the set of all languages recognizable by polynomial-time NDTMs for which the number of accepting computations is bounded by a fixed polynomial in the size of the input.

Note that FewP is contained in NP by definition. A less obvious (and more interesting) containment is the following. Recall the class $\oplus$P introduced in the previous section, defined in terms of NDTMs whose answers are determined by the parity of the number of accepting computations. Although there are indications that this is a very powerful class, at present we do not even know if $\oplus$P contains NP. (Oracles exist for which the two classes are incomparable [241].) In [51] it is proved that $\oplus$P does contain FewP.

See Fig. 4 for a schema of the relations between the classes of decision problems introduced in this and the previous section. For more on the classes UP and FewP, and the relation of the former to cryptography, see the above references, and [133, 160].

4.3. *Random Turing machines and the classes R, co-R, and ZPP*

In the previous section, we considered NDTMs that had either no accepting computations or exactly one. The restriction that yields the classes highlighted in this section is somewhat different, but has a similar flavor. In the definitions that follow, we continue to assume, as in Section 1.3, that our NDTMs are normalized so that all computations are finite and have the same length. Now, however, we also assume (without loss of generality) that every internal node in the computation tree has either one or two successors, and that every computation path has the same number of "branch points", i.e., nodes with two successors.

DEFINITION. A *random Turing machine* (RTM) is an NDTM such that, for each possible input string, either there are no accepting computations or else at least half of all computations are accepting.

DEFINITION. The class R consists of all decision problems solved by polynomial-time RTMs.

Here "R" stands for "random polynomial time", a terminology introduced in [3]. (Some writers use "RP" to denote the same class.) Note that $\mathrm{P} \subseteq \mathrm{R} \subseteq \mathrm{NP}$. The term "random" comes from the following observation. Suppose we attempted to simulate the operation of a given NDTM on a given input as follows: Starting at the initial configuration we proceed deterministically until we reach a branch point. At each branch point, we randomly pick one of the two alternatives for the next move, and then proceed. Given our assumption that all computations of the NDTM contain the same number of branch points, the probability we will end up in an accept state is simply the ratio of the number of accepting computations to the total number of computations.

Thus for a problem in R, the probability that such a simulation will mistakenly answer "no" (i.e., fail to accept) when the answer is "yes" is 0.5 or less. Note that, as with the definition of NP, this definition is one-sided; the probability that we will answer "yes" when the answer is "no" is 0.

A fortunate side-effect of this one-sidedness is that we can arbitrarily increase the probability of correctness by repeating the experiment. If we perform k independent random simulations and ever find an accepting computation, we know the answer is "yes", otherwise the probability that the answer is "yes" is at most $1/2^k$. Even for moderate values of k, say $k=50$, this can be less than the probability of a machine error in our computer, and so we will be fairly safe in asserting that the answer is "no".

Thus, computationally speaking, it is almost as good for a problem to be in R as for it to be in P. Moreover, the possibility that $\text{R}=\text{P}$ has not been ruled out. There are oracles for which the two classes are distinct and either $\text{R}\neq\text{NP}$ or $\text{R}=\text{UP}=\text{NP}$ [202]. There are also oracles for which all of P, UP, R, and NP are all distinct [88]. Relative to a random oracle however, $\text{P}=\text{R}\neq\text{NP}$ [29].

Several important problems, not known to be in P, are known either to be in R or co-R (the complements of decision problems in R, where it is the "yes" answers that may be false, although only with probability 0.5 or less). For a first example, consider the following.

PRODUCT POLYNOMIAL INEQUIVALENCE

Instance: Two collections $P=\{P_1,\ldots,P_n\}$ and $Q=\{Q_1,\ldots,Q_m\}$ of multivariate polynomials over the rationals, each polynomial represented by listing its terms with non-zero coefficients.

Answer: "Yes" if $\prod_{i=1}^{n} P_i$ and $\prod_{j=1}^{m} Q_j$ are different polynomials.

This problem is not known to be in P. We cannot simply multiply together each collection and compare the results, as such multiplications may result in an exponential blow-up in the number of terms. Nor can we factor the individual polynomials and compare the two lists of irreducible factors, since the irreducible factors of a polynomial can also have exponentially more terms than the original. There is, however, a simple randomized test for inequivalence of the two products, suggested by Schwartz in [221], that runs in polynomial time. One merely chooses a set of values for the arguments in an appropriate random fashion, *evaluates* each of the P_is and Q_is with these values substituted in, and then multiplies the two resulting collections of rational numbers together, thus obtaining the values of the two product polynomials at the given arguments. Schwartz shows that the evaluated products must differ at least half the time if the product polynomials are inequivalent. (They of course cannot differ if the product polynomials are equivalent.) The above "randomized algorithm" thus can serve as a basis for a polynomial-time RTM that solves PRODUCT POLYNOMIAL INEQUIVALENCE, which consequently is in R.

A second, although as we shall see possible weaker, candidate for membership in $\text{R}-\text{P}$ is COMPOSITE NUMBER. As has been observed in [201, 233], a positive integer n is composite if and only if more than half of the integers b, $1\leqslant b<n$, are "witnesses" to this in a technical sense that can be verified in polynomial time given just n and b. The

following "randomized algorithm" can thus serve as the basis for a polynomial-time RTM that solves COMPOSITE NUMBER: Pick a random integer b between 1 and n and say yes if and only if b is a witness to the compositeness of n. The total time is $O(\log^3 n)$, i.e., polynomial in the size of the input number n. Thus COMPOSITE NUMBER is in R (and its complementary problem, PRIME NUMBER, is in co-R). COMPOSITE NUMBER is a weaker candidate for membership in $R-P$ than PRODUCT POLYNOMIAL INEQUIVALENCE for two reasons. The first is the already cited result of [188] that, if the Extended Riemann Hypothesis holds, then COMPOSITE NUMBER $\in$ P. The second is the recent discovery claimed in [2] (and building strongly on results in [95]) that PRIME NUMBER is also in R, thus placing both the PRIME and COMPOSITE NUMBER problems in the elite class "ZPP".

DEFINITION. $\mathrm{ZPP} = \mathrm{R} \cap \text{co-R}$.

We refer to ZPP as an "elite class" because it also equals the set of those decision problems that can be solved by randomized algorithms that *always* give the correct answer and run in expected polynomial time. Note that every problem in ZPP is solvable by such an algorithm: One simply runs both the R and co-R algorithms for the problem repeatedly until one discovers a witness to the answer. The expected number of iterations is so small that the expected running time is proportional to that of the slower of the R and co-R algorithms. (In the case of COMPOSITE NUMBER, this unfortunately may be as bad as $\Omega(n^{100})$; the Adleman and Huang result [2] is of mostly theoretical interest.) Conversely, if a decision problem can be solved by a randomized algorithm that runs in expected polynomial time and always gives the correct answer, that problem must be in $\mathrm{R} \cap \text{co-R}$: Associated with any such expected polynomial-time algorithm is a polynomial p such that, for any input x, the probability that the algorithm will halt before $p(|x|)$ steps have been taken exceeds 0.5. The required RTMs can be constructed by simulating the algorithm for $p(|x|)$ steps.

Another characterization of ZPP is that it consists of all those problems solvable by polynomial-time "Las Vegas" algorithms. This term, introduced in [262], refers to randomized algorithms which either give the correct answer or no answer at all (and for which the latter option occurs less than half the time). This is as opposed to "Monte Carlo" algorithms, where "Monte Carlo" is typically used as a generic term for "randomized". (See [132] for a more extended discussion of these terms, as well as more on probabilistic classes in general.)

See Fig. 5 for a schema relating R, co-R, and ZPP to the major nonrandomized classes presented earlier (as well as to some additional randomized classes that will be introduced in Section 4.5). There are many who believe that membership in ZPP is a strong hint of membership in P. Indeed, relative to a random oracle, $\mathrm{P} = \mathrm{ZPP} = \mathrm{R}$, with all properly contained in NP [29]. Oracles do exist, however, for which $\mathrm{P} \neq \mathrm{ZPP}$ [123] and $\mathrm{ZPP} \neq \mathrm{R}$ [21].

Returning to the question of R versus P, observe that although we have presented potential examples of problems in $R-P$, we have provided no evidence that these are the most likely examples. That is, we have not identified any "R-complete problems". The fact is, we do not know of any, and it is unlikely that we shall find any soon. As is the case for $\mathrm{NP} \cap \text{co-NP}$, there are oracles such that R contains no problems that are complete for it under polynomial transformations [228] or under polynomial-time Turing reductions [119].

We conclude this section by discussing ways in which R appears to differ from NP. As we have seen, although there are oracles such that $R = NP$, the two classes are distinct for a random oracle. Further evidence in favor of the proper containment of R in NP comes from the fact, pointed out in [1], that R has "small circuits". More precisely, R is contained in the following class, defined in terms of the Boolean circuit families of Section 1.3.

DEFINITION. The class P/poly consists of all those languages recognizable by (not necessarily uniform) families of polynomial-size circuits.

In other words, for any problem X in P/poly, there is a polynomial p_X with the following property: for any instance size n, there is a Boolean circuit B_n with n inputs and $O(p_X(n))$ gates that correctly outputs X's answer for all instances of size n. According to the definition, there need be no uniform way of generating the circuits B_n in time polynomial in n, and the proof in [1] that $R \subseteq P/\text{poly}$ takes advantage of this. That is, it proves that the circuits exist, but does not show how to construct them. Thus it does not also imply that $R = P$. It does, however, provide strong evidence that $R \neq NP$. Although nontrivial oracles exist for which $NP \subseteq P/\text{poly}$ (i.e., ones for which we also have $P \neq NP$) [126], such a containment would imply that the polynomial hierarchy collapses into Σ_2^p [152].

As a digression, we should point out that P/poly has an alternative definition, this one in terms of Turing machines with "advice".

DEFINITION. An *advice-taking Turing machine* is a Turing machine that has associated with it a special "advice oracle", one that is a (not necessarily recursive) function $A: Z^+ \to \{0,1\}^*$. On input x, a special "advice tape" is automatically loaded with $A(|x|)$ and from then on the computation proceeds as normal, based on the two inputs, x and $A(|x|)$.

DEFINITION. An advice-taking TM uses *polynomial advice* if its advice oracle A satisfies $|A(n)| \leqslant p(n)$ for some fixed polynomial p and all nonnegative integers n.

DEFINITION. If X is a class of languages defined in terms of resource-bounded TMs, then X/poly is the class of languages defined by TMs with the same resource bounds but augmented by polynomial advice.

Since the "advice" for an instance of size n can be the description of a polynomial size, n-input Boolean circuit, it is clear that a language in P/poly according to the circuit family definition is also in it according to the advice definition; the converse is almost as immediate.

Other results about polynomial advice from [152] include

(a) $\text{PSPACE} \subseteq P/\text{poly}$ implies $\text{PSPACE} \subseteq \Sigma_2^p \cap \Pi_2^p$ (and hence the polynomial hierarchy collapses and equals PSPACE);

(b) EXPTIME$\subseteq$P/poly implies EXPTIME$=\Sigma_2^p$ (and hence P$\neq$NP); and
(c) EXPTIME$\subseteq$PSPACE/poly implies EXPTIME$=$PSPACE.
The proof techniques used in these results are elaborated upon in [18, 19].

We conclude this section by alluding to an additional result that can be viewed as supporting the conjecture that R is strictly contained in NP. This is a result of A.C. Yao in [256] linking the complexity of R to the existence of "strong" one-way functions. We omit the technical details of the latter definition here, but suffice it to say that, whereas the "one-way functions" defined in the previous section were only difficult to invert in a worst-case sense, strong one-way functions are also difficult to invert "on average" (and even for nonuniform circuits). This is obviously a more useful cryptographic property, and many theoretical encryption schemes are based on the assumption that such functions exist (and indeed, that the DISCRETE LOG problem of the previous section provides one). However, if we assume that such functions exist, and also that NP-complete problems take exponential time (a slightly stronger assumption than P$\neq$NP, but a common one), then Yao's result implies that R is strictly contained in NP. He shows that, assuming that strong one-way functions exist, we must have R$\subseteq\bigcap_{\varepsilon>0}$DTIME$(2^{n^\varepsilon})$.

4.4. *Randomized reductions and NP*

When we discussed the class NP previously in Section 2.1, we mentioned several different classes of transformations with respect to which problems could be proved "complete" for NP, and indicated that there would be more in a later section, once the appropriate groundwork had been laid. The introduction of random Turing machines in the previous section has laid that groundwork, and in this section we discuss three new forms of reduction, based on variants of random Turing machines, and the kinds of "completeness for NP" that they yield. Although other variants are possible, these three have to date been the most successful.

Each can be viewed as a variant to the γ-reduction described in Section 2.2, and all include polynomial transformations as a special case. With each, we reduce a decision problem X to decision problem Y by means of a polynomial-time NDTM that yields an output for each accepting computation and satisfies certain properties. The property for γ-reductions was that there be at least one accepting computation for each string x, and for each output y, y had the same answer in Y as x had in X. Here are properties that our three randomized reductions must satisfy for any given input string x:

(1) *Random reduction* (R-reduction) [3]: (a) At least half the computations are accepting, and (b) for all outputs y, y has the same answer in Y as x has in X. (Note that every R-reduction is thus also a γ-reduction, something that cannot be said for the following two types of reduction.)

(2) *Unfaithful random reduction* (UR-reduction) [4]: (a) All computations are accepting, (b) the outputs must be "faithful" for yes-instances, i.e., if the answer for x in X is "yes", then all outputs y have answer "yes" in Y, and (c) correct outputs must be "abundant" for no-instances, i.e., if the answer for x in X is "no", then at least $1/p(|x|)$ of the outputs y have answer "no" in Y, where p is a fixed polynomial.

(This definition is generalized somewhat from that given in [4], for which "abundant" meant "at least half". The generalization allows us to include that UR-reductions, like the other two types, are transitive.)

(3) *Reverse unfaithful random reduction* (RUR-reduction) [245]: Same as for UR-reductions, except now the outputs must be faithful for no-instances and correct outputs must be abundant for yes-instances. (This reduction goes unnamed in [245]; the above name was the best we could come up with on short notice.)

The properties that make these reductions useful are as follows:

(1) If X is hard for NP under R-reductions, then $X \in$ ZPP implies NP = ZPP and $X \in$ R implies NP = R.

(2) If X is hard for NP under UR-reductions, then $X \in$ co-R implies NP = ZPP.

(3) If X is hard for NP under RUR-reductions, then $X \in$ R implies NP = R.

Note that in all three cases we have at the very least that if X is in ZPP then NP = R. Thus, although proving that a problem is complete (or simply hard) for NP under any of the above three types of reduction is not as strong an argument for intractability as proving it NP-complete, such a proof still provides believable evidence that the problem cannot be solved in either polynomial time or polynomial expected time. We conclude this section with examples of complete problems of each type.

For a problem that is complete for NP under R-reductions, consider the following: given positive integers a, b, and c, where a is a power of 2, are there positive integers x and y such that $axy + by = c$? In [3] this problem is shown to be complete for NP under R-reductions, assuming the Extended Riemann Hypothesis. The latter assumption was used only for certifying primality however, so the new result that PRIME NUMBER is in R allows us to dispense with that hypothesis. This problem also turns out to be complete for NP under UR-reductions [4], and so it can be in neither R nor co-R without dire consequences.

A second, less number-theoretic example of a problem complete for NP under UR-reductions is the following problem, from [248]: ENCODING BY DTM: Given two strings x,y over $\{0, 1\}$ and an integer K, is there a DTM with K or fewer states that, when started with x on its worktape and its read-write head located at the leftmost symbol of x, writes y on its output tape in just $|y|$ steps?

RUR-reductions were introduced in [245] for the purpose of showing that the following variant on SATISFIABILITY, called "UPSAT" in [133] is hard: If there are no satisfying truth assignments, the answer is "no". If there is exactly one, the answer is "yes". If there are more than one satisfying truth assignments, then both "yes" and "no" are valid answers. (Note that this is not the same as the UNIQUE SATISFIABILITY problem described in Section 2.4.) Although UPSAT is not a decision problem as defined, it can be turned into one by specifying a particular answer, "yes" or "no", for each instance with more than one satisfying truth assignment. In [245] it was shown that all such restrictions are RUR-hard for NP. As a consequence (assuming R $\neq$ NP), SATISFIABILITY would remain hard even if one could somehow be "promised" that no instances with more than one satisfying truth assignment would ever arise. (For a fuller discussion of such "promise" problems and how to reason about them, see [73, 133, 245].)

An important second example of the use of RUR-reductions concerns the $\oplus$SAT problem of Section 4.1, which is also shown to be RUR-hard for NP in [245]. It was this result that was generalized in [240] to prove that PH is ⊕P-easy in a randomized sense, or equivalently, that ⊕SAT is hard for PH under randomized reductions. (The actual reductions used in [240] are of yet a new type, and might be called "BPP-reductions". We will not define them here, but the interested reader should be able to deduce the definition after reading the next section.)

4.5. *Probabilistic Turing machines and the classes PP and BPP*

In the previous two sections we were for the most part concerned with "randomized" computations that used different criteria for the answers "yes" and "no". In this section we consider the situation when the two answers are treated symmetrically. For the definitions we are about to provide, we continue to assume as before that our NDTMs are normalized so that all branch points have outdegree 2, and all computations terminate in the same number of steps and contain the same number of branch points. We also assume that all computations terminate in either a "yes" or a "no" state.

DEFINITION. A *probabilistic Turing machine* (a PTM) is an NDTM whose output for a given input string x is "yes" if more than half of the computations terminate in "yes" states, and is "no" if more than half of the computations terminate in "no" states. (If the number of yes-computations equals the number of no-computations, the output is "don't know".)

DEFINITION. A PTM *solves* a problem X if and only if the PTM outputs the correct answer for each instance of the problem (and never claims ignorance).

Note that if, as before, we imagine ourselves as simulating such a PTM by making random choices at each branch point, the probability that we obtain the correct answer will by definition always exceed 0.5.

The first complexity class to be derived using these definitions is due to Gill [90].

DEFINITION. The class PP is the set of all decision problems that can be solved by polynomial-time PTMs.

Note that this class does not have the positive practical advantages of R, since a polynomial number of iterations of a PP algorithm may not be able to increase our confidence in the answer to worthwhile levels. For instance, it may be that the answer is correct only with probability $\frac{1}{2}+(\frac{1}{2})^n$, in which case we cannot reduce the error probability below $\frac{1}{4}$ without an exponential number of iterations. Indeed, it is not difficult to show that PP contains $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$, and hence NP, co-NP, and many presumably intractable problems [28]. PP is, however, contained in PSPACE. (Relative to a random oracle, both containments PP $\subseteq$ PSPACE and NP$\cup$co-NP $\subseteq$ PP are proper [29].)

Unlike R, the class PP is known to have complete problems under polynomial transformations; in the canonical one we are given an instance of SATISFIABILITY and asked if more than half of the possible truth assignments satisfy all the clauses [90, 225]. Note the close relationship between this problem and that of actually counting the number of satisfying truth assignments, the problem that was observed to be complete for #P in Section 4.1. Indeed, it is easy to show that $P^{PP} = P^{\#P}$, i.e. an oracle for a problem in PP is just as good as an oracle for a problem in #P [10]. In light of Toda's result, mentioned in Section 4.1, that $PH \subseteq P^{\#P}$ [240], this implies that PH is also contained in P^{PP}, a corollary of which is that PP cannot lie in the polynomial hierarchy unless the hierarchy collapses.

Although PP does not of itself seem to guarantee useful randomized algorithms, there is an important subclass that maintains its symmetric nature and is capable of providing such algorithms. This class contains R∪co-R and can be viewed as the most general class of "efficiently solvable" problems.

DEFINITION. The class BPP is the set of all decision problems solvable by polynomial-time PTMs in which the answer always has probability at least $\frac{1}{2} + \delta$ of being correct, for some fixed $\delta > 0$.

The "B" in BPP stands for "bounded away from $\frac{1}{2}$". Note that for randomized algorithms based on such PTMs, the probability of correctness can be rapidly increased by iteration; ineed we can simplify the definition without loss of generality by replacing "$\frac{1}{2} + \delta$" by $\frac{2}{3}$.

BPP is an interesting class. By definition we have $R \cup \text{co-}R \subseteq BPP \subseteq PP$, but currently we know nothing definite about the inclusion relations, if any, between NP and BPP. What we do know is the following: If $NP \subseteq BPP$, then $R = NP$ and, in addition, the polynomial hierarchy must collapse to BPP [158, 259]. One might thus conjecture that $BPP \subseteq NP$, especially since for a random oracle we have $P = ZPP = R = BPP \subseteq NP$, with the latter containment proper [29]. Moreover, like R (and for much the same reason, e.g., see [20]), BPP has "small circuits", i.e., $BPP \subseteq P/\text{poly}$ as defined in Section 4.3. There are oracles, however, for which BPP is not contained in NP, indeed, is not even contained in Δ_2^P [235]. BPP can, however, be shown to lie within the polynomial hierarchy, in fact in $\Sigma_2^P \cap \Pi_2^P$ [171, 230] (see Fig. 5).

The current consensus seems to be that the two sets NP and BPP are incomparable (as well as the two sets NP∪co-NP and BPP). Current candidates for membership in NP − BPP include the NP-complete problems. There are also copious candidates for BPP − NP, since BPP, being symmetric, contains both R and co-R. (The class co-R does not appear to be contained in NP, although it is contained in co-NP.) Thus we can get a candidate for BPP − NP simply by choosing a problem in co-R that is not known to be in NP, such as PRODUCT POLYNOMIAL EQUIVALENCE (the complement of the inequivalence problem introduced in the previous section). Analogous candidates for BPP − (NP∪co-NP) are harder to come by. Indeed all problems that have to date been identified as members of BPP are actually members of NP∪co-NP, although not necessarily in R∪co-R. (See [16] for examples of number-theoretic problems in NP that are candidates for BPP − (R∪co-R).) The hope that complete problems for BPP might offer candidates for BPP − (NP∪co-NP) is somewhat dim, given that there are oracles for which BPP has no complete problems (under polynomial transformations [108] or polynomial-time Turing reductions [117]). For more on BPP, see aso [260, 261], the former reference providing further insights into the classes of Sections 4.3 and 4.6 as well.

As a final class definable by symmetric radomized computations, let us briefly consider the class PPSPACE of decision problems solvable by polynomial *space* bounded PTMs. This clearly contains PSPACE. Surprisingly, as proved in [226], it is also contained in PSPACE, and hence is identical to it!

4.6. *Stochastic Turing machines, interactive proofs, and the classes they define*

So far in Section 4 we have seen counting Turing machines (CTMs), unambiguous Turing machines (UTMs), random Turing machines (RTMs) and probabilistic Turing machines (PTMs). In this section we introduce one final variant, the *stochastic* Turing machine of [190]. This will be a hybrid between a probabilistic Turing machine and an alternating Turing machine. As usual, we shall define it in terms of an NDTM computation tree. For simplicity in our definition, we assume that all configurations (except the leaves) have outdegree 2 in the computation tree, that all computation paths are of the same length, and that they all contain the same number of non-leaf nodes.

Definition. A *stochastic Turing machine* (STM) is an NDTM whose output for a given input string x is specified as follows: As in an ATM, each configuration in the computation tree is identified as one of two types; this time, however, the types are "existential" and "random", rather than "existential" and "universal". We assume the types alternate from level to level. An *admissible computation* for such a tree is a subtree obtained by deleting one of the two subtrees hanging from each existential node. The output for input string x is "yes" if and only if the resulting computation tree contains an admissible subtree in which more than half the leaves are accepting.

As with ATMs we can view the computation of an STM as corresponding to a game, but here the game will be between a normal player and an "indifferent opponent", one who simply makes random moves, choosing with equal probability between the potential immediate successors. The answer for input x is "yes" if and only if there is a strategy under which the existential player has a probability greater than 0.5 of winning against his random opponent. We say that an STM solves a given decision problem X if the existential player has such a strategy for every yes-instance of X, but no such strategy for any no-instance of X.

The name "PPSPACE" was recycled in [190] to denote the class of decision problems solvable by polynomial-time STMs (the "time" in the resource restriction became "SPACE" in the class name because of the alternation involved in the machine model). As with the PPSPACE we saw in the previous section however, the name is not important. This PPSPACE turns out to be identical to ordinary PSPACE, as did its predecessor. Its main advantage is hence as a means for showing interesting new types of problems to be PSPACE-complete, such as certain scheduling problems where the task lengths vary according to a Poisson process, as well as problems from control theory [190]. One simple-to-understand example is the following variant on the #P-equivalent GRAPH RELIABILITY problem of Section 4.1.

DYNAMIC GRAPH RELIABILITY

Instance: Directed graph $G(V, A)$ with specified source and sink vertices s and t. For each pair v, a, where v is a vertex and a an arc, a rational number $p(v, a) \in [0, 1]$.

Answer: "Yes" if there is a strategy by which you can, starting at s, reach the destination t with probability exceeding $\frac{1}{2}$, assuming your travels are subject to the following rules: Your traversal proceeds in steps, where in each step you leave your current vertex and travel along a (still-existent) outgoing arc to an adjacent vertex. Initially all arcs in A exist and are traversable, but the arcs fail (disappear permanently) according to a random process that occurs while your traversal proceeds, with the probability that a given arc a disappears during a step that you started at vertex v being $p(v, a)$.

Here the "random opponent" is the process generating the arc failures. Note how much more game-theoretic this problem is than the original GRAPH RELIABILITY problem, in which one could view the random arc failures as all happening at the beginning of the process, after which one could count on the remaining arcs to persist, and thus could easily walk from s to t, assuming a path still existed.

Although the above first attempt at using polynomial-time STMs to define a new complexity class failed, subsequent attempts have been substantially more productive. These have generated new classes by imposing further restrictions on the STM computations, just as we restricted PTMs in order to define BPP. A first such class is the following (its name will be explained below).

DEFINITION. The class AM[poly] consists of all decision problems solvable by polynomial-time STMs satisfying the restriction that, for any input x, the existential player's best strategy yields a winning probability that either exceeds $\frac{2}{3}$ or is less than $\frac{1}{3}$.

This class (and its name) was introduced in [12], which provided an alternative definition in terms of conversations between an all-knowing wizard "Merlin" and a skeptical listener "Arthur", who has polynomial-time bounded computational resources and the ability to flip unbiased coins. Merlin's goal is to convince Arthur "beyond a reasonable doubt" that a given string x is a yes-instance of decision problem X. When Arthur speaks, he is limited to simply telling Merlin the outcome of some number of coin flips (polynomial in $|x|$). When Merlin speaks, his message (also polynomial in $|x|$) can depend on x and all the previous messages. The conversation is allowed to run for a polynomial number of interchanges, after which Arthur inputs x and a transcript of the conversation to a deterministic polynomial-time algorithm, which tells him whether or not to believe that x is a yes-instance. Protocols for conversations of this sort correspond to STMs, and X is in AM[poly] if there is a protocol such that, for any yes-instance x, Merlin can with probability greater than $\frac{2}{3}$ convince Arthur of this fact, and for each no-instance the probability is less than $\frac{1}{3}$ that

Merlin can fool Arthur, no matter what Merlin does. (In what follows, we will say such a protocol "solves" X.) Note that, given this interpretation of AM[poly] in terms of STMs, we have AM[poly] $\subseteq$ PPSPACE $=$ PSPACE, and Merlin need not in fact have arbitrary computing power, but can settle for polynomial space.

Of particular interest are the subclasses of AM[poly] in which only a bounded number of messages are sent.

DEFINITION. For each $k > 0$, the class MA[k] consists of all those decision problems solvable by Arthur–Merlin protocols in which Merlin goes first and there are exactly k messages sent.

DEFINITION. For each $k > 0$, the class AM[k] consists of all those decision problems solvable by Arthur–Merlin protocols in which Arthur goes first and there are exactly k messages sent.

Note that MA[1] is simply the set of decision problems solvable by protocols in which the conversation begins and ends with Merlin's first transmission. This is the same as STMs all of whose configurations are existential, and clearly equals NP. Analogously, the class AM[1] consists of those decision problems solvable by protocols in which only Arthur speaks, and hence equals BPP. Things become a bit more interesting once we begin to allow some true interaction between Arthur and Merlin. As shorthands, we shall use the following notation, introduced in [12], for what turn out to be the two most important classes:

DEFINITION. MA $=$ MA[2]; AM $=$ AM[2].

The class MA, in which both parties speak once with Merlin first, can be viewed as a randomized version of NP, consisting of those problems for which all answers have short *probabilistic* proofs (i.e., proofs for which the validity problem is in BPP). The class AM, where Arthur speaks and Merlin responds, is also a generalization of NP, in that it contains precisely those languages that are in NP^B for almost all oracles B [189]. (The analogous result, that BPP consists of precisely those languages that are in P^B for almost all oracles B, was proved in [29].)

In [12], it is shown that NP $\subseteq$ MA $\subseteq$ AM. An oracle exists for which AM properly includes MA (and hence NP) [215]. A potential member of AM $-$ NP is given in [12]. This is the problem MATRIX GROUP EXACT ORDER: given a prime power $q = p^n$, integers k and m, and a collection C of $k \times k$ matrices over GF(q), does the matrix group generated by C have order equal to m, i.e., are there precisely m distinct matrices that can be obtained by multiplying together sequences of members of C (where repetitions are allowed and the product of the empty sequence is taken to be the identity matrix)? Although telling whether the order is divisible by a given integer is in NP [15], the question for the EXACT ORDER problem remains open, even though it is shown in [12] to be in AM (indeed, in AM $\cap$ co-AM).

Turning to the classes MA[k] and AM[k], $k > 2$, we have yet another potential hierarchy. In this case, however, something unexpected happens: the hierarchy collapses! For all $k > 2$, $MA[k] = AM[k] = AM$ ($= AM[2]$) [12]. The question of whether all of AM[poly] collapses to AM remains open, however, and their exist oracles for which it does not, indeed for which AM[poly] − PH is nonempty [6]. As to AM and MA, it can be shown that $AM \subseteq \Pi_2^p$ and that $MA \subseteq \Pi_2^p \cap \Sigma_2^p$ [12]. A final question is whether AM, which contains NP, also contains co-NP. This now seems unlikely, as it is shown in [39] that this would imply that the polynomial hierarchy collapses into AM. The class co-NP may not be in AM[poly] either: although no one has yet shown that PH would collapse if this occurred, there does exist an oracle under which containment fails to hold [80]. AM (and hence AM[poly]) does, however, contain one interesting problem in co-NP that is not known to be in NP: GRAPH NONISOMORPHISM [91]. (This is the complementary problem to the GRAPH ISOMORPHISM problem introduced in Section 1.1, with the answer being "yes" only if the two graph representation are *not* isomorphic.) (For new developments on AM[poly], see Section 6.1.)

Another surprising result compares the power of Arthur–Merlin protocols to the apparently much more general notion of an "interactive proof system" introduced in [96]. Such a system is also based on two players, one all powerful, one with a source of random bits and polynomial-time computational power. The distinction is that, while Merlin (now called "the prover") continues to be operating under the same constraints, Arthur (now called "the verifyer" and *always* sending the first message), can be cleverer. He no longer need simply send the random bits he generated. He may send the result of an arbitrary polynomial-time computation based on the input x, some new random bits, the transcript of the conversation so far, and the list of all random bits he previously generated.

The definition of "solving" a decision problem is the same for interactive proof systems as it was for Arthur–Merlin protocols. Note, however, that the fact that the verifyer can in effect keep "secrets" from the prover means that interactive proof systems cannot be modelled directly by STMs, as could Arthur–Merlin protocols. Let us thus define for interactive proof systems the analogs of the classes we had for Arthur–Merlin games.

DEFINITION. The class IP consists of all those decision problems solvable by interactive proof systems in which the total computation time of the verifyer is polynomially bounded, but there is otherwise no limit on the amount of interaction.

DEFINITION. For each $k > 0$, the class $IP[k]$ is the set of all decision problems solvable by interactive proof systems in which the total computation time of the verifyer is polynomially bounded and neither the prover nor the verifyer sends more than k messages.

Note that, by definition, $AM[poly] \subseteq IP$ and, for all $k \geq 1$, $AM[2k] \cup MA[2k] \subseteq IP[k]$ (and so $AM \subseteq IP[1]$). Surprisingly, all the extra power that the verifyer has in an interactive proof system (and all the work we just went to making the above definitions) is for naught. As shown in [97], $IP = AM[poly]$ and $IP[k] \subseteq AM$, for all $k \geq 1$ (so yet another potential hierarchy collapses to AM).

Also contained in AM (indeed, in AM$\cap$co-AM) is the set of all decision problems solvable by "perfect zero-knowledge" polynomial-time interactive proof systems, even when the number of rounds is unbounded [7, 79]. (The concept of a "zero-knowledge" proof system, which can convince a verifyer of a statement without revealing anything beyond the fact that the statement is true, was introduced in [96]. It is too involved to describe here, and so interested readers are directed to [137] and the references therein.)

For a schema of the classes introduced in this section, See Fig. 6.

5. Inside P

In Sections 2 through 4, we have concentrated on complexity classes that *contain* P, viewing equality with P as the most desirable possible outcome. It is not the case, however, that membership in P by itself ensures tractability in any practical sense. Thus researchers have of late devoted equal, if not more time to investigating classes that are in various senses "easier" than P (or at least incomparable). These classes are the subject of this final section. In contrast to the previous sections, our general order of traversal will be downwards rather than upwards, from larger classes to smaller ones.

5.1. *Classes defined by sublinear space bounds*: *POLYLOG-SPACE, L, NL, and SC*

With much of today's computing being done on personal computers that have limited main memory (and do not have sophisticated paging systems), the amount of memory an algorithm requires can often be more crucial than its running time. Consequently, there has been much interest in algorithms that use significantly less workspace than the size of their input (which can be read as needed off the floppy disk or input tape on which it is delivered). This ideal is captured by complexity theorists in the concept of the "log-space" and, less restrictively, the "polylog-space" DTM.

Definition. The class L consists of all decision problems solvable by DTMs with workspace bounded by $O(\log|x|)$ on input x.

Definition. The class POLYLOG-SPACE consists of all decision problems solvable with workspace bounded by $O(\log^k|x|)$ for some fixed k.

In the terminology of Section 2.6, L = DSPACE[O(log n)] and POLYLOG-SPACE = $\bigcup_{k>1}$DSPACE[$\log^k n$] (or DSPACE[$\log^{O(1)} n$] for short). It is immediate that L $\subseteq$ POLYLOG-SPACE, with the containment proper because of Theorem H2 of Section 1.4. What is more interesting are the relations between these two classes and P. Clearly L $\subseteq$ P, since no log-space DTM can have more than a polynomial number of distinct memory states. (Indeed, for any log-space bounded DTM, there will be a polynomial p such that for any input x, if the DTM runs longer than time $p(|x|)$ on input x, it will never halt.) For POLYLOG-SPACE and P, however, the situation is more complicated. In a result analogous to the one for NP and ETIME in Section 3.1, it can be shown [36] that POLYLOG-SPACE $\neq$ P, although we do not know whether the two classes are incomparable or one properly contains the other. This inequality follows from the answer to a question we have previously asked about other classes: can POLYLOG-SPACE have a complete problem (in this case under log-space transformations)? Surprisingly, the answer is "no", and not just in special relativized worlds, but for (unrelativized) POLYLOG-SPACE itself. Essentially, it can be shown that should such a complete problem exist, we would have POLYLOG-SPACE $\subseteq$ LOG^k-SPACE (the set of decision problems solvable by $O(\log^k|x|)$-space DTMs) for some fixed k, an impossibility given Theorem H2. But this means that POLYLOG-SPACE cannot equal P, since there are problems that *are* log-space complete for P, as we saw in Section 1.5.

Of the three possibilities for the relationship between POLYLOG-SPACE and P, most researchers would probably favor incomparability. There exist likely candidates for P $-$ POLYLOG-SPACE, for instance the problems that are log-space complete for P, since if any such problem is in POLYLOG-SPACE, then P $\subseteq$ POLYLOG-SPACE and in fact is contained in LOG^k-SPACE for some fixed k, an unlikely prospect. We have already seen two examples of such "P-complete" problems in Section 1.7: DTM ACCEPTANCE and LINEAR PROGRAMMING. Here are two more well-known examples. The first, proved P-complete in [168], can be viewed simply as a restatement of DTM ACCEPTANCE in terms of the Boolean circuit model of computation. The second, proved P-complete in [57], has an interesting graph-theoretic flavor and has been the setting for some intriguing lower bound arguments about algorithmic complexity.

CIRCUIT VALUE

Instance: A description of an n-input, 1-output Boolean circuit, together with an input value (0 or 1) for each of the input gates.

Answer: "Yes", if the output of the circuit on the given input is 1.

PATH SYSTEM ACCESSIBILITY

Instance: A finite set X of *nodes*, a relation $R \subseteq X \times X \times X$, and two sets $S, T \subseteq X$ of *source* and *terminal* nodes.

Answer: "Yes", if there is an "accessible" terminal node, where all sources are accessible, and if $(x, u, v) \in R$ and u and v are accessible, then x is accessible. (Note that this can be viewed as the question of whether a given straight-line program computes its claimed outputs, and is trivially in P.)

Although no one has yet been able to prove that P-complete problems require more than polylogarithmic space, there are some partial results in that direction for the latter problem. In particular, it has been shown to require $\Omega(|x|^{1/4})$ space under a fairly wide class of methods [60, 64].

Although POLYLOG-SPACE has no complete problems, one can come up with plausible candidates for POLYLOG-SPACE $-$ P by considering problems that are complete for subclasses of POLYLOG-SPACE, such as DSPACE$[O(\log^k n)]$ for a fixed $k > 1$, under log-space transformations. Note that the simple proof that L $\subseteq$ P does not carry over to DSPACE$[O(\log^k n)]$ for any $k > 1$ since for each such k a $\log^k|x|$-space DTM has a potentially superpolynomial number of memory states.

Complete problems for DSPACE[O($\log^k n$)], $k>1$, do exist, but unfortunately, at present we do not have any "natural" candidates. Instead, we must settle for the relevant analog of the DTM ACCEPTANCE problem of Section 1.7. In this case, the question is simply: Given a DTM M, an integer c and a string x, does M, if given input x, halt in an accept state after a computation that never uses more than $c \log^k n$ space?

Assuming that both polynomial time and polylogarithmic space are important properties for an algorithm to have, but that neither appears to guarantee the other, a natural class to consider is the class "SC" consisting of all decision problems solvable by algorithms that *simultaneously* obey both types of bounds. To define this class formally, let us first introduce a notation analogous to the "TA[$t(n)$, $a(n)$]" used in Section 3.3 to capture the notion of simultaneous time and alternation bounds:

DEFINITION. The class TS[$t(n)$, $s(n)$] is the set of all problems solvable by DTMs that use at most $t(n)$ time and $s(n)$ space on inputs of size n.

DEFINITION. The class SC = TS[$n^{O(1)}$, $log^{O(1)}n$], i.e., it is the class of all decision problems solvable by DTMs that simultaneously obey polynomial time bounds and polylogarithmic space bounds.

The name "SC" stands for "Steve's Class", in honor of Steven Cook, who proved the first deep result about the class [61]. To motivate Cook's result, let us first observe that although SC$\subseteq$P$\cap$POLYLOG-SPACE, It may not *equal* that intersection. It might well be the case that a problem X can be solved by a polynomial-time algorithm and by a second algorithm that uses only polylog space, but by no algorithm that *simultaneously obeys* both resource bounds. Cook's result, mentioned above, addressed a class of apparent candidates for P$\cap$POLYLOG-SPACE$-$SC, the deterministic context-free languages ("DCFLs", see [32, 122]). These were known to be recognizable by linear-time algorithms and by $\log^2 n$ space algorithms, but did not in general seem to be recognizable by "SC-type" algorithms. Cook effectively dried up this potential source of candidates, by showing that all DCFLs can be recognized in simultaneous $\log^2 n$ space and polynomial (albeit not necessarily linear) time. Thus they are all in SC. To be more precise, they are in the subclass SC^2 of SC, where SC^k is the class of decision problems solvable in simultaneous $\log^k n$ space and polynomial time. (Note that SC^1 = L.) There are other sources of candidates for P$\cap$POLYLOG-SPACE$-$SC however, as we shall see below.

To conclude this section, let us look briefly at the important nondeterministic analogs of the classes we have seen in this section. The major candidate here is the following one.

DEFINITION. The class NL consists of all those decision problems solvable by NDTMs that use space bounded by O($\log|x|$), given input x.

Other natural possibilities for nondeterministic classes have significantly less interest: There seems little need for a nondeterministic version of SC; the class NPOLYLOG-SPACE, defined analogously to NL, simply equals POLYLOG-SPACE, as a consequence of Theorem 3 of Section 2.6; and the class co-NL simply equals NL, as a consequence of Theorem 4 of Section 2.6. (That theorem also implies, incidentally, that there is no point in defining analogs for log-space of the probabilistic classes R and ZPP: these turn out also to equal NL [43]. The analogs of BPP and PP may well be distinct, however. We shall have more to say about them in Section 5.3.)

By definition and [216], we have $L \subseteq NL \subseteq LOG^2$-SPACE $\subseteq$ POLYLOG-SPACE, with the last containment proper by Theorem H2. Somewhat surprisingly, we also have $NL \subseteq P$ [59]. Making this more surprising is the fact that there is an oracle A such that NL^A is not contained in P^A [169]. ($NL \subseteq LOG^2$-SPACE also fails to relativize.) Ladner and Lynch [169] argue that these failures are due to the non-step-by-step nature of the simulations that prove the two nonrelativizing results. Their oracle Turing machine model, however, does not include the cells of the oracle tape among the space to be bounded, and this omission can also be viewed as the source of the difficulty. Unfortunately, it is not clear that there is *any* reasonable way to construct relativizations of classes with sublinear space bounds, and so we will ignore such results for the remainder of this survey. Readers who are nonetheless interested in such results are referred to the above cited paper, and to [44, 45, 178, 218, 253, 254] for more of them.

Given that $NL \subseteq P \cap$ POLYLOG-SPACE, the obvious question to ask is whether $NL \subseteq SC$. We suspect the answer is no. Containment does not follow from the simulation that shows $NL \subseteq P$, as that simulation requires polynomial space. Moreover, there are serious candidates for NL − SC (which are hence also the candidates for $P \cap$ POLYLOG-SPACE − SC promised above). These are the problems that are log-space complete for NL. They are also candidates for NL − L, another presumably nonempty set. They can be in L (SC) if and only if NL = L ($NL \subseteq SC$). For a variety of examples, see [140]. We shall give just one, from [138, 217] that can be viewed as a simpler version of the PATH SYSTEM ACCESSIBILITY problem above.

GRAPH ACCESSIBILITY
Instance: A directed graph G with specified vertices s and t.
Answer: "Yes", if there is a directed path in G from s to t.

There are no similarly natural candidates for SC − NL, but once again an appropriate analog of DTM ACCEPTANCE will suffice. This is one that is log-space complete for $TS[O(n^2), O(\log^2 n)]$: Given a DTM M, constants c and d, and a string x, does M, when given x as input, halt after cn^2 or fewer steps, having used space bounded by $d \log^2 n$?

Before leaving this section, there is one last nondeterministic class worth mentioning, this one inspired by the undirected version of the above GRAPH ACCESSIBILITY problem, also studied (from a randomized point of view) in [8]. Clearly this is a special case of (directed) GRAPH ACCESSIBILITY. Could it be easier? This may well be the case, as it can be shown to be log-space complete for a class that appears to be intermediate between L and NL. To define this class, we need yet one more variant on the NDTM, introduced in [175]. This is the "symmetric Turing machine", an NDTM in which the move relation is symmetric. (If it is possible to move from configuration c to configuration c' in one step, then it is also possible to more from c' to c in one step, for any c and c'.)

Definition. The class SL consists of all those decision problems solvable by log-space bounded symmetric Turing machines.

Another problem complete for SL is the problem, given a graph G, of determining whether G contains an odd cycle. For more on SL and symmetric space-bounded computation, see [175]. It should be pointed out that, although the class is based on one sort of symmetry, it is not known to possess a more standard sort. That is, even though we now know that NL is closed under complement, no one has yet been able to prove that SL and co-SL are identical [43].

5.2. *Parallel computing and the classes NC and RNC*

In the previous section we used the personal computer, a relatively recent computing phenomenon, as a motivation. In this section we turn to another type of computer that is just now coming into its own: the massively parallel computer. Despite our arguments in the previous section as to why space could be more important than time for practical computing, there are situations where polynomial time, even linear time, may be too much, and practitioners would be more than willing to apply more processors to a task if doing so would substantially reduce the running time.

In addressing this issue from their traditional asymptotic point of view, theoretical computer scientists have focused on the PRAM model of computation from Section 1.3, and a class defined in terms of simultaneous time and processor bounds:

Definition. The class TP[$t(n)$, $p(n)$] is the set of all problems solvable by PRAMs that use at most $t(n)$ time and $p(n)$ processors on inputs of size n.

Definition. The class $\mathrm{NC} = \mathrm{TP}[\log^{O(1)} n, n^{O(1)}]$, i.e., it consists of all those decision problems that are solvable on a PRAM that simultaneously obeys a polylogarithmic bound on the running time and a polynomial bound on the number of processors used.

More informally, we might say that NC consists of those problems solvable with a polynomial-bounded amount of hardware in polylog time. As with the sequential complexity classes of previous sections, this class is substantially model-independent. There are equivalent definitions in terms of uniform Boolean circuits of polynomial size and polylog depth, in terms of ATMs obeying simultaneous polylog time and log-space bounds, and in terms of a variety of other models and variants [62, 198, 213]. The name "NC" stands for "Nicks's Class", in honor of Nicholas Pippenger, the first researcher to study the class seriously.

A first observation about NC is that it lies in P, since a single processor can simulate a $\log^j n$ time PRAM computation that uses n^k processors in time $O(n^k \log^j n)$, a polynomial in n. NC also lies inside POLYLOG-SPACE, since by a result of [41] POLYLOG-SPACE is contained in the set of decision problems solvable in polylog time on PRAMs with an *unbounded* number of processors. Only slightly more difficult is the result that $\mathrm{NL} \subseteq \mathrm{NC}$. (There is a straightforward NC algorithm for GRAPH ACCESSIBILITY which is log-space complete for NL.)

It is interesting to compare these results with those for the analogously named class SC. There are superficial reasons why one might thing that NC and SC would be identical. Both are contained in P$\cap$POLYLOG-SPACE. Moreover, each class is defined in terms of two simultaneous resource constraints, and there is a one-to-one correspondence between these constraints if they are considered individually. To see this, let us take the definition of NC in terms of polynomial-size, polylog-depth Boolean circuits that are "log-space uniform", where this uniformity condition is defined as follows.

DEFINITION. A family $\{B_n: n \geqslant 1\}$ of Boolean circuits is *log-space uniform* if there is a DTM that, given n, constructs B_n using space $O(\log n)$.

Note that under this definition, the construction time must be polynomially bounded and hence so must be the size of the circuits. In what follows, we follow [61] in using "uniform" to mean log-space uniform unless we specifically say otherwise.

For the claimed individual correspondences, we use two results from [41]. First, polylog-space DTMs have precisely the same power as polylog-depth uniform Boolean circuits (if one ignores running time for the DTM and circuit size for the Boolean circuits). Second, polynomial-time DTMs have the same power as polynomial-size uniform Boolean circuits (if one ignores the space used by the DTM and the depth of the circuits).

Unfortunately, as we saw when we compared SC and P$\cap$POLYLOG-SPACE, the fact that resource constraints must be obeyed simultaneously can substantially change their effects, and it appears that in fact SC and NC are incomparable. We have already noted one difference: NL is contained in NC but is not known be contained in SC. Thus log-space complete problems for NL, such as graph accessibility, are prime candidates for NC$-$SC. In [61] it was proposed that deterministic context-free languages were prime candidates for membership in SC$-$NC, but in [213] it was subsequently shown that all context-free languages are in NC, even nondeterministic ones. In their place, an alternative candidate for SC$-$NC was proposed. This candidate is a restricted version of the P-complete CIRCUIT VALUE problem mentioned in the previous section. The version of this problem restricted to circuits of polylog "width", as defined in Section 1.3, is a currently viable candidate for SC$-$NC (the version restricted to circuits of polylog depth is likewise a candidate for NC–SC) [213]. (For a alternative discussion of NC and SC, see [130].)

In considering the above examples, we must not lose sight of the fact that a far more important class comparison problem has yet to be resolved. This is the question of whether NC$=$P, a question widely viewed as the natural analog (for parallel computing) of the question of P versus NP. As with P versus NP, there has been little progress toward the resolution of NC versus P, but a flowering of research devoted to classifying the complexities of problems on the assumption that the two classes differ. Here the key lower bound technique is, of course, the completeness result, in this case log-space completeness for P. Since L$\subseteq$NC, a problem that is log-space complete for P cannot be in NC unless NC$=$P. We have already seen this to be the case with "NC" replaced by "POLYLOG-SPACE", "SC", "NL", or "L", but it was only when the connection to parallel computing was noticed and such completeness results could be interpreted as implying the "inherently sequential nature" of a problem, that the question of log-space completeness for P began to attract widespread attention. In this survey we shall make do with the four examples of problems log-space complete for P already exhibited in Sections 1.7 and 5.1, but industrious readers can find many more, for example in [69, 94, 139, 153], and the extensive, although as yet unpublished survey [121].

Balancing the above negative results, we also have had a substantial outpouring of results showing that important problems *are* in NC, and a rapidly developing body of expertise in parallel algorithm design, a survey of which appears elsewhere in this Handbook [153]. Much of this work has been devoted to functions and more general search problems. According to the above definition of NC as a class of decision problems, which corresponds to that given in the original papers that described the class such as [61], such problems are technically ineligible for membership in NC. This has not, however, prevented researchers from ascribing membership to them, either by abusing notation, or simply by redefining NC to be a class of search problems. There has been little consistency in this. For instance, in a sequence of papers authored by Cook [61, 63, 43], NC has switched from a class of decision problems to a class of search problems and back again. Indeed, even this Handbook is not consistent; the choice made here disagrees with the one made in at least one other chapter [153]. For consistency with other class definitions, however, it seems more appropriate to leave NC as a class of decision problems and introduce a new name for the corresponding search problem class. The new class name is chosen by analogy with the distinction we have already made between P and FP:

DEFINITION. The class FNC consists of all those search problems solvable in polylog time by PRAMs with a polynomially bounded number of processors.

Note that NC$\subseteq$FNC by definition. Among the more general search problems in FNC are matrix multiplication (by straightforward techniques), finding minimum spanning trees and Euler tours [11, 54], and a wide variety of algebraic problems, e.g., see [14, 207]. We shall see further examples in the next section.

As was the case for sequential computation, we can often finesse the difference between NC and FNC by showing that there is a decision problem X that has essentially the same complexity as the search problem Y that we are really interested in, i.e., X is in NC if and only if Y is in FNC. This is the case, for instance, when Y is a function whose output size is polynomially bounded. In this case X can simply be: "Given an instance I of Y and an integer k, is the kth bit of Y's answer for I equal to 1?" If X is in NC, one can simply combine a polynomial number of copies of an NC circuit for X (one for each output bit of Y) to obtain an FNC circuit for Y.

When Y is a more general search problem, however, with the possibility of a variety of answers, the correspondence between search and decision problems becomes much less clear than it was in the sequential case. As in the sequential case, the answer to the search problem may be easy to determine given a polynomial number of calls to a subroutine for the decision problem, but if those calls cannot be made in parallel (as they could when Y was a function), we may be unable to satisfy an overall polylog time bound even if the subroutine itself does. For more on this issue, see [155]. In light of the issue, it is often crucial that we perform our complexity analyses on search problems directly, rather than simply on decision problems that would traditionally have been their stand-ins.

Let us conclude this section by mentioning two interesting equivalence classes of problems that so far do not appear to be either in FNC or to be log-space complete for FP. These classes can be viewed as analogs of the class of "GRAPH-ISOMORPHISM equivalent" problems mentioned in Section 2.1, and will contain both decision problems and more general search problems.

For our notion of "equivalence" here, we shall introduce yet another type of reduction, one presumably more powerful than the log-space reduction, but one that is still compatible with NC and FNC (in the sense of Section 1.6). In the definition of this reduction, we assume that the PRAM model is augmented by a special shared memory for the (parallel) construction of oracle queries and for receiving the oracle's answers (which are presumed to arrive one step after the query construction is signalled to be complete). The oracle may process multiple queries in parallel.

DEFINITION. An NC (*Turing*) *reduction* from a search problem X to a search problem Y is an oracle PRAM program that, given an oracle for Y, solves X in polylog time using at most a polynomial number of processors.

These reductions are presumed to be more powerful than log-space reductions both because they allow multiple calls to the oracle, and because NC is presumably more powerful than L. We shall say two problems X and Y are "NC-equivalent" if there are NC reductions from X to Y and from Y to X.

The first class we shall describe is a generalization to search problems of the class of "CC-complete" problems introduced in [185]. Its first member is the "COMPARATOR CIRCUIT VALUE" problem (CCV), that variant of the CIRCUIT VALUE problem in which all circuit elements are "comparators", i.e., two-input gates with two outputs, one yielding $x \vee y$ and one yielding $x \wedge y$, where x and y are the (binary) values of the two inputs. (This problem is "CC-complete" by definition: CC, as defined in [185], is the class of decision problems that are log-space reducible to CCV.) Although no NC-style algorithms are known for CCV, it seems unlikely to be P-complete, given the lack of "fan-out" in the circuit elements. (In P-completeness reductions, as in NP-completeness reductions, one seems always to need a method for transmitting information from one location in the construction to many other locations, and fan-out, in one guise or other, seems to be what is necessary to do the trick.)

The class of CCV-equivalent search problems has surprising variety. A second member is LEXICOGRAPHIC MAXIMAL MATCHING, the problem of finding the lexicographically first maximal matching in a graph G, under some given naming of the graph's edges, where a matching M is *maximal* if all edges in G are either in M or share an endpoint with some edge in M. Note that without the lexicographic restriction, this problem is in FNC, as follows from the result of [156] that finding a maximal independent set is in FNC. (Interestingly enough, the lexicographic version of the latter problem appears to be harder than LEXICOGRAPHIC MAXIMAL MATCHING: finding the lexicographically first maximal independent set *is* known to be log-space complete for FP [63].)

A final example of a CCV-equivalent problem is the STABLE ROOMMATES problem. In this problem, we are given a set S of $2n$ people who are to be assigned to n 2-person rooms, together with a preference list $l(p)$ for each person p (p's rank-ordering of all the other people as potential roommates). We ask whether there is a partition of S into n roommate pairs such that no two non-roommates prefer each other to their current roommates. (That this problem was even in P was only discovered in 1985 [127].)

By altering the STABLE ROOMMATES problem slightly, one obtains the problem that forms the basis for our second (and presumably incomparable) class. Suppose that, instead of preference lists, we specify for each person p a set $a(v)$ of "acceptable" roommates (with p being acceptable to q only if q is acceptable to p). The question of whether there is a partition of S into n pairs of mutually acceptable roommates is the simply the question of whether an undirected graph contains a perfect matching (a problem whose restriction to bipartite graphs was discussed in Section 4.1). Let us call this the PERFECT MATCHING problem.

Among the problems NC-equivalent to PERFECT MATCHING are such problems as computing a maximum *weight* perfect matching (assuming edge weights are written in unary), constructing a maximum cardinality matching, and finding the maximum source-to-sink flow in a directed graph (with unary edge capacities) [154]. (The last of these problems is log-space complete for P if the edge capacities are written in binary [94]. It is not yet known whether maximum weight perfect matching with binary weights shares this property.)

All of these problems are in P or FP (e.g., see [247]), but none are known to be in NC (FNC) or to be log-space complete for P (FP). What is known, however, makes it unlikely that the latter is the case. It has been shown in [154], that PERFECT MATCHING is in the class "RNC", which is the randomized counterpart to NC in the same sense that R is the randomized counterpart to P. RNC could be defined in terms of coin-flipping PRAMs with certain probabilities of obtaining the correct answer, but it will be quicker to define it as follows.

DEFINITION. A decision problem X is in RNC if and only if there is a polynomial p and a problem Y in NC such that for all input strings x the following two properties hold:

(A) If x is a yes-instance of X, then for over half of the possible strings y with $|y| = p(|x|)$, the string xy is a yes-instance of Y.

(B) If x is a no-instance of X, then for all of the possible strings y with $|y| = p(|x|)$, the string xy is a no-instance of Y.

Note that if one replaces NC by P in the above definition, one obtains a definition of the class R, as claimed. As with NC, there is confusion in the literature over whether RNC is allowed to contain arbitrary search problems or not. Once again, we choose to resolve this confusion by giving the more general class a different name.

DEFINITION. A decision problem X is in FRNC if and only if there is a polynomial p and a search problem Y in FNC such that for all input strings x the following two properties hold:

(A) If y is such that $|y| = p(|x|)$, then any answer for xy in problem Y is an answer for x in problem X.

(B) If x has an answer in X, then for over half the possible strings y with $|y| = p(|x|)$, the string xy has an answer in Y.

Under this definition, all the search problems mentioned above as reducible to PERFECT MATCHING are in FRNC [154].

As a final comment on these problems, we note that, due to a result of [148], PERFECT MATCHING is in fact in RNC$\cap$co-RNC, and there is a randomized algorithm for it that uses a polynomially bounded number of processors, that always gives the right answer, and that runs in expected polylog time for every instance. (Just as R$\cap$co-R = ZPP, RNC$\cap$co-RNC equals the analogously defined class ZPNC.) We still do not know whether PERFECT MATCHING is in NC, however. The more general question of whether NC = RNC is also open. For more on NC and RNC, see [153] elsewhere in this Handbook.

5.3. Inside NC

In this and the next two sections, we shall briefly consider the structure of NC and survey the prominent complexity classes inside NC. In doing so, we must make a fundamental shift. Once inside NC, it is no longer possible to talk about complexity classes as "model-independent" in the sense we have used before. Precise details of the model in question often need to be specified if the class is to be well-defined.

For instance, the first classes we shall consider are the classes NC^k, $k \geqslant 1$, of decision problems solvable with polynomial hardware in time $O(\log^k |x|)$. (These classes are analogous to the subclasses SC^k of SC mentioned in the previous section.) For the PRAM definition of NC, the precise nature of the classes NC^k can depend heavily on the assumptions one makes about how algorithms behave when two processors want to access the same memory location at the same time. For instance, one might allow concurrent "reads" but disallow concurrent "writes", as in the "CREW" PRAM (concurrent-read, exclusive-write), allow both operations to take place concurrently, as in the "CRCW" PRAM, or allow neither, as in the "EREW" PRAM. In the case of concurrent writes, there is the additional choice as to which processor actually succeeds in writing to a cell when many attempt to do so concurrently, with the options being "random" (one writer will succeed, but you cannot predict which), "priority" (the processor with lowest index succeeds), or "common" (algorithms are constrained so that no two processors ever attempt to write different things to the same memory cell at the same time). For discussions of the relative power provided by the different choices, see for instance [77, 78, 153].

To avoid these issues, we shall follow most authors and use a definition in terms of uniform Boolean circuits, although even here there are choices to be made. In particular, the class NC^1 can depend significantly on the precise "uniformity" condition imposed on the circuits [213]. Here we make the most common choice and once again use log-space uniformity. We shall also continue our distinction between classes of decision problems and classes of search problems.

Definition. For each $k \geqslant 1$, the class NC^k (FNC^k) consists of all languages recognizable (search problems solvable) by log-space uniform classes of Boolean circuits having polynomial size and depth $O(\log^k n)$.

A first observation about the classes NC^k is that $NC^k \subseteq LOG^k$-SPACE for all $k \geqslant 1$ [41]. A second observation concerns the relation of the NC^k to another sequence of classes that stratifies NC. This second sequence is defined in terms of "unbounded fan-in circuits", a variant on our standard Boolean circuit model in which AND-gates and OR-gates are allowed to have arbitrarily large fan-in. Note that something very much like unbounded fan-in occurs in programmable logic arrays (PLAs) and indeed any time one uses a bus to distribute data. Thus this concept is not simply a theoretical construct. Nevertheless, it does offer surprising powers. For instance the decision problem: "Is the input string made up of all 0s?" can be solved by a depth-1 unbounded fan-in circuit, whereas in an ordinary circuit this problem would require depth at least $\Omega(\log n)$, just so all the input bits could be *communicated* to the output gate. The classes based on this model are defined analogously to the classes NC^k.

Definition. For each $k \geqslant 1$, the class AC^k consists of all languages recognizable by log-space uniform classes of unbounded fan-in circuits having polynomial size and depth $O(\log^k n)$.

It is not difficult to show that for all $k \geqslant 0$, $AC^k \subseteq NC^{k+1} \subseteq AC^{k+1}$, and hence the union of all the classes AC^k is simply NC. We shall not have anything more to say about the AC^k for now, but will have a lot to say about the especially interesting class AC^0 of bounded-depth circuits in Section 5.4.

So far, not many of the classes NC^k have proved individually interesting. Thus there has, however, been considerable research effort (and complexity class generation) inside the lowest classes of this hierarchy (NC^1 and NC^2). We shall cover the contents of NC^1 in the next section and will conclude this section by examining classes of problems that appear to lie *between* NC^1 and NC^2.

Before embarking on this discussion, however, we should admit that there is some controversy among theoreticians about the significance of the NC^k hierarchy. Many argue that the divison of NC into subclasses based on running times obscures the real bottleneck for parallel computing, which is the number of processors required. An algorithm that requires $|x|$ processors and $\log^2|x|$ running time is likely to be far more useful than one that requires $|x|^2$ processors and takes $\log|x|$ time, but the latter is in NC^1, the more restrictive (and hence presumably better) class. In the real world, where processors are limited, it is the *time* $\times$ *processor* product that may be the relevant complexity measure for parallel computation, as it provides the basis for a time–processor trade-off as more processors become available.

For this trade-off to be beneficial with relatively small numbers of processors, we would ideally like the product to equal the best running time known for a sequential algorithm that solves the problem, and indeed, parallel algorithm designers strive to meet this goal, at least to within a low-order polylog factor. Note, however, that useful near-optimal time–processor trade-offs can hold even for problems not in NC. A P-complete problem that can be solved by a parallel algorithm that runs in time $O(n)$ using n processors may well be much easier in practice than a problem in NC that can be solved in time $O(\log n)$ using n^3 processors, given that in the real world even obtaining n processors may be impossible for realistic values of n. Thus, one should be careful about taking at face value all claims (such as the one made in the previous section) that the P-complete problems are "inherently sequential".

Despite the above limitations, there is still much theoretical interest in determining where certain important problems lie in the NC^k hierarchy. Before we go on to consider this issue in detail, let us address two final general questions. First, how far up does the hierarchy extend? Is it infinite, or does it collapse to some level? (As with the polynomial hierarchy, if for some k, $NC^k = NC^{k+1}$, than $NC = NC^k$.) A collapse seems unlikely, but at present we know of no diagonalization argument that precludes it (as Theorem H2 precluded the collapse of the LOG^k-SPACE hierarchy). Indeed, at present we cannot prove even that $NC^1 \neq NP$!

A related question is the following: Can NC have complete problems? We can ask this question both for log-space transformations (which are in FNC^2 for the right choice of machine model) or for NC^1 reductions. The latter are defined in terms of the following machine model.

DEFINITION. An *oracle-augmented Boolean circuit* is a Boolean circuit with an additional class of "oracle" gates allowed, where the latter can have any number of inputs and outputs. The input string for such a gate is the sequence of the values on its input gates; the output string is the sequence of values on its output gates. Given a search problem X as oracle, the output string of an oracle gate with input string x is any y that is an answer for x in X. (If no answer exists, the circuit containing the gate fails.)

DEFINITION. An NC^1 reduction from problem X to problem Y is a log-space uniform family of oracle-augmented Boolean circuits that

(1) solves X given Y as oracle,

(2) contains at most a polynomial number of gates, and

(3) has $O(\log n)$ depth where, for each oracle gate g, the contribution of that gate to the length of the circuit paths containing it is counted as $\log(g_{in} + g_{out})$, where g_{in} and g_{out} are the number of input and output gates of g respectively (e.g., see [63]).

It is not difficult to see that NC^1 reductions are compatible with all the classes NC^k, $k \geq 1$, and that log-space reductions are compatible with all NC^k, $k \geq 2$. Consequently, if NC has complete problems under either log-space or NC^1 reductions, the NC^k hierarchy will collapse. It would thus be a major result if any such problems were to be identified, but they are at present not strictly ruled out.

Let us now turn to the promised discussion of classes of problems that are contained in NC^2 and contain NC^1. We have already seen three such classes; it can be shown that

$NC^1 \subseteq L \subseteq SL \subseteq NL \subseteq NC^2$. Recall that the GRAPH ACCESSIBILITY problems for directed and undirected graphs were complete for NL and SL respectively under log-space transformations. To obtain a complete problem for L (under NC^1 reductions), we need only restrict GRAPH ACCESSIBILITY to directed (or undirected) forests [266]. (For other examples, see [62, 264, 266].) Beyond L, SL, and NL, perhaps the most famous class in the range from NC^1 to NC^2 is the following.

DEFINITION. The class LOGCFL consists of all those decision problems that are log-space reducible to a context-free language.

This class has several alternative characterization. We will mention three, but see also [249]. First, LOGCFL is the set of decision problems solvable by nondeterministic auxiliary pushdown automata in log space and polynomial time (see [238] for definitions and proof). This can be shown to imply that $NL \subseteq LOGCFL$. Second, LOGCFL consists of those decision problems solvable by alternating Turing machines obeying an $O(\log n)$ space bound and a polynomial bound on the total size of the computation tree [212]. Using this characterization, one can conclude that LOGCFL $\subseteq AC^1$ and hence is contained in NC^2 [212]. Third, LOGCFL consists of precisely those decision problems solvable by AC^1 circuits in which no AND-gate has fan-in exceeding 2 (i.e., the fan-in is "semiunbounded" [43]). A final structural result about LOGCFL, proved using techniques similar to those used for proving Theorem 4 (Section 2.6), is that LOGCFL is closed under complement [43]. (NC^k and AC^k, being deterministic classes, are automatically closed under complement for all k.)

Examples of problems that are complete for LOGCFL include the "hardest context-free language" of [98] and the CIRCUIT VALUE problem restricted to monotone circuits having "degree" at most n [63, 265]. (The notion of "degree" used here is defined inductively: the degree of a constant and of an input variable or its negation is 1, the degree of an OR-gate is the maximum of the degrees of its inputs, and the degree of an AND-gate is the sum of the degrees of its inputs [231].) An example of a problem that is in LOGCFL but may not be complete for it is the CIRCUIT VALUE problem for monotone planar circuits [71], where a circuit is monotone if it contains no NOT-gates. (If either one of the two restrictions "planar" or "monotone" is applied by itself, the problem becomes log-space complete for P [92].) Also in LOGCFL are all decision problems log-space reducible to *deterministic* context-free languages, a class we might call "LOGDCFL", and one that is in $NC \cap SC$, by the result of [61] mentioned in the previous section. For more on LOGCFL and the problems in it, see [43, 63, 212, 213, 238].

A second class that has attracted attention, and appears to be incomparable to LOGCFL (as well as AC^1), although it contains L, SL, and NL [43] is the class "DET", first introduced in [63]. This is another class that has been defined both as a class of search problems [63] and as one of decision problems [43]. Since both versions have their usefulness, we shall once again introduce more precise terminology below.

DEFINITION. The class DET (FDET) consists of all those decision problems (search problems) that are log-space reducible to INTEGER DETERMINANT (the problem of computing the determinant of an n by n matrix of n-bit integers).

INTEGER DETERMINANT can be shown to be in FNC^2, and consequently FDET $\subseteq$ FNC^2 (and DET $\subseteq$ NC^2) [42]. The interesting complete problems here are all search problems, and hence technically complete only for FDET. The precise nature of reductions involved in the completeness results, and indeed in the definitions of DET and FDET, are not spelled out in [63], but presumably something like log-space Turing reductions with at most a constant number of calls to the oracle will do. Examples of complete problems from [63] are, in addition to computing the INTEGER DETERMINANT, the problems of computing the inverse of an n by n integer matrix as above (DETERMINANT INVERSE), and of computing the product of n such matrices (ITERATED MATRIX PRODUCT).

The class DET is also of note because it contains two probabilistic complexity classes [42, 43]. The most inclusive of these is the "unbounded two-sided error" class PL, whose relation to L is the same as was that of PP to P in Section 4.5. Both PL and its subclass BPL (the bounded two-sided error analog of BPP) may well be strictly larger than NL (unlike the analogs of R and ZPP, which can be shown to equal NL [43].) As far as we now know, PL and BPL are incomparable to LOGCFL. Further probabilistic classes, based on requiring simultaneous log-space and expected polynomial time can also be defined and placed inside DET (see [43]). For a schema of the classes discussed in this and the previous two sections, see Fig. 7.

5.4. *Inside NC^1*

We conclude Section 5 with a brief look inside NC^1. Before we can begin, however, we have to deal with the fact that there are many NC^{1}s. As mentioned in Section 5.3, even when restricting oneself to the uniform Boolean circuit definition of this class, the precise class defined can depend on the uniformity condition imposed. So far we have restricted attention to the standard log-space uniformity, but other possibilities exist and may be more appropriate.

For instance, one might ask why in practice one should require that the circuits be computable in log-space, rather than allowing full use of the power of polynomial time (in which case we call the circuits "P-uniform"). If in fact one were going to manufacture many copies of each circuit, one might well be able to amortize the polynomial design cost. We did not raise this issue earlier, as we know of no examples higher up in the NC^k hierarchy that suggest that the two types of uniformity differ. There *are* examples, however, when one compares (log-space uniform) NC^1 to P-uniform NC^1, or more precisely, when one compares the corresponding classes of search problem.

For example, consider the ITERATED PRODUCT problem, in which one is given n n-bit integers and asked simply to compute their product. This problem is in P-uniform FNC^1 [26], but the best log-space uniform circuit family known for it has depth $O(\log n \log\log n)$ [207], and hence is just slightly too deep to qualify. For further examples, see [26].

Alternatively, instead of asking for laxer definitions of uniformity, one might argue the conditions should be more stringent than log-space uniformity. As we saw in the previous section, L may properly contain NC^1. Thus in assuming log-space uniformity in the definition of NC^1, we are allowing the machine that constructs the circuits to have more power than the circuits themselves, a bothersome property when one is trying to make fine distinctions about the computational power of such circuits. For this and for other more technical reasons, some researchers feel that the uniformity condition used in defining NC^1 should be no stronger than NC^1-computability itself. Advocates of this position, including the authors of [43, 63], now usually suggest that we use the notion of "U_{E^*}-uniformity" originally proposed in [213].

This notion of uniformity is as technical as its name would suggest, and we shall not describe it in detail here. An essential point, however, is that the machine involved in the definition does not have to construct the circuits; it merely has to recognize a language describing the interconnections of their gates. This makes the machine's task easier, and thus makes it possible for us to get by with reduced computing power. The "reduced power" machine chosen for the definition of U_{E^*}-uniformity is a specially modified alternating Turing machine whose running time is $O(\log n)$, where n is the length of the input. The modification replaces the standard input tape with a random access mechanism: The Turing machine writes the address of an input bit it wishes to obtain on a length $O(\log n)$ indexing tape, and then receives that bit in a special read-only register on the next step. In this way, although no single $O(\log n)$ computation path can look at all the bits of an n-bit input, the entire computation tree *can* take all the input bits into account. (Had we used the standard linear input tape, the computation tree would have been restricted to the first $O(\log n)$ bits of the input.)

It should be pointed out that, in the definition of U_{E^*}-uniformity, these $O(\log n)$ time "random access" ATMs are given a significant boost. The strings in the "extended connection language" E^* that the ATMs must recognize, although of length n where n is the number of inputs to the circuit, contain only $O(\log n)$ relevant bits. (These are padded out to the required length with additional, meaningless bits.) In comparing the power of these ATMs to that of NC^1 machines, however, the relevant question is how well the ATMs can do without such help. More precisely, if the ATMs are to be no more powerful, the following class must be contained in U_{E^*}-uniform NC^1.

DEFINITION. The class ALOGTIME consists of all those decision problems solvable by $O(\log n)$ time bounded ATMs, where n is the length of the input.

Surprisingly, not only is ALOGTIME contained in U_{E^*}-uniform NC^1, the two classes are equal [213]! There is currently, however, no proof of equality between U_{E^*}-uniform NC^1 and log-space uniform NC^1. Thus all we know at present is that

$$\text{ALOGTIME} = U_{E^*}\text{-uniform } NC^1 \subseteq \text{log-space uniform } NC^1.$$

Note that, as suggested above, the distinctions between these uniformity conditions disappears for NC^k, $k > 1$. For such k, log-space uniform NC^k equals U_{E^*}-uniform NC^k, and both equal the appropriate generalization of ALOGTIME, i.e., the class of languages recognized by ATMs with time and space bounded by $O(\log^k n)$ and $O(\log n)$ respectively [213].

Even if ALOGTIME does not equal (log-space uniform) NC^1, we can consider it to be the largest interesting class contained therein. The *smallest* nontrivial class that we shall consider is the analog of ALOGTIME for deterministic Turing machines.

DEFINITION. The class DLOGTIME consists of all those decision problems solvable by a "random access" DTM in $O(\log n)$ time, where n is the length of the input.

Needless to say, the $O(\log n)$ time DTM is a *very* weak model of computation, as its answers must ignore all but $\log n$ bits of the input (although *which* $O(\log n)$ bits it is may depend on the values of the bits seen). DLOGTIME will be contained in every significant class we examine from now on. It contains little in the way of interesting problems itself, unless one considers such tasks as using binary search to verify the length of your input as interesting. There are important uses for $O(\log n)$ time DTMs, however.

For one thing, such machines can be used to define a uniformity condition that is in no danger of providing hidden power to the "uniform" classes of machines it defines (as we worried that log-space uniformity might do to NC^1). As with U_{E^*} uniformity, this new "DLOGTIME-uniformity" condition is defined in terms of recognizing a language that describes the interconnection patterns of the gates in the "uniform" circuits. Again the strings in the language contain only $O(\log n)$ significant bits, but are padded out to length n. (The language itself is slightly different: a "direct" rather than "extended" connection language.) Adding to the theoretical appeal of DLOGTIME-uniformity is the fact that it is equivalent to a seemingly very different notion of uniformity proposed in [124] and based on definability by first-order formulae of mathematical logic [23]. Also appealing is the fact that DLOGTIME-uniform NC^1 remains equal to U_{E^*}-uniform NC^1.

A second use for $O(\log n)$ time DTMs is in reductions. Again the very weakness of DLOGTIME will ensure that the resulting class of transformations is compatible with all the classes in which we will be interested.

DEFINITION. A polynomial transformation f from a problem X to a problem y is a DLOGTIME *transformation* if the language $\{(x, i, c)$: the ith bit of $f(x)$ is $c\}$ is in DLOGTIME.

We shall return to these reductions at the end of this section, where we shall give some examples of problems complete for ALOGTIME under them. First, however, let us examine more of the subclasses of ALOGTIME that are presumably proper subclasses, and hence incapable of containing such ALOGTIME-complete problems.

Perhaps the most studied of these subclasses is AC^0, the class of all decision problems solvable by constant-depth, polynomial-size, unbounded fan-in circuits (as previously defined in Section 5.3), a class that is easily seen to be a strict superset of DLOGTIME. What makes constant-depth, unbounded fan-in circuits so interesting is the lower bound results we can prove for them. First, one can prove that AC^0 contains a noncollapsing hierarchy, based on alternation depth (which in this case is simply depth, since unbounded fan-in means that there is no need for two OR-gates or two

AND-gates in a row). Let the *depth* of an unbounded fan-in circuit be the maximum number of AND- and OR-gates in any path from an input to the output. (Depth 0 would be a circuit that simply hooked some single input gate directly to the output.)

Definition. For all $k \geqslant 0$, the class (uniform) AC_k^0 consists of all problems solvable by DLOGTIME-uniform, depth-k, polynomial-size, unbounded fan-in circuits.

Note that $AC_0^0 \subset \text{DLOGTIME} \subset AC_2^0$, while DLOGTIME and AC_1^0 are incomparable. With classes *this* simple, it is easy to get separations. Higher up in the AC_k^0 hierarchy, things become more interesting (and nontrivial). As shown in [229] however, there are problems in $AC_k^0 - AC_{k-1}^0$ for each $k > 0$. These problems are artificial ones designed to make maximal use of the alternation inherent in AC_k^0. Finding interesting "natural" problems inside AC^0 is more of a challenge. Indeed the class is more interesting for the problems it has been shown *not* to contain, such as PARITY (is the total number of 1s in the input odd?) and MAJORITY (are more than half the input bits 1s?), and a variety of others [40, 75, 84, 232].

Note that in order to prove non-membership in AC^0, one must in fact prove superpolynomial lower bounds on circuit size. Constant-depth, unbounded fan-in circuits are at present the most powerful model in which we have been able to prove such results for problems as "easy" as the ones in NP. The model is so weak, however, that one can prove these bounds for problems that are in NC^1, as are all the above examples. This is not to say that the results do not have any practical implications; they confirm the popular wisdom among VLSI designers that functions like parity and majority cannot be computed by reasonably sized fixed-depth PLAs[229]. Moreover, the results are in a sense "stronger" than necessary for this, as the lower bounds also hold for *nonuniform* circuit families. (Indeed, when the class AC^0 is mentioned in the literature, it is typically viewed in nonuniform terms, and compared to nonuniform versions of the NC^k classes. One also sees mention of the nonuniform classes AC^k, $k > 0$, which are the unbounded fan-in analogs of the NC^k. For our purposes here, we shall mean the DLOGTIME-uniform version of a class unless we state otherwise.)

As an aside, we note that the above lower bounds have recently been strengthened to be truly exponential (rather than merely superpolynomial), and this has important theoretical corollaries, given the formal analogies between AC^0 and the polynomial hierarchy of Section 2.4, elaborated in [40, 84, 229]. (Note that the levels of each can be viewed as offering the opportunity for bounded alternation.) Nonuniform exponential lower bounds on size in the cases of PARITY, proved in [257] and tightened in [114, 115], yield the oracle set for which PSPACE $\neq$ PH. Similarly, exponential lower bounds on size for depth-$(k-1)$ nonuniform circuit families solving problems in AC_k^0, announced in [257] and spelled out in [114, 115], yield an oracle for which PH does not collapse.

Given that bounded-depth, unbounded fan-in circuits made up of AND-, OR- and NOT-gates cannot solve everything in NC^1, a natural question to ask is whether the addition of more powerful gates might help. For example, if you add a "MOD(2)-gate", i.e., one that outputs a 1 if and only if an even number of its input gates are non-zero, then bounded-depth, unbounded fan-in circuits *can* solve the PARITY problem. They cannot, however, solve all problems in NC^1 [203]. More generally, let us consider the following classes of extensions to AC^0, all easily seen to be in NC^1.

DEFINITION. For any positive integer $k > 1$, let $AC^0(k)$ consist of those languages recognized by polynomial-size, bounded-depth, unbounded fan-in Boolean circuits, augmented by "MOD(k)-gates", i.e., unbounded fan-in gates that output "1" if and only if the number of their non-zero inputs is congruent to 0 MOD(k).

For all primes p, $AC^0(p)$ is *strictly* contained in NC^1. In particular, for any primes $p \neq q$, the problem of determining congruence to 0 MOD(q) is not in $AC^0(p)$ [232]. This is as far as this line of research has gone, however: the classes $AC^0(p)$, p prime, are the largest classes known to be properly contained in NC^1. So far, no one has been able to prove even that $AC^0(6)$ does not equal NP! [24].

Should the $AC^0(6)$ challenge fall, there are two more classes of augmented AC^0 circuits waiting in the wings to be the next candidates for proper inclusion in NC^1. They are the following.

DEFINITION. The class ACC consists of all those languages recognized by polynomial-size, bounded-depth, unbounded fan-in Boolean circuits in which any MOD(k)-gate, $k > 1$, may be used.

DEFINITION. The class TC^0 consists of all those languages recognized by polynomial-size, bounded-depth, unbounded fan-in Boolean circuits augmented by "threshold" gates, i.e., unbounded fan-in gates that output "1" if and only if more than half their inputs are non-zero.

For more on these classes, see [22, 24, 100, 208]. It is conjectured that the latter class contains a hierarchy of bounded-depth classes TC_k^0 analogous to the AC_k^0, although so far the highest level separation proved is between classes TC_2^0 and TC_3^0 [100]. The relation between these and our previous classes is summarized as

$$AC^0 \subset ACC \subseteq TC^0 \subseteq NC^1.$$

We conclude our coverage of the classes inside NC^1 with a brief discussion of hopes for an alternative hierarchy, one that would in a sense be perpendicular to the one provided by AC^0. This hierarchy is defined by placing constant bounds on the *width* of circuits (as defined in Section 1.3), rather than on their depth. We also restrict ourselves once more to circuits with bounded fan-in, say, fan-in 2. Define BW_k^0 to be the set of all problems solvable by polynomial-size, bounded fan-in circuits of width k or less, and $BW^0 = \bigcup_{k>1} BW_k^0$. It is not difficult to show that for all $k \geqslant 0$, $AC_k^0 \subseteq BW_k^0$. Moreover, AC^0 is properly contained in BW^0, since PARITY is in BW^0. Are there any candidates for $NC^1 - BW^0$? Here the answer is, surprisingly, no. Unlike the case with AC^0, we have $NC^1 = BW^0$. Moreover, the BW_k^0 hierarchy collapses to its fourth level and we in fact have $NC^1 = BW_4^0$ [22]. As with the above results for AC^0 and its variants, these last results hold in both the uniform and nonuniform case (DLOGTIME-uniformity will suffice [23].) For a somewhat expanded coverage of AC^0 and BW^0, see [134, 136]. For a summary of the inclusion relations between the classes inside NC^1 that we have presented, see Fig. 8.

This last result brings us to the final topic of this section: complete problems for NC^1 (or more precisely, ALOGTIME$=U_{E^*}$-uniform NC^1). Although the NC^k hierarchy would collapse if NC had complete problems, there is no such technical difficulty for NC^1, and indeed problems complete for it under DLOGTIME reductions have already been identified. (One can also get completeness results using the more powerful "AC^0-reduction", based on bounded-depth unbounded fan-in circuits with oracle gates, but so far the extra power seems unnecessary.)

The proof that BW_4^0 equals NC^1 follows directly from the observation that the former contains a problem that is complete for the latter under DLOGTIME-reductions [23, 22]. The problem in question is yet another "product" problem, following in line with ITERATED MATRIX PRODUCT of Section 5.4 and ITERATED PRODUCT of this section. This is PERMUTATION GROUP PRODUCT: "Given a sequence of elements from the permutation group S_5, does their product equal the identity?" A corollary of this completeness result that may be of interest to formal language theorists is that, whereas the hardest context-sensitive languages are complete for PSPACE (see Section 2.6), and the hardest context-free languages are complete for LOGCFL (see the previous section), the hardest regular languages are only complete for ALOGTIME [22]. (It is easy to see that all regular languages are contained in ALOGTIME.)

For our final example of an ALOGTIME-complete problem, we have the BOOLEAN FORMULA VALUE problem, where formulas are strings with syntax and semantics defined inductively as follows: "1" is a formula with value 1, "0" is a formula of value 0, and if f and g are formulas with values $v(f)$ and $v(g)$ respectively, then "$(f \wedge g)$" is a formula whose value is the logical AND of $v(f)$ and $v(g)$ and "$(f \vee g)$" is a formula whose value is the logical OR of $v(f)$ and $v(g)$. The hard part here is showing that the BOOLEAN FORMULA VALUE problem is even *in* ALOGTIME; see [46]. Its presence in the class indicates that NC^1 circuits, unlike the classes of circuits determining DLOGTIME, AC^0, ACC, and TC^0, can perform tasks that are far from rudimentary. Thus, although proving that $NC^1 \neq NP$ would only be a small first step along the way to a proof that $P \neq NP$, it could be the first important one.

6. New developments, tables and figures

We begin in this section with a brief look at an important new result that was obtained since the body of the chapter was written and that has cast some doubts on the comments made in Section 1.8 about the significance of relativized results. We then conclude with a collection of tables and figures that summarize the material in our "Catalog of Complexity Classes" and thus can be used for "ready reference".

6.1. New developments

In Section 4.6 we introduced the concept of "interactive proof system" and the class IP (also known as AM[poly]) of decision problems solvable by interactive proof systems with a polynomial number of rounds. We also observed that IP$\subseteq$PSPACE.

Furthermore, we noted that there is an oracle for which IP does not contain co-NP [80], thus seeming to imply, by the remarks of Section 1.8 on relativization, that co-NP$\subseteq$IP (and hence IP = PSPACE) would be hard to prove.

Surprisingly, both results have now been proved in rapid succession. First came the result that co-NP$\subseteq$IP and in fact PH$\subseteq$IP [267]. Adi Shamir then strengthened this result to show that all of PSPACE was in IP and hence IP = PSPACE [268]. Not only does this give us a precise determination of the power of interactive proofs, it also raises questions about the proper interpretation of relativization results.

In Section 1.8 we indicated that the standard proof techniques for answering questions about complexity classes were "all" known to relativize. This turns out to have been incorrect. It remains true that *most* standard techniques, such as simulation and diagonalization, do relativize, and so the existence of a relativized world in which a conjecture does not hold rules out the use of such techniques in proving it. Shamir's proof, however, is relatively simple and can itself be viewed as an instance of a "standard technique", albeit one that has been remarkable until now only for its failures. This is the approach commonly taken in the many false "proofs" that P = NP: show that a complete (or "hard") problem for the supposedly larger class (for example HAMILTONIAN CIRCUIT in the case of NP) is a member of the supposedly smaller class (i.e., in the case of P, can be solved in polynomial time). Such a proof would not relativize, since individual problems do not relativize. (Oracles can be attached to machines, but there is no natural concept of "relativized HAMILTONIAN CIRCUIT".) What Shamir has done is show that the PSPACE-complete QUANTIFIED BOOLEAN FORMULA problem of Section 2.6 can be solved by a polynomial round interactive proof system. (The problem used to show PH$\subseteq$IP in [267] was the PERMANENT COMPUTATION problem of Section 4.1, which is "hard" for PH based on the results of [240, 243], as discussed in that section.)

It is still not clear why this "specific hard problem" technique should have proved successful with IP while not elsewhere, but one should note that the interactive nature of the computation and the fact that information is hidden from one of the parties is crucial to the details of the proof. Thus it is not clear that this first major success of the specific hard problem approach will signal further successes in disparate domains. (In a similar domain, however, another success of the approach has recently been announced [263]. This time the result is that the class MIP of decision problems solvable by polynomially bounded interactive proof systems in which there are *two* provers is exactly equal to NEXPTIME. See [263] for full definitions and details.)

6.2. *Tables and figures*

This section is devoted to tables and figures that help summarize the material in this "Catalog of Complexity Classes". Tables 2(a) and 2(b) provide an index to the complexity classes we have defined, from AC_0 to ZPP. For each class, we indicate the section in which it is defined, any additional sections in which it is mentioned, and any figures in which it is represented. The list is intended to provide pointers, at least indirectly, for all the classes mentioned in the text. (We omit a few relatives of the included classes, but each of these can be located by following the pointers for the corresponding included class.) Table 3 provides similar indexing information for the

Table 2(a).
Index to the classes

CLASS NAME	DEFINED	MENTIONED/ILLUSTRATED
AC^0	5.3	5.4, Fig. 8
$AC^0(k)$	5.4	
AC^0_k	5.4	Fig. 7
AC^k	5.3	Fig. 7
ACC	5.4	Fig. 8
ALOGTIME	5.4	Fig. 8
AM	4.6	6.1, Fig. 6
AM[poly]	4.6	Fig. 6
BH	2.4	Fig. 1
BH_k	2.4	Fig. 1
BPL	5.3	Fig. 7
BPP	4.5	4.6, 5.1, 5.2, Figs. 5, 6
BW^0	5.4	Fig. 8
BW^0_k	5.4	Fig. 8
CC	5.2	
co-NP	2.2	2.3–2.5, 4.3, 4.5, Figs. 1, 2, 5, 6
co-R	4.3	4.4, 4.5, Fig. 5
D^P	2.4	Fig. 1
DET	5.3	Fig. 7
Δ^P_2	2.3	2.4, 2.5, 4.5, Figs. 1, 2
Δ^P_k	2.5	Fig. 2
DLOGTIME	5.4	Fig. 8
EH	3.3	Fig. 3
ELEMENTARY	3.3	Fig. 3
ETIME	3.1	5.1, Fig. 3
EXPSPACE	3.3	Fig. 3
EXPTIME	3.1	4.3, Fig. 3
$F\Delta^P_2$	2.3	2.4, 5.1
$F\Delta^P_k$	4.1	
FewP	4.2	Fig. 4
FDET	5.3	
FNC	5.2	
FNC^k	5.3	
FP	1.5	1.6, 1.7, 2.4, 4.2, 5.2
FP^{NP}	2.3	2.4
$FP^{NP[k]}$	2.4	
$FP^{NP[O(\log n)]}$	2.4	
FPH	4.1	
FRNC	5.2	
FUP	4.2	
IP	4.6	6.1, Fig. 6
$IP[k]$	4.6	
L	5.1	5.2, 5.3, Fig. 7
LIN-SPACE	2.6	
LOG^k-SPACE	5.1	5.3, Fig. 7
LOGCFL	5.3	Fig. 8
LOGDCFL	5.3	Fig. 8

Table 2(b)
Index to the classes (*continued*)

CLASS NAME	DEFINED	MENTIONED/ILLUSTRATED
MA	4.6	Fig. 6
NC	5.2	5.3, Fig. 7
NC^k	5.3	Fig. 7
NC^1	5.3	5.4, Figs. 7, 8
NETIME	3.2	Fig. 3
NEXPTIME	3.2	6.1, Fig. 3
NLIN-SPACE	2.6	
NL	5.1	5.2, 5.3, Fig. 7
NP	2.1	2.1–2.6, 4.1–4.6, 5.1–5.4, 6.1, Figs. 1, 2 4, 5, 6
$NP \cap co\text{-}NP$	2.2	2.4, 2.5, Fig. 1
#P	4.1	4.2, 4.5, Fig. 4
$\#P_1$	4.1	
OptP	4.1	
$\oplus P$	4.1	4.2, 4.4, Fig. 4
P	1.5	1.7, 2.1–2.6, 4.1–4.5, 5.1, 5.2, 6.1, Figs. 1, 2 4, 5, 6
P^{NETIME}	3.2	
PH	2.5	3.3, 4.1, 4.3–4.6, Figs. 2, 4, 5
Π_2^p	2.5	4.3, 4.5, 4.6, Figs. 2, 5, 6
Π_k^p	2.5	Fig. 3
PL	5.3	Fig. 7
POLYLOG-SPACE	5.1	5.2, Fig. 7
P^{NP}	2.3	2.4, 2.5
$P^{NP[k]}$	2.4	
$P^{NP[O(\log n)]}$	2.4	4.5, Figs. 1, 5
$P^{\#P}$	4.1	4.5, Figs. 4, 5
P^{PP}	4.5	Fig. 5
PP	4.5	5.1, 5.3, Fig. 5
PPSPACE	4.5	4.6, Fig. 5
P/poly	4.3	4.5
PSPACE	2.6	2.4, 3.1, 4.3, 4.5, 4.6, Figs. 2, 3
QH	2.4	Fig. 1
QH_k	2.4	
R	4.3	4.4, 4.5, 5.3, Fig. 5
RNC	5.2	
Σ_2^p	2.5	2.1, 4.3, 4.5, 4.6, Figs. 2, 5, 6
Σ_k^p	2.5	Fig. 3
SC	5.1	5.2, 5.3, Fig. 7
SC^k	5.1	5.3, Fig. 7
SL	5.1	5.3, Fig. 7
span-P	4.1	4.2
$\bigcup_{k>0} TA[2^{n^k}, n]$	3.3	Fig. 3
TC^0	5.4	Fig. 8
TC_k^0	5.4	
UP	4.2	4.3, Fig. 4
UP_k	4.2	Fig. 4
ZPP	4.3	4.4, 4.5, 5.1, 5.3, Fig. 5

Table 3
Index to reducibilities and models of computation

REDUCIBILITY	DEFINED	MENTIONED
AC^0-reduction	5.4	
BPP-reduction	4.4	
DLOGTIME transformation	5.4	
γ-reduction	2.2	4.4
log-space transformation ($\leqslant_{\text{log-space}}$)	1.6	1.7, 5.1–5.3
metric reduction	2.4	4.1
NC-reduction	5.2	
NC^1-reduction	5.3	
nondeterministic polynomial-time Turing reduction	2.2	
parsimonious transformation	4.1	
polynomial transformation ($\leqslant_p$)	1.6	2.1, 2.2, 4.2, 4.3
polynomial-time isomorphism	2.1	
polynomial-time Turing reduction ($\leqslant_T$)	1.6	2.2, 2.3, 4.2, 4.3
R-reduction	4.4	
RUR-reduction	4.4	
strong nondeterministic polynomial-time Turing reduction	2.2	
UR-reduction	4.4	

MODEL OF COMPUTATION	DEFINED	MENTIONED
"advice-taking" Turing machine	4.3	4.5
Arthur–Merlin game	4.6	
alternating Turing machine (ATM)	1.3	2.6, 3.3, 4.6, 5.4
Boolean circuit family	1.3	
nonuniform	1.3	4.3, 5.4
log-space uniform	5.2	5.3, 5.4
P-uniform	5.4	
U_{E^*}-uniform	5.4	
DLOGTIME-uniform	5.4	
counting Turing machine (CTM)	4.1	
deterministic Turing machine (DTM)	1.3	1.5, 1.6, 2.6, 3.1, 3.3, 5.1, 5.2, 5.4
interactive proof	4.6	
nondeterministic Turing machine (NDTM)	1.3	2.1, 3.2, 3.3
oracle-augmented Boolean circuit family	5.3	1.8, 5.4
oracle Turing machine(OTM)	1.3	1.8, 2.3, 2.4, 2.5, 3.2, 4.1, 4.5
parallel random access machine (PRAM)	1.3	5.2, 5.3
parity Turing machine ($\oplus$TM)	4.1	
probabilistic Turing machine (PTM)	4.5	
"random access" Turing machine	5.4	
random Turing machine (RTM)	4.3	
stochastic Turing machine (STM)	4.6	
symmetric Turing machine	5.1	
unambiguous Turing machine (UTM)	4.2	

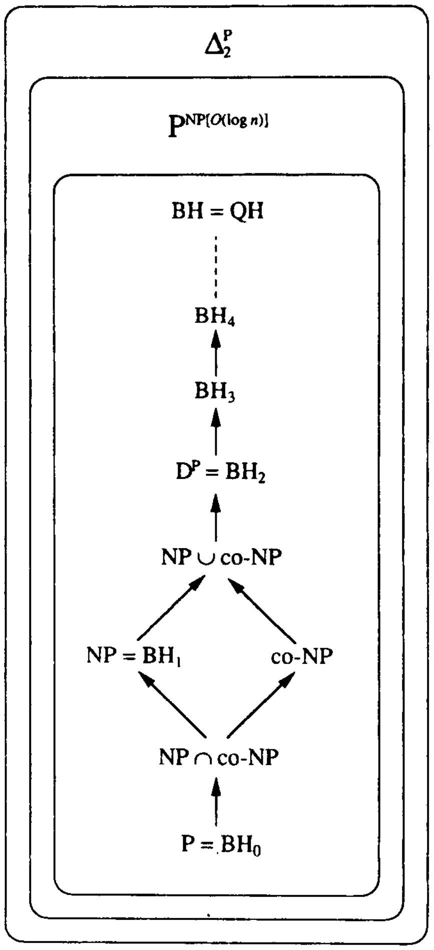

Fig. 1. The Boolean hierarchy.

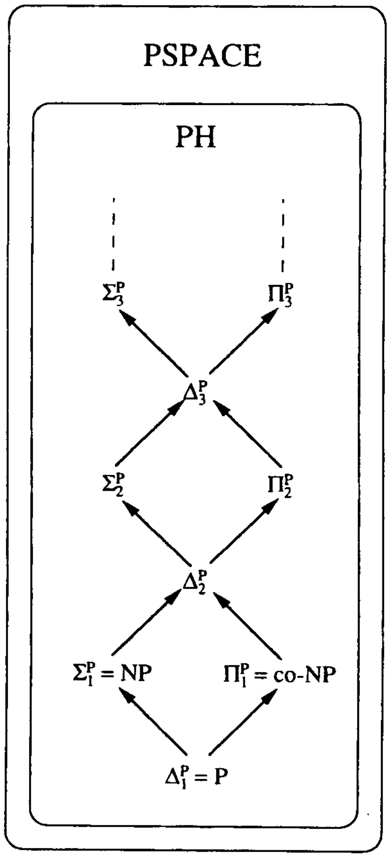

Fig. 2. The polynomial hierarchy.

various notions of "reducibility" and for the various models of computation that we have discussed. For less specialized directories, see the outline at the beginning of the chapter and the overall index to this Handbook.

We also include in this section the eight figures that have already been mentioned in the text. In these figures, an arrow from class A to class B indicates that $A \subseteq B$, and all arrows are drawn upwards. All currently known containment relations are indicated, although some are only present implicitly. (If there is an arrow from A to B and an arrow from B to C, there will be no arrow from A to C, even though the containment relation $A \subseteq C$ is implied.) The figures do not indicate which of the containments are proper. For what is known on this account, consult the appropriate sections of the text. (Because of space constraints, some of the arrows are omitted in Fig. 3. In this case, class A contains class B if the name of class A is written immediatey above that of class B.)

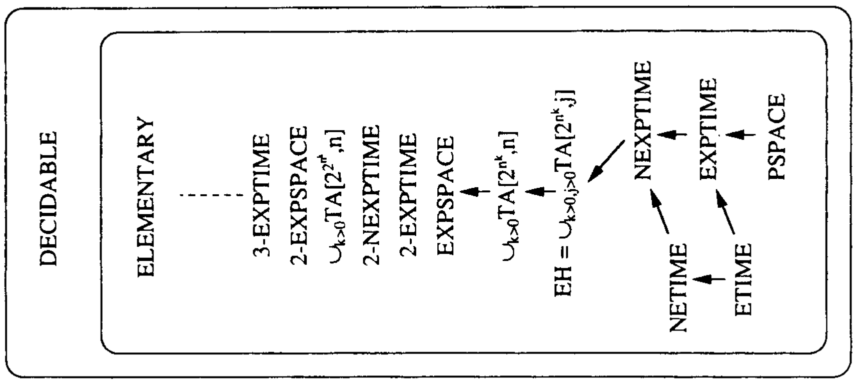

Fig. 3. Provably intractable problems.

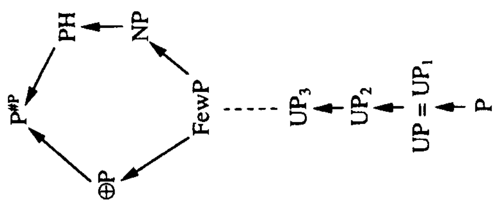

Fig. 4. Counting classes.

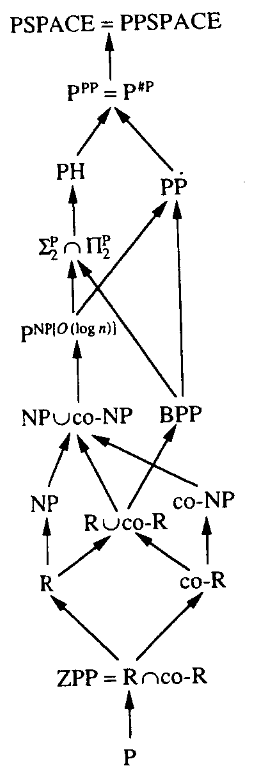

Fig. 5. Randomized complexity classes.

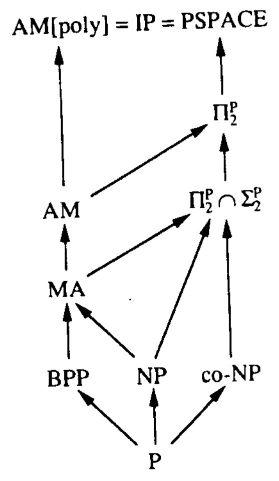

Fig. 6. Interactive complexity classes.

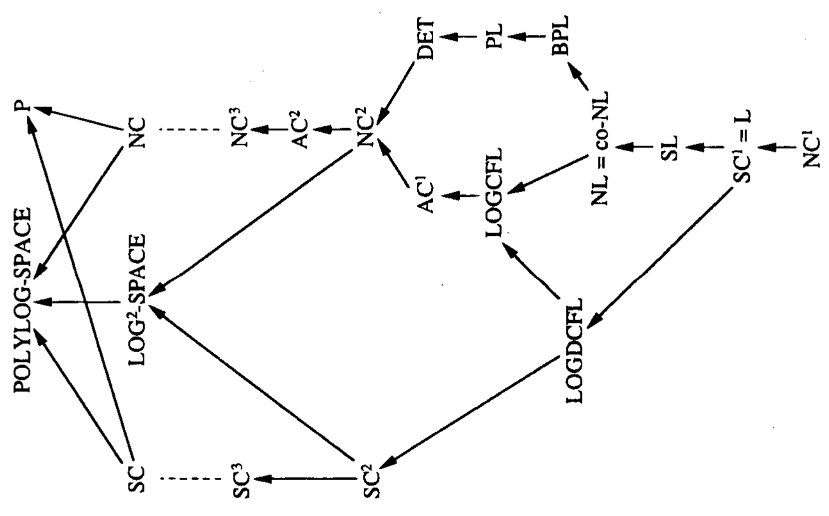

Fig. 7. Classes inside P.

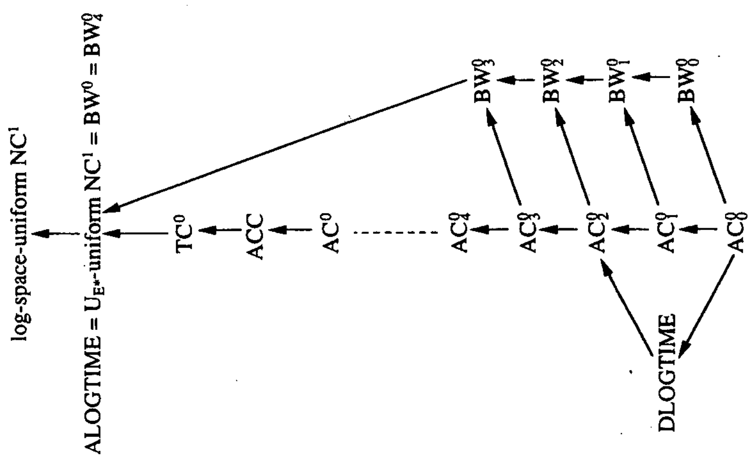

Fig. 8. Classes inside NC¹.

References

[1] ADLEMAN, L., Two theorems on random polynomial time, in: *Proc. 19th Ann. IEEE Symp on Foundations of Computer Science* (1978) 75–83.

[2] ADLEMAN, L. and M. HUANG, Recognizing primes in random polynomial time, in: *Proc. 19th Ann. ACM Symp. on Theory of Computing* (1987) 462–470.

[3] ADLEMAN, L. and K. MANDERS, Reducibility, randomness, and intractability, in: *Proc. 9th Ann. ACM Symp. on Theory of Computing* (1977) 151–163.

[4] ADLEMAN, L.M. and K. MANDERS, Reductions that lie, in: *Proc. 20th Ann. IEEE Symp. on Foundations of Computer Science* (1979) 397–410.

[5] AHO, A.V., J.E. HOPCROFT and J.D. ULLMAN, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[6] AIELLO, W., S. GOLDWASSER and J. HASTAD, On the power of interaction, in: *Proc. 27th Ann. IEEE Symp. on Foundations of Computer Science* (1986) 368–379.

[7] AIELLO, W. and J. HASTAD, Perfect zero-knowledge languages can be recognized in two rounds, in: *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science* (1987) 439–448.

[8] ALELIUNAS, R., R.M. KARP, R.J. LIPTON, L. LOVÁSZ and C. RACKOFF, Random walks, traversal sequences and the complexity of maze problems, in: *Proc. 20th. Ann. IEEE Symp. on Foundations of Computer Science* (1979) 218–223.

[9] ALLENDER, E., The complexity of sparse sets in P, in: A.L. Selman, ed., *Structure in Complexity Theory*, Lecture Notes in Computer Science, Vol. 223 (Springer, Berlin, 1986) 1–11.

[10] ANGLUIN, D., On counting problems and the polynomial-time hierarchy, *Theoret. Comput. Sci.* **12** (1980) 161–173.

[11] AWERBUCH, B., A. ISRAELI and Y. SHILOACH, Finding Euler circuits in logarithmic parallel time, in: *Proc. 16th Ann. ACM Symp. on Theory of Computing* (1984) 249–257.

[12] BABAI, L., Trading group theory for randomness, in: *Proc. 17th Ann. ACM Symp. on Theory of Computing* (1985) 421–429; a subsequent journal paper covering some of this material is: L. Babai and S. Moran, Arthur–Merlin games: A randomized proof system, and a hierarchy of complexity classes, *J. Comput. System Sci.* **36** (1988) 254–276.

[13] BABAI, L., Random oracles separate PSPACE from the polynomial-time hierarchy, *Inform. Process. Lett.* **26** (1987) 51–53.

[14] BABAI, L., E. LUKS and A. SERESS, Permutation groups in NC, in: *Proc. 19th Ann. ACM Symp. on Theory of Computing* (1987) 409–420.

[15] BABAI, L. and E. SZEMERÉDI, On the complexity of matrix group problems, in: *Proc. 25th Ann. IEEE Symp. on Foundations of Computer Science* (1984) 229–240.

[16] BACH, E., G. MILLER and J. SHALLIT, Sums of divisors, perfect numbers and factoring, *SIAM J. Comput.* **15** (1986) 1143–1154.

[17] BAKER, T., J. GILL and R. SOLOVAY, Relativizations of the $P=^{?}NP$ question, *SIAM J. Comput.* **4** (1975) 431–442.

[18] BALCÁZAR, J. L., Logspace self-reducibility, in: *Proc. Structure in Complexity Theory (3rd Ann. IEEE Conf.)* (1988) 40–46.

[19] BALCÁZAR, J.L., Nondeterministic witnesses and nonuniform advice, in: *Proc. Structure in Complexity Theory (4th Ann. IEEE Conf.)* (1989) 259–269.

[20] BALCÁZAR, J.L., J. DIAZ and J. GABARRÓ, *Structural Complexity I* (Springer, Berlin, 1988).

[21] BALCÁZAR, J.L. and D.A. RUSSO, Immunity and simplicity in relativizations of probabilistic complexity classes, *Theoret. Inform. Appl.* **22** (1988) 227–244.

[22] BARRINGTON, D.A., Bounded-width polynomial-size branching programs can recognize exactly those languages in NC^1, *J. Comput. System Sci.* **38** (1989) 150–164.

[23] BARRINGTON, D.A.M., N. IMMERMAN and H. STRAUBING, On uniformity conditions within NC^1, in: *Proc. Structure in Complexity Theory (3rd Ann. IEEE Conf.* (1988) 47–59.

[24] BARRINGTON, D.A.M. and D. THÉRIEN, Finite monoids and the fine structure of NC^1, *J. Assoc. Comput. Mach.* **35** (1988) 941–952.

[25] BARWISE, J., *Handbook of Mathematical Logic* (North-Holland, Amsterdam, 1977).

[26] Beame, P.W., S.A. Cook and H.J. Hoover; Log depth circuits for division and related problems, *SIAM J. Comput.* **15** (1986) 994–1003.

[27] Beigel, R., On the relativized power of additional accepting paths, in: *Proc. Structure in Complexity Theory* (*4th Ann. IEEE Conf.*) (1989) 216–224.

[28] Beigel, R., L.A. Hemachandra and G. Wechsung, On the power of probabilistic polynomial time: $P^{NP[\log]} \subseteq PP$, in: *Proc. Structure in Complexity Theory* (*4th Ann. IEEE Conf.*) (1989) 225–227.

[29] Bennett, C.H. and J. Gill, Relative to a random oracle A, $P^A \neq NP^A \neq$co-NP^A with probability 1, *SIAM J. Comput.* **10** (1981) 96–113.

[30] Berman, L., The complexity of logical theories, *Theoret. Comput. Sci.* **11** (1980) 71–78.

[31] Berman, L. and J. Hartmanis, On isomorphisms and density of NP and other complete sets, *SIAM J. Comput.* **6** (1977) 305–322.

[32] Berstel, J. and L. Boasson, Context-free languages, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. B* (North-Holland, Amsterdam, 1990) Chapter 2.

[33] Blass, A. and Y. Gurevich, On the unique satisfiability problem, *Inform. and Control* **55** (1982) 80–88.

[34] Book, R.V., On languages accepted in polynomial time, *SIAM J. Comput.* **1** (1972) 281–287.

[35] Book, R.V., Comparing complexity classes, *J. Comput. System Sci.* **9** (1974) 213–229.

[36] Book, R.V., Translational lemmas, polynomial time, and $(\log n)^j$-space, *Theoret. Comput. Sci.* **1** (1976) 215–226.

[37] Book, R.V. On the complexity of formal grammars, *Acta Inform.* **9** (1978) 171–182.

[38] Booth, K.S., Isomorphism testing for graphs, semigroups, and finite automata are polynomially equivalent problems, *SIAM J. Comput.* **7** (1978) 273–279.

[39] Boppana, R.B., J. Hastad and S. Zachos, Does co-NP have short interactive proofs?, *Inform. Process. Lett.* **25** (1987) 127–133.

[40] Boppana, R. and M. Sipser, The complexity of finite functions, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. A* (North-Holland, Amsterdam, 1990) 757–804.

[41] Borodin, A., On relating time and space to size and depth, *SIAM J. Comput* **6** (1977) 733–744.

[42] Borodin, A., S.A. Cook and N. Pippenger, Parallel computations for well-endowed rings and space-bounded probablistic machines, *Inform. and Control.* **58** (1983) 113–136.

[43] Borodin, A., S.A. Cook, P.W. Dymond, W.L. Ruzzo and M.L. Tompa, Two applications of inductive counting for complementation problems, *SIAM J. Comput.* **18** (1989) 559–578.

[44] Buss, J.F., A theory of oracle machines, in: *Proc. Structure in Complexity Theory* (*2nd Ann. IEEE Conf.*) (1987) 175–181.

[45] Buss, J.F., Relativized alternation and space bounded computation, *J. Comput. System Sci.* **36** (1988) 351–378.

[46] Buss, S.R., The Boolean formula value problem is in ALOGTIME, in: *Proc. 19th Ann. ACM Symp. on Theory of Computing* (1987) 123–131.

[47] Cai, J.-Y., With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy, in: *Proc. 18th Ann. ACM Symp. on Theory of Computing* (1986) 21–29; journal version in: *J. Comput. System Sci.* **38** (1988) 68–85.

[48] Cai, J.-Y., T. Gundermann, J. Hartmanis, L.A. Hemachandra, V. Sewelson, K. Wagner and G. Wechsung, The Boolean hierarchy I: Structural properties, *SIAM J. Comput.* **17** (1988) 1232–1252.

[49] Cai, J.-Y., T. Gundermann, J. Hartmanis, L.A. Hemachandra, V. Sewelson, K. Wagner and G. Wechsung, The Boolean hierarchy II: Applications, *SIAM J. Comput.* **18** (1989) 95–111.

[50] Cai, J.-Y. and L.A. Hemachandra, The Boolean hierarchy: Hardware over NP, Report No. TR 85-724, Dept. Computer Science, Cornell Univ., Ithaca, NY, 1985.

[51] Cai, J.-Y. and L.A. Hemachandra, On the power of parity polynomial time, in: *Proc. 6th Ann. Symp. on Theoretical Aspects of Computing*, Lecture Notes in Computer Science, Vol. 349 (Springer, Berlin, 1989) 229–239.

[52] Chandra, A.K., D.C. Kozen and L.J. Stockmeyer, Alternation, *J. Assoc. Comput. Mach.* **28** (1981) 114–133.

[53] Chang, R. and J. Kadin, The Boolean hierarchy and the polynomial hierarchy: a closer connection, Report No. TR 89-1008, Dept. Computer Science, Cornell Univ., Ithaca, NY, 1989.

[54] Chin, F.-Y., J. Lam and I.-N. Chen, Efficient parallel algorithms for some graph problems, *Comm. ACM* **25** (1982) 659–665.

[55] CHUNG, M.J. and B. RAVIKUMAR, Strong nondeterministic Turing reduction — a technique for proving intractability, in: *Proc. Structure in Complexity Theory (2nd Ann. IEEE Conf.)* (1987) 132–137.

[56] COBHAM, A., The intrinsic computational difficulty of functions, in: Y. Bar-Hillel, ed., *Proc. 1964 Internat. Congress for Logic Methodology and Philosophy of Science* (North-Holland, Amsterdam, 1964) 24–30.

[57] COOK, S.A., Path systems and language recognition, in: *Proc. 2nd Ann. ACM Symp. on Theory of Computing* (1970) 70–72.

[58] COOK, S.A., The complexity of theorem-proving procedures, in: *Proc. 3rd Ann. ACM Symp. on Theory of Computing* (1971) 151–158.

[59] COOK, S.A., Characterizations of pushdown machines in terms of time-bounded computers, *J. Assoc. Comput. Mach.* **18** (1971) 4–18.

[60] COOK, S.A., An observation on time-storage trade-off, *J. Comput. System Sci.* **9** (1974) 308–316.

[61] COOK, S.A., Deterministic CFL's are accepted simultaneously in polynomial time and log squared space, in: *Proc. 11th Ann. ACM Symp. on Theory of Computing* (1979) 338–345.

[62] COOK, S.A., Towards a complexity theory of synchronous parallel computation, *Enseign. Math.* **27** (1981) 99–124.

[63] COOK, S.A., A taxonomy of problems with fast parallel algorithms, *Inform. and Control* **64** (1985) 2–22.

[64] COOK, S. and R. SETHI, Storage requirements for deterministic polynomial time recognizable languages, *J. Comput. System Sci.* **13** (1976) 25–37.

[65] DAVIS, M., Unsolvable problems, in: J. Barwise, ed., *Handbook of Mathematical Logic* (North-Holland, Amsterdam, 1977) 567–594.

[66] DEKHTYAR, M.I., On the relativation of deterministic and nondeterministic complexity classes, in: *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 45 (Springer, Berlin, 1976) 255–259.

[67] DEMILLO, R.A. and R.J. LIPTON, The consistency of "P = NP" and related problems with fragments of number, theory in: *Proc. 12th Ann. ACM Symp. on Theory of Computing* (1980) 45–57.

[68] DOBKIN, D., R.J. LIPTON and S. REISS, Linear programming is log-space hard for P, *Inform. Process. Lett.* **8** (1979) 96–97.

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

[70] DYER, M.E. and A.M. FRIEZE, On the complexity of computing the volume of a polyhedron, *SIAM J. Comput.* **80** (1989) 205–226.

[71] DYMOND, P.W. and S.A. COOK, Complexity theory of parallel time and hardware, *Inform. and Comput.* **80** (1989) 205–226.

[72] EDMONDS, J., Paths, trees, and flowers, *Canad. J. Math.* **17** (1965) 449–467.

[73] EVEN, S., A.L. SELMAN and Y. YACOBI, The complexity of promise problems with applications to cryptography, *Inform. and Control* **61** (1984) 159–173.

[74] EVEN, S. and R.E. TARJAN, A combinatorial game which is complete in polynomial space, *J. Assoc. Comput. Mach.* **23** (1976) 710–719.

[75] FAGIN, R., M.M. KLAWE, N.J. PIPPENGER and L. STOCKMEYER, Bounded-depth, polynomial size circuits for symmetric functions, *Theoret. Comput. Sci.* **36** (1985) 239–250.

[76] FERRANTE, J. and C. RACKOFF, A decision procedure for the first order theory of real addition with order, *SIAM J. Comput.* **4** (1975) 69–76.

[77] FICH, F.E., P. RAGDE and A. WIGDERSON, Relations between concurrent-write models of parallel computation, *SIAM J. Comput.* **17** (1988) 606–627.

[78] FICH, F.E., P. RAGDE and A. WIGDERSON, Simulations among concurrent-write PRAMs, *Algorithmica* **3** (1988) 43–52.

[79] FORTNOW, L., The complexity of perfect zero-knowledge, in: *Proc. 19th Ann. ACM Symp. on Theory of Computing* (1987) 204–209.

[80] FORTNOW, L. and M. SIPSER, Are there interactive protocols for co-NP languages?, *Inform. Process. Lett.* **28** (1988) 249–251.

[81] FORTUNE, S., D. LEIVANT and M. O'DONNELL, The expressiveness of simple and second-order type structures, *J. Assoc. Comput. Mach.* **30** (1983) 151–185.

[82] FORTUNE, S. and J. WYLLIE, Parallelism in random access machines, in: *Proc. 10th Ann. ACM Symp. on Theory of Computing* (1978) 114–118.

[83] FRAENKEL, A.S. and D. LICHTENSTEIN, Computing a perfect strategy for $n \times n$ chess requires time exponential in n, *J. Combin. Theory Ser. A.* **31** (1981) 199–213.

[84] FURST, M., J. SAXE and M. SIPSER, Parity, circuits, and the polynomial time hierarchy, *Math. Systems Theory.* **17** (1984) 13–27.

[85] GALPERIN, H. and A. WIGDERSON, Succinct representation of graphs, *Inform. and Control* **56** (1983) 183–198.

[86] GAREY, M.R. and D.S. JOHNSON, Strong NP-completeness results: motivation, examples, and implications, *J. Assoc. Comput. Mach.* **25** (1978) 499–508.

[87] GAREY, M.R. and D.S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979).

[88] GESKE, J. and J. GROLLMAN, Relativizations of unambiguous and random polynomial time classes, *SIAM J. Comput.* **15** (1986) 511–519.

[89] GILBERT, J.R., T. LENGAUER and R.E. TARJAN, The pebbling problem is complete for polynomial space, *SIAM J. Comput.* **9** (1980) 513–524.

[90] GILL, J., Computational complexity of probabilistic Turing machines, *SIAM J. Comput.* **6** (1977) 675–695.

[91] GOLDREICH, O., S. MICALI and A. WIGDERSON, Proofs that yield nothing but their validity and a methodology of cryptographic protocol design, in: *Proc. 27th Ann. IEEE Symp. on Foundations of Computer Science* (1986) 174–187.

[92] GOLDSCHLAGER, L.M., The monotone and planar circuit value problems are log space complete for P, *SIGACT News* **9**(2) (1977) 25–29.

[93] GOLDSCHLAGER, L. and I. PARBERRY, On the construction of parallel computers from various bases of Boolean functions, *Theoret. Comput. Sci.* **43** (1986) 43–58.

[94] GOLDSCHLAGER, L., R. SHAW and J. STAPLES, The maximum flow problem is log space complete for P, *Theoret. Comput. Sci.* **21** (1982) 105–111.

[95] GOLDWASSER, S. and J. KILIAN, Almost all primes can be quickly certified, in: *Proc. 18th Ann. ACM Symp. on Theory of Computing* (1986) 316–330.

[96] GOLDWASSER, S., S. MICALI and C. RACKOFF, The knowledge complexity of interactive proof-systems, in: *Proc. 17th Ann. ACM Symp. on Theory of Computing* (1985) 291–304; a journal version under the same title appears in: *SIAM J. Comput.* **18** (1989) 186–208.

[97] GOLDWASSER, S. and M. SIPSER, Private coins versus public coins in interactive proof systems, in: *Proc. 18th Ann. ACM Symp. on Theory of Computing* (1986) 59–68.

[98] GREIBACH, S.A., The hardest context-free language, *SIAM J. Comput.* **2** (1973) 304–310.

[99] GROLLMAN, J. and A.L. SELMAN, Complexity measures for public-key cryptosystems, *SIAM J. Comput.* **17** (1988) 309–335.

[100] HAJNAL, A., W. MAASS, P. PUDLÁK, M. SZEGEDY and G. TURÁN, Threshold circuits of bounded depth, in: *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science* (1987) 99–110.

[101] HARARY, F. and E.M. PALMER, *Graphical Enumeration* (Academic Press, New York, 1973).

[102] HAREL, D., *Algorithmics: The Spirit of Computing* (Addison-Wesley, Reading, MA, 1987).

[103] HARTMANIS, J., Independence results about context-free languages and lower bounds, *Inform. Process. Lett.* **20** (1985) 241–248.

[104] HARTMANIS, J., Solving problems with conflicting relativizations, *Bull. EATCS* **27** (1985) 40–49.

[105] HARTMANIS, J., Structural complexity column: Sparse complete sets for NP and the optimal collapse of the polynomial hierarchy, *Bull. EATCS* **32** (1987) 73–81.

[106] HARTMANIS, J., Structural complexity column: The collapsing hierarchies, *Bull. EATCS* **33** (1987) 26–39.

[107] HARTMANIS, J., Structural complexity column: Some observations about relativization of space bounded computations, *Bull. EATCS* **35** (1988) 82–92.

[108] HARTMANIS, J. and L. HEMACHANDRA, Complexity classes without machines: On complete languages for UP, *Theoret. Comput. Sci.* **58** (1988) 129–142.

[109] HARTMANIS, J. and J.E. HOPCROFT, Independence results in computer science, *SIGACT News* **8**(4) (1976) 13–24.

[110] HARTMANIS, J. and N. IMMERMAN, On complete problems for NP$\cap$co-NP, in: *Proc. Internat. Coll. on Automata, Languages, and Programming*, Lecture Notes in Computer Science, Vol. 194 (Springer, Berlin, 1985) 250–259.

[111] HARTMANIS, J., N. IMMERMAN and V. SEWELSON, Sparse sets in NP$-$P: EXPTIME versus NEXPTIME, *Inform. and Control* **65** (1985) 159–181.

[112] HARTMANIS, J., P.M. LEWIS and R.E. STEARNS, Classification of computations by time and memory requirements, in: *Proc. IFIP Congress 1965* (Spartan, New York, 1965) 31–35.

[113] HARTMANIS, J. and R.E. STEARNS, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (1965) 285–306.

[114] HASTAD, J., Improved lower bounds for small depth circuits, in: *Proc. 18th Ann. ACM Symp. on Theory of Computing* (1986) 6–20.

[115] HASTAD, J., *Computational Limitations for Small-Depth Circuits* (MIT Press, Cambridge, MA, 1987).

[116] HEMACHANDRA, L.A., The strong exponential hierarchy collapses, in: *Proc. 19th Ann. ACM Symp. on Theory of Computing* (1987) 110–122; also: *J. Comput. System Sci.* **39** (1989) 299–322.

[117] HEMACHANDRA, L.A., Structure of complexity classes: separations, collapses, and completeness, in: *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 324 (Springer, Berlin, 1988) 59–73.

[118] HEMACHANDRA, L.A., Private communication, 1989.

[119] HEMACHANDRA, L.A. and S. JAIN, On relativization and the existence of Turing complete sets, Report No. TR-297, Computer Science Dept., Univ. of Rochester, Rochester, NY, 1989.

[120] HOMER, S. and A.L. SELMAN, Oracles for structural properties: the isomorphism problem and public-key cryptography, in: *Proc. Structure in Complexity Theory* (*4th Ann. IEEE Conf.*) (1989) 3–14.

[121] HOOVER, H.J. and W.L. RUZZO, A compendium of problems complete for P, Manuscript, 1985.

[122] HOPCROFT, J.E. and J.D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).

[123] HUNT, J.W., Topics in probabilistic complexity, Ph.D. Dissertation, Dept. of Electrical Engineering. Stanford Univ., Stanford, CA, 1978.

[124] IMMERMAN, N., Expressibility as a complexity measure: results and directions, in: *Proc. Structure in Complexity Theory* (*2nd Ann. IEEE Conf.*) (1987) 194–202.

[125] IMMERMAN, N., Nondeterministic space is closed under complementation, *SIAM J. Comput.* **17** (1988) 935–938.

[126] IMMERMAN, N. and S.R. MAHANEY, Oracles for which NP has polynomial size circuits, in: *Proc. Conf. on Complexity Theory* (1983) 89–93.

[127] IRVING, R.W., An efficient algorithm for the stable room-mates problem, *J. Algorithms* **6** (1985) 577–595.

[128] JAZAYERI, M., W.F. OGDEN and W.C. ROUNDS, The intrinsically exponential complexity of the circularity problem for attribute grammars, *Comm. ACM* **18** (1975) 697–706.

[129] JOHNSON, D.S., The NP-completeness column: an ongoing guide (1st edition), *J. Algorithms* **2** (1981) 393–405.

[130] JOHNSON, D.S., The NP-completeness column: an ongoing guide (7th edition), *J. Algorithms* **4** (1983) 189–203.

[131] JOHNSON, D.S., The NP-completeness column: an ongoing guide (9th edition), *J. Algorithms* **4** (1983) 397–411.

[132] JOHNSON, D.S., The NP-completeness column: an ongoing guide (12th edition), *J. Algorithms* **5** (1984) 433–447.

[133] JOHNSON, D.S., The NP-completeness column: an ongoing guide (15th edition), *J. Algorithms* **6** (1985) 291–305.

[134] JOHNSON, D.S., The NP-completeness column: an ongoing guide (17th edition): Computing with one hand tied behind your back, *J. Algorithms* **7** (1986) 289–305.

[135] JOHNSON, D.S., The NP-completeness column: an ongoing guide (19th edition): The many faces of polynomial time, *J. Algorithms* **8** (1987) 285–303.

[136] JOHNSON, D.S., The NP-completeness column: an ongoing guide (20th edition): Announcements, updates, and greatest hits, *J. Algorithms* **8** (1987) 438–448.

[137] JOHNSON, D.S., The NP-completeness column: an ongoing guide (21st edition): Interactive proof systems for fun and profit, *J. Algorithms* **9** (1988) 426–444.

[138] JONES, N.D., Space-bouned reducibility among combinatorial problems, *J. Comput. System Sci.* **11** (1975) 68–85.

[139] JONES, N.D. and W.T. LAASER, Complete problems for deterministic polynomial time, *Theoret. Comput. Sci.* **3** (1976) 105–117.

[140] JONES, N.D., Y.E. LIEN and W.T. LAASER, New problems complete for nondeterministic log space, *Math. Systems Theory* **10** (1976) 1–17.

[141] JOSEPH, D., Polynomial time computations in models of ET, *J. Comput. System Sci.* **26** (1983) 311–338.

[142] JOSEPH, D. and P. YOUNG, Independence results in computer science?, *J. Comput. System Sci.* **23** (1981) 205–222.

[143] JOSEPH, D. and P. YOUNG, Corrigendum, *J. Comput. System Sci.* **24** (1982) 378.

[144] JOSEPH, D. and P. YOUNG, A survey of some recent results on computational complexity in weak theories of arithmetic, Report No. 83-10-01, Computer Science Dept., Univ. of Wisconsin, Madison, WI, 1983.

[145] JOSEPH, D. and P. YOUNG, Some remarks on witness functions for nonpolynomial and noncomplete sets in NP, *Theoret. Comput. Sci.* **39** (1985) 225–237.

[146] KADIN, J., $P^{NP[\log n]}$ and sparse Turing-complete sets for NP, in: *Proc. Structure in Complexity Theory (2nd Ann. IEEE Conf.)* (1987) 33–40.

[147] KADIN, J., The polynomial hierarchy collapses if the Boolean hierarchy collapses, *SIAM J. Comput.* **17** (1988) 1263–1282 (errors in the proof in this paper are corrected in [53]).

[148] KARLOFF, H., A Las Vegas RNC algorithm for maximum matching, *Combinatorica* **6** (1986) 387–392.

[149] KARMARKAR, N., A new polynomial-time algorithm for linear programming, *Combinatorica* **4** (1984) 373–395.

[150] KARP, R.M., Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations* (Plenum Press, New York, 1972) 85–103.

[151] KARP, R.M., On the complexity of combinatorial problems, *Networks* **5** (1975) 45–68.

[152] KARP, R.M. and R.J. LIPTON, Some connections between nonuniform and uniform complexity classes, in: *Proc. 12th Ann. ACM Symp. on Theory of Computing* (1980) 302–309; appeared in journal form as: R.M. KARP and R.J. LIPTON, Turing machines that take advice, *Enseign. Math.* **28** (1982) 191–209.

[153] KARP, R.M. and V. RAMACHANDRAN, Parallel algorithms for shared-memory machines, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. A* (North-Holland, Amsterdam, 1990) 869–941.

[154] KARP, R.M., E. UPFAL and A. WIGDERSON, Constructing a maximum matching is in random NC, *Combinatorica* **6** (1986) 35–48.

[155] KARP, R.M., E. UPFAL and A. WIGDERSON, The complexity of parallel search, *J. Comput. System Sci.* **36** (1988) 225–253.

[156] KARP, R.M. and A. WIGDERSON, A fast parallel algorithm for the maximal independent set problem, *J. Assoc. Comput. Mach.* **32** (1986) 762–773.

[157] KHACHIYAN, L.G., A polynomial algorithm in linear programming, *Dokl. Akad. Nauk. SSSR* **244** (1979) 1093–1096 (in Russian); English translation in: *Soviet Math. Dokl.* **20** (1979) 191–194.

[158] KO, K.-I., Some observations on the probabilistic algorithms and NP-hard problems, *Inform. Process. Lett.* **14** (1982) 39–43.

[159] KO, K.-I., Relativized polynomial time hierarchies having exactly *K* levels, *SIAM J. Comput.* **18** (1989) 392–408.

[160] KÖBLER, J., U. SCHÖNING, S. TODA and J. TORÁN, Turing machines with few accepting computations and low sets for PP, in: *Proc. Structure in Complexity Theory (4th Ann. IEEE Conf.)* (1989) 208–215.

[161] KÖBLER, J., U. SCHÖNING and J. TORÁN, On counting and approximation, *Acta Inform.* **26** (1989) 363–379.

[162] KOZEN, D., A clique problem equivalent to graph isomorphism, Manuscript, 1978.

[163] KRENTEL, M., The complexity of optimization problems, *J. Comput. System Sci.* **36** (1988) 490–509.

[164] KRENTEL, M., Generalizations of OptP to the polynomial hierarchy, Report No. TR88-79, Dept. of Computer Science, Rice Univ., Houston, TX, 1988.

[165] KURTZ, S.A., On the random oracle hypothesis, *Inform. and Control* **57** (1983) 40–47.

[166] KURTZ, S.A., S.R. MAHANEY and J.S. ROYER, The isomorphism conjecture fails relative to a random oracle, in: *Proc. 21st Ann. ACM Symp. on Theory of Computing* (1989) 157–166.

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

[168] LADNER, R.E., The circuit value problem is log space complete for P, *SIGACT News* **7**(1)(1975) 18–20.

[169] LADNER, R.E. and N.A. LYNCH, Relativizations about questions of log space computability, *J. Comput. System Sci.* **10** (1976) 19–32.

[170] LADNER, R.E., N.A. LYNCH and A.L. SELMAN, A comparison of polynomial time reducibilities, *Theoret. Comput. Sci.* **1** (1975) 103–124.

[171] LAUTEMANN, C., BPP and the polynomial hierarchy, *Inform. Process. Lett.* **17** (1983) 215–218.

[172] LEGGETT JR, E.W. and D.J. MOORE, Optimization problems and the polynomial hierarchy, *Theoret. Comput. Sci.* 15 (1981) 279–289.

[173] LEIVANT, D., Unprovability of theorems of complexity theory in weak number theories, *Theoret. Comput. Sci.* **18** (1982) 259–268.

[174] LEVIN, L.A., Universal sorting problems, *Problemy Peredaci Informacii* **9** (1973) 115–116 (in Russian); English translation in: *Problems of Information Transmission* **9** (1973) 265–266.

[175] LEWIS, H.R. and C.H. PAPADIMITRIOU, Symmetric space-bounded computation, *Theoret. Comput. Sci.* **19** (1982) 161–187.

[176] LICHTENSTEIN, D. and M. SIPSER, GO is polynomial-space hard, *J. Assoc. Comput. Mach.* **27** (1980) 393–401.

[177] LONG, T.J., Strong nondeterministic polynomial-time reducibilities, *Theoret. Comput. Sci.* **21** (1982) 1–25.

[178] LYNCH, N., Log space machines with multiple oracle tapes, *Theoret. Comput. Sci.* **6** (1978) 25–39.

[179] MAHANEY, S.R., Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis, *J. Comput. System Sci.* **25** (1982) 130–143.

[180] MAHANEY, S.R., Sparse sets and reducibilities, in: R.V. Book, ed., *Studies in Complexity Theory* (Wiley, New York, 1986) 63–118.

[181] MAHANEY, S.R. and P. YOUNG, Reductions among polynomial isomorphism types, *Theoret. Comput. Sci.* **39** (1985) 207–224.

[182] MANDERS, K. and L. ADLEMAN, NP-complete decision problems for binary quadratics, *J. Comput. System Sci.* **16** (1978) 168–184.

[183] MATHON, R., A note on the graph isomorphism counting problem, *Inform. Process. Lett.* **8** (1979) 131–132.

[184] MATIJACEVIC, Y., Enumerable sets are Diophantine, *Dokl. Akad. Nauk. SSSR* **191** (1970) 279–282 (in Russian); English translation in: *Soviet Math. Doklady* **11** (1970) 354–357.

[185] MAYR, E.W. and A.S. SUBRAMANIAN, The complexity of circuit value and network stability, in: *Proc. Structure in Complexity Theory (4th Ann. IEEE Conf.)* (1989) 114–123.

[186] MEYER, A.R., Weak monadic second order theory of successor is not elementary recursive, Manuscript, 1973.

[187] MEYER, A.R. and L.J. STOCKMEYER, The equivalence problem for regular expressions with squaring requires exponential time, in: *Proc. 13th Ann. IEEE Symp. on Switching and Automata Theory* (1972) 125–129.

[188] MILLER, G.L., Riemann's hypothesis and tests for primality, *J. Comput. System Sci.* **13** (1976) 300–317.

[189] NISAN, N. and A. WIGDERSON, Hardness vs. randomness, in: *Proc. 29th Ann. IEEE Symp. on Foundations of Computer Science* (1988) 2–11.

[190] PAPADIMITRIOU, C.H., Games against nature, in: *Proc. 24th Ann. IEEE Symp. on Foundations of Computer Science* (1983) 446–450; revised version appeared as: C.H. PAPADIMITRIOU, Games against nature, *J. Comput. System Sci.* **31** (1985) 288–301.

[191] PAPADIMITRIOU, C.H., On the complexity of unique solutions, *J. Assoc. Comput. Mach.* **31** (1984) 392–400.

[192] PAPADIMITRIOU, C.H. and K. STEIGLITZ, *Combinatorial Optimization: Algorithms and Complexity* (Prentice-Hall, Englewood Cliffs, NJ, 1982).

[193] PAPADIMITRIOU, C.H. and D. WOLFE, The complexity of facets resolved, *J. Comput. System Sci.* **37** (1988) 2–13.

[194] PAPADIMITRIOU, C.H. and M. YANNAKAKIS, The complexity of facets (and some facets of complexity), *J. Comput. System Sci.* **28** (1984) 244–259.

[195] PAPADIMITRIOU, C.H. and M. YANNAKAKIS, A note on succinct representations of graphs, *Inform. and Control* **71** (1986) 181–185.

[196] PAPADIMITRIOU, C.H. and S. ZACHOS, Two remarks on the power of counting, in: *Proc. 6th GI Conf. on Theoretical Computer Science*, Lecture Notes in Computer Science, Vol. 145 (Springer, Berlin, 1983) 269–276.

[197] PAUL, W.J., N. PIPPENGER, E. SZEMERÉDI and W.T. TROTTER, On determinism versus non-determinism and related problems, in: *Proc. 24th Ann. IEEE Symp. on Foundations of Computer Science* (1983) 429–438.

[198] PIPPENGER, N., On simultaneous resource bounds, in: *Proc. 20th Ann. IEEE Symp. on Foundations of Computer Science* (1979) 307–311.

[199] PRATT, V., Every prime has a succinct certificate, *SIAM J. Comput.* **4** (1975) 214–220.

[200] PROVAN, J.S., The complexity of reliability computations in planar and acyclic graphs, *SIAM J. Comput.* **15** (1986) 694–702.

[201] RABIN, M.O., Probabilistic algorithm for testing primality, *J. Number Theory* **12** (1980) 128–138.

[202] RACKOFF, C., Relativized questions involving probabilistic algorithms, *J. Assoc. Comput. Mach.* **29** (1982) 261–268.

[203] RAZBOROV, A.A., Lower bounds for the size of bounded-depth networks over a complete basis with logical addition, *Mat. Zametki* **41** (1987) 598–607 (in Russian); English translation in: *Math. Notes Acad. Sci. USSR* **41** (1987) 333–338.

[204] REGAN, K., The topology of provability in complexity theory, *J. Comput. System Sci.* **36** (1988) 384–432.

[205] REIF, J.H., Universal games of incomplete information, in: *Proc. 11th Ann. ACM Symp. on Theory of Computing* (1979) 288–308.

[206] REIF, J.H., Complexity of the mover's problem and generalizations, in: *Proc. 20th Ann. IEEE Symp. on Foundations of Computer Science* (1979) 421–427; a formal version of this paper appeared as: Complexity of the generalized movers problem, in: J. Schwartz, ed., *Planning Geometry and Complexity of Robot Motion* (Ablex, Norwood, NJ, 1985) 421–453.

[207] REIF, J.H., Logarithmic depth circuits for algebraic functions, *SIAM J. Comput.* **15** (1986) 231–282.

[208] REIF, J.H., On threshold circuits and polynomial computation, in: *Proc. Structure in Complexity Theory* (*2nd Ann. IEEE Conf.*) (1987) 118–123.

[209] RIVEST, R., Cryptography, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. A* (North-Holland, Amsterdam, 1990) 717–755.

[210] ROBSON, J.M., *N* by *N* checkers is Exptime complete, *SIAM J. Comput.* **13** (1984) 252–267.

[211] RUDICH, S., Limits on provable consequences of one-way functions, Doctoral Dissertation, Univ. of California, Berkeley, CA, 1988.

[212] RUZZO, W.L., Tree-size bounded alternation, *J. Comput. System Sci.* **21** (1980) 218–235.

[213] RUZZO, W.L., On uniform circuit complexity, *J. Comput. System Sci.* **22** (1981) 365–383.

[214] SALOMAA, A., Formal languages and power series, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. B* (North-Holland, Amsterdam, 1990).

[215] SANTHA, M., Relativized Arthur–Merlin versus Merlin–Arthur games, *Inform. and Comput.* **80** (1989) 44–49.

[216] SAVITCH, W.J., Relationship between nondeterministic and deterministic tape classes, *J. Comput. System Sci.* **4** (1970) 177–192.

[217] SAVITCH, W.J., Nondeterministic log *n* space, in: *Proc. 8th Ann. Princeton Conf. on Information Sciences and Systems*, Dept. of Electrical Engineering, Princeton Univ., Princeton, NJ (1974) 21–23.

[218] SAVITCH, W.J., A note on relativized log space, *Math. Systems Theory* **16** (1983) 229–235.

[219] SCHAEFER, T.J., Complexity of some two-person perfect-information games, *J. Comput. System Sci.* **16** (1978) 185–225.

[220] SCHÖNING, U., Graph isomorphism is in the low hierarchy, in: *Proc. 4th Symp. on Theoretical Aspects of Computing*, Lecture Notes in Computer Science, Vol. 247 (Springer, Berlin, 1986) 114–124.

[221] SCHWARTZ, J.T., Fast probabilistic algorithms for verification of polynomial identities, *J. Assoc. Comput. Mach.* **27** (1980) 710–717.

[222] SEIFERAS, J.I., Relating refined space complexity classes, *J. Comput. System Sci.* **14** (1977) 100–129.

[223] SEIFERAS, J.I., M.J. FISCHER and A.R. MEYER, Separating nondeterministic time complexity classes, *J. Assoc. Comput. Mach.* **25** (1978) 146–167.

[224] SHMOYS, D.B. and E. TARDOS, Computational complexity, in: *Handbook of Combinatorics* (North-Holland, Amsterdam, to appear).

[225] SIMON, J., On some central problems in computational complexity, Doctoral Thesis, Dept. of Computer Science, Cornell Univ., Ithaca, NY, 1975.

[226] SIMON, J., Space-bounded probabilistic Turing machine complexity classes are closed under complement, in: *Proc. 13th Ann. ACM Symp. on Theory of Computing* (1981) 158–167.

[227] SIMON, J., On tape-bounded probabilistic Turing machine acceptors, *Theoret. Comput. Sci.* **16** (1981) 75–91.

[228] SIPSER, M., On relativization and the existence of complete sets, in: *Proc. Internat. Coll. on Automata, Languages, and Programming*, Lecture Notes in Computer Science, Vol. 140 (Springer, Berlin, 1982) 523–531.

[229] SIPSER, M., Borel sets and circuit complexity, in: *Proc. 15th Ann. ACM Symp. on Theory of Computing* (1983) 61–69.

[230] SIPSER, M., A complexity theoretic approach to randomness, in: *Proc. 15th Ann. ACM Symp. on Theory of Computing* (1983) 330–335.

[231] SKYUM, S., and L.G. VALIANT, A complexity theory based on Boolean algebra, *J. Assoc. Comput. Mach.* **32** (1985) 484–502.

[232] SMOLENSKI, R., Algebraic methods in the theory of lower bounds for Boolean circuit complexity, in: *Proc. 19th Ann. ACM Symp. on Theory of Computing* (1987) 77–82.

[233] SOLOVAY, R. and V. STRASSEN, A fast Monte-Carlo test for primality, *SIAM J. Comput.* **6** (1977) 84–85.

[234] STOCKMEYER, L., The polynomial time hierarchy, *Theoret. Comput. Sci.* **3** (1976) 1–22.

[235] STOCKMEYER, L., On approximation algorithms for #P, *SIAM J. Comput.* **14** (1985) 849–861.

[236] STOCKMEYER, L., Classifying the computational complexity of problems, *J. Symbolic Logic* **52** (1987) 1–43.

[237] STOCKMEYER, L.J. and A.R. MEYER, Word problems requiring exponential time, in: *Proc. 5th Ann. ACM Symp. on Theory of Computing* (1973) 1–9.

[238] SUDBOROUGH, I.H., On the tape complexity of deterministic context-free languages, *J. Assoc. Comput. Mach.* **25** (1978) 405–414.

[239] SZELEPCSÉNYI, R., The method of forcing for nondeterministic automata, *Bull. EATCS* **33** (1987) 96–100.

[240] TODA, S., On the computational power of PP and ⊕P, in: *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science* (1989) 514–519.

[241] TORÁN, J., Structural properties of the counting hierarchies, Doctoral Dissertation, Facultat d'Informatica, UPC Barcelona, 1988.

[242] VALIANT, L.G., Relative complexity of checking and evaluating, *Inform. Process. Lett.* **5** (1976) 20–23.

[243] VALIANT, L.G., The complexity of computing the permanent, *Theoret. Comput. Sci.* **8** (1979) 189–201.

[244] VALIANT, L.G., The complexity of enumeration and reliability problems, *SIAM J. Comput.* **8** (1979) 410–421.

[245] VALIANT, L.G. and V.V. VAZIRANI, NP is as easy as detecting unique solutions, *Theoret. Comput. Sci.* **47** (1986) 85–93.

[246] VAN EMDE BOAS, P., Machine models and simulations, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. A* (North-Holland, Amsterdam, 1990) 1–66.

[247] VAN LEEUWEN, J., Graph algorithms, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. A* (North-Holland, Amsterdam, 1990) 525–631.

[248] VAZIRANI, U.V. and V.V. VAZIRANI, A natural encoding scheme proved probabilistic polynomial complete, *Theoret. Comput. Sci.* **24** (1983) 291–300.

[249] VENKATESWARAN, H. and M. TOMPA, A new pebble game that characterizes parallel complexity classes, *SIAM J. Comput.* **18** (1989) 533–549.

[250] WAGNER, K.W., Bounded query computation, in: *Proc. Structure in Complexity Theory (3rd Ann. IEEE Conf.)* (1988) 260–277.

[251] WANG, H., Proving theorems by pattern recognition, *Bell Systems Tech. J.* **40** (1961) 1–42.

[252] WECHSUNG, G., On the Boolean closure of NP, in: *Proc. Internat. Conf. on Fundamentals of Computation Theory*, Lecture Notes in Computer Science, Vol. 199 (Springer, Berlin, 1985) 485–493.

[253] WILSON, C.B., Relativized circuit complexity, in: *Proc. 24th Ann. IEEE Symp. on Foundations of Computer Science* (1983) 329–334.

[254] WILSON, C.B., A measure of relativized space which is faithful with respect to depth, *J. Comput. System Sci.* **36** (1988) 303–312.

[255] WRATHALL, C., Complete sets for the polynomial time hierarchy, *Theoret. Comput. Sci.* **3** (1976) 23–34.

[256] YAO, A.C., Theory and applications of trapdoor functions, in: *Proc. 23rd Ann. IEEE Symp. on Foundations of Computer Science* (1982) 80–91.

[257] YAO, A.C.-C., Separating the polynomial-time hierarchy by oracles, in: *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science* (1985) 1–10.

[258] YOUNG, P., Some structural properties of polynomial reducibilities and sets in NP, in: *Proc. 15th Ann. ACM Symp. on Theory of Computing* (1983) 392–401.

[259] ZACHOS, S., Collapsing probabilistic polynomial hierarchies, in: *Proc. Conf. on Complexity Theory* (1983) 75–81.

[260] ZACHOS, S., Probabilistic quantifiers and games, *J. Comput. System Sci.* **36** (1988) 433–451.

[261] ZACHOS, S. and H. HELLER, A decisive characterization of BPP, *Inform. and Control* **69** (1986) 125–135.

[262] BABAI, L., Talk presented at the 21st Annual Symposium on Foundation of Computer Science, San Juan, Puerto Rico, October 29, 1979 (not in the Proceedings).

[263] BABAI, L., L. FORTNOW and C. LUND, Non-deterministic exponential time has two-prover interactive protocols, Manuscript, 1990.

[264] CHEN, J., A new complete problem for DSPACE(log n), *Discrete Applied Math.* **25** (1989) 19–26.

[265] COOK, S.A., Personal communication correcting typographical error in [63].

[266] COOK, S.A. and P. MCKENZIE, Problems complete for deterministic logarithmic space, *J. Algorithms* **8** (1987) 385–394.

[267] LUND, C., L. FORTNOW, H. KARLOFF and N. NISAN, The polynomial time hierarchy has interactive proofs, Announcement by electronic mail, December 13, 1989.

[268] SHAMIR, A., IP = PSPACE, Manuscript, 1989.