Texts in Computational Complexity: P/poly and PH

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel.

November 28, 2005

Summary: We consider variations on the complexity classes P and NP. We refer specifically to the non-uniform version of P, and to the Polynomial-time Hierarchy (which extends NP). These variations are motivated by relatively technical considerations, and the resulting classes are referred to quite frequently in the literature.

Non-uniform polynomial-time (P/poly) captures efficient computations that are carried out by devices that handle specific input lengths. The basic formalism ignore the complexity of constructing such devices (i.e., a uniformity condition). A finer formalism that allows to quantify the amount of non-uniformity refers to so called "machines that take advice."

The Polynomial-time Hierarchy (PH) generalizes NP by considering statements expressed by a quantified Boolean formula with a fixed number of alternations of existential and universal quantifiers. It is widely believed that each quantifier alternation adds expressive power to the class of such formulae.

The two different classes are related by showing that if NP is contained in P/poly then the Polynomial-time Hierarchy collapses to its second level. Assuming the latter collapse does not occur, this means that there is hope to resolve the P-vs-NP Question by showing that NP is not contained in "non-uniform P" (P/poly).

1 Non-uniform polynomial-time (P/poly)

In this section we consider two formulations of the notion of non-uniform polynomial-time, presenting two models of non-uniform (polynomial-time) computing devices. These models are derived by specializing the treatment of non-uniform computing devices, to the case of polynomially bounded complexities. It turns out that both models allow to solve the same class of computational problems, which is a strict superset of the class of problems solvable by polynomial-time algorithms.

The two models discussed below are Boolean circuits and "machines that take advice". We will focus on the restriction of both models to the case of polynomial complexities, considering (non-uniform) polynomial-size circuits and polynomial-time algorithms that take (non-uniform) polynomially bounded advice.

The main motivation for considering non-uniform polynomial-size circuits is that their limitations imply analogous limitations on polynomial-time algorithms. The hope is that, as is often the case in mathematics and Science, disposing of an auxiliary condition that is not well-understood (i.e., uniformity) may turn out fruitful. In particular, it may facilitate a low-level analysis of the evolution of a computation, by using combinatorial techniques. This hope has materialized in the study of restricted classes of circuits (e.g., monotone circuits and constant-depth circuits).

Polynomial-time algorithms that take polynomially bounded advice are useful in modeling auxiliary information available to possible efficient strategies that are of interest to us. Indeed, the typical cases are the modeling of adversaries in the context of cryptography and the modeling of arbitrary randomized algorithms in the context of derandomization. Furthermore, the model of polynomial-time algorithms that take advice allows for a quantitative study of the amount of non-uniformity, ranging from zero to polynomial.

1.1 Boolean Circuits

We assume familiarity with the definition of (families of) Boolean circuits and the functions computed by them. For concreteness and simplicity, we assume throughout this section that all circuits has bounded fan-in. We highlight the following simple result.

Theorem 1 (circuit evaluation): *There exists a polynomial-time algorithm that, given a circuit $C:\{0,1\}^n \to \{0,1\}^m$ and an n-bit long string x, returns $C(x)$.*

Recall that the algorithm works by performing the value-determining process described when defining the value of the circuit vertices on a given input.

Circuit size as a complexity measure. We recall the standard definitions of circuit complexity: The size of a circuit is defined as the number of edges, and the length of its description is almost linear in the latter; that is, a circuit of size s is commonly described by the list of its edges and the labels of its vertices, which means that its description length is $O(s \log s)$. We are interested in families of circuits that solve computational problems, and say that the circuit family $(C_n)_{n\in\mathbb{N}}$ computes the function $f:\{0,1\}^* \to \{0,1\}^*$ if for every $x \in \{0,1\}^*$ it holds that $C_{|x|}(x) = f(x)$. The size complexity of this family is the function $s:\mathbb{N}\to\mathbb{N}$ such that $s(n)$ is the size of C_n. The circuit complexity of a function f, denoted s_f, is the size complexity of the smallest family of circuits that computes f. An equivalent alternative follows.

Definition 2 (circuit complexity): *The* circuit complexity of $f:\{0,1\}^* \to \{0,1\}^*$ *is the function $s_f:\mathbb{N}\to\mathbb{N}$ such that $s_f(n)$ is the size of the smallest circuit that computes the restriction of f to n-bit strings.*

We stress that non-uniformity is implicit in this definition, because no conditions are made regarding the relation between the various circuits used to compute the function on different input lengths.

We will be interested in the class of problems that are solvable by families of polynomial-size circuits. That is, a problem is solvable by polynomial-size circuits if it can be solved by a function f that has polynomial circuit complexity (i.e., there exists a polynomial p such that $s_f(n) \le p(n)$, for every $n \in \mathbb{N}$).

A detour: uniform families. A family of *polynomial-size* circuits $(C_n)_n$ is called uniform if given n one can construct the circuit C_n in poly(n)-time. More generally:

Definition 3 (uniformity): *A family of circuits $(C_n)_n$ is called* uniform *if there exists an algorithm A that on input n outputs C_n within a number of steps that is polynomial in the size of C_n.*

We note that stronger notions of uniformity have been considered. For example, one may require the existence of a polynomial-time algorithm that on input n and v, returns the label of vertex v as well as the list of its children (or an indication that v is not a vertex in C_n).

Proposition 4 *If a problem is solvable by a uniform family of polynomial-size circuits then it is solvable by a polynomial-time algorithm.*

The converse holds as well. The latter fact follows easily from the proof of the NP-completeness of CSAT (see also the proof of Theorem 6).

Proof: On input x, the algorithm operates in two stages. In the first stage, it invokes the algorithm guaranteed by uniformity condition, on input $n \stackrel{\text{def}}{=} |x|$, and obtains C_n. Next, it invokes the circuit evaluation algorithm (asserted in Theorem 1) on input C_n and x, and obtains $C_n(x)$. Since the size and the description length of C_n are polynomial in n, it follows that each stage of our algorithm runs in polynomial time (in $n = |x|$). Thus, the algorithm emulates the computation of $C_{|x|}(x)$, and does so in time polynomial in the length of its own input (i.e., x). ■

1.2 Machines that take advice

General (non-uniform) families of polynomial-size circuits and uniform families of polynomial-size circuits are two extremes with respect to the "amounts of non-uniformity" in the computing device. Intuitively, in the former, non-uniformity is only bounded by the size of the device, whereas in the latter the amounts of non-uniformity is zero. Here we consider a model that allows to decouple the size of the computing device from the amount of non-uniformity, which may indeed range from zero to the device's size. Specifically, we consider algorithms that "take a non-uniform advice" that depends only on the input length. The amount of non-uniformity will be defined to equal the length of the corresponding advice (as a function of the input length). Here we specialize the definition of such algorithms to the case of polynomial-time algorithms.

Definition 5 (non-uniform polynomial-time and $\mathcal{P}$/poly): *We say that a function f is* computed in polynomial-time with advice of length $\ell : \mathbb{N} \rightarrow \mathbb{N}$ *if these exists a polynomial-time algorithm A and an infinite* advice sequence $(a_n)_{n\in\mathbb{N}}$ *such that*

1. *For every $x \in \{0,1\}^*$, it holds that $A(a_{|x|}, x) = f(x)$.*
2. *For every $n \in \mathbb{N}$, it holds that $|a_n| = \ell(n)$.*

We say that a computational problem can be solved in polynomial-time with advice of length ℓ if a function solving this problem can be computed within these resources. We denote by $\mathcal{P}/\ell$ the class of decision problems that can be solved in polynomial-time with advice of length ℓ, and by $\mathcal{P}$/poly the union of $\mathcal{P}/p$ taken over all polynomials p.

Clearly, $\mathcal{P}/0 = \mathcal{P}$. But allowing some (non-empty) advice increases the power of the class (as we shall see below), and allowing advice of length comparable to the time complexity yields a formulation equivalent to circuit complexity (see Theorem 6). We highlight the greater flexibility available by the formalism of machines that take advice, which allows for separate specification of time complexity and advice length. (This comes at the expense of a more cumbersome formulation when we wish to focus on the case that both measures are equal.)

Relation to families of polynomial-size circuits. As hinted before, the class of problems solvable by polynomial-time algorithms with polynomially bounded advice equals the class of problems solvable by families of polynomial-size circuits. For concreteness, we state this fact for decision problems.

Theorem 6 *A decision problem is in* $\mathcal{P}/\mathrm{poly}$ *if and only if it can solved by a family of polynomial-size circuits.*

More generally, for any function t, the following proof establishes that equivalence of the power of machines having time complexity t and taking advice of length t versus families of circuits of size polynomially related to t.

Proof: Suppose that a problem can be solved by a polynomial-time algorithm A using the polynomially bounded advice sequence $(a_n)_{n\in\mathbb{N}}$. Adapting the proof of the NP-completeness of CSAT, observe that the computation of $A(a_{|x|},x)$ can be emulated by a circuit of $\mathrm{poly}(|x|)$-size, which incorporates $a_{|x|}$ and is given x as input. That is, we construct a circuit C_n such that $C_n(x) = A(a_n,x)$ holds for every $x \in \{0,1\}^n$ (analogously to the way C_x was constructed in the proof of the NP-completeness of CSAT, where $C_x(y) = M_R(x,y)$).

On the other hand, given a family of polynomial-size circuits, we obtain a polynomial-time algorithm for emulating this family using advice that provide the description of the relevant circuits. Specifically, we use the evaluation algorithm asserted in Theorem 1, while using the circuit description as advice. We use the fact that a circuit of size s can be computed using advice of length $O(s\log s)$, where the log factor is due to the fact that a graph with v vertices and e edges can be described by a string of length $2e\log_2 v$. ■

Another perspective. A set S is called sparse if there exists a polynomial p such that for every n it holds that $|S\cap\{0,1\}^n| \le p(n)$. We note that $\mathcal{P}/\mathrm{poly}$ equals the class of sets that are Cook-reducible to a sparse set (see Exercise 13). Thus, SAT is Cook-reducible to a sparse set if and only if $\mathcal{NP} \subset \mathcal{P}/\mathrm{poly}$. In contrast, SAT is Karp-reducible to a sparse set if and only if $\mathcal{NP} = \mathcal{P}$ (see Exercise 21).

The power of $\mathcal{P}/\mathrm{poly}$. We prove the following result.

Theorem 7 (the power of advice, revisited): *The class* $\mathcal{P}/1 \subseteq \mathcal{P}/\mathrm{poly}$ *contains* $\mathcal{P}$ *as well as some undecidable problems.*

Actually, $\mathcal{P}/1 \subset \mathcal{P}/\mathrm{poly}$. Furthermore, by using a counting argument, one can show that for any two polynomially bounded functions $\ell_1,\ell_2:\mathbb{N}\to\mathbb{N}$ such that $\ell_2-\ell_1>0$ is unbounded, it holds that $\mathcal{P}/\ell_1$ is strictly contained in $\mathcal{P}/\ell_2$; see Exercise 14.

Proof: Clearly, $\mathcal{P} = \mathcal{P}/0 \subseteq \mathcal{P}/1 \subseteq \mathcal{P}/\mathrm{poly}$. To prove that $\mathcal{P}/1$ contains some undecidable problems, we note the existence of uncomputable Boolean functions that only depend on their input length. Such a function f' can be obtained from any uncomputable Boolean function f by letting $f'(x) = f(|x|)$, where $|x|$ is viewed as a binary string of length $\log_2|x|$. Thus, there exists an undecidable set $S \subset \{0,1\}^*$ such that for every pair of equal length strings (x,y) it holds that $x \in S$ if and only if $y \in S$. In other words, for every $x \in \{0,1\}^*$ it holds that $x \in S$ if and only if $1^{|x|} \in S$. But such a set is easily decidable in polynomial-time by a machine that takes one bit of advice; that is, consider the algorithm A and the advice sequence $(a_n)_{n\in\mathbb{N}}$ such that $a_n = 1$ if and only if $1^n \in S$ and $A(a,x) = a$ (for $a \in \{0,1\}$ and $x \in \{0,1\}^*$). Note that indeed $A(a_{|x|},x) = 1$ if and only if $x \in S$. ■

2 The Polynomial-time Hierarchy (PH)

We start with an informal motivating discussion, which will be made formal in Section 2.1.

Sets in $\mathcal{NP}$ can be viewed as sets of valid assertions that can be expressed as quantified Boolean formulae using only existential quantifiers. That is, a set S is in $\mathcal{NP}$ if there is a Karp-reduction of S to the problem of deciding whether or not an existentially quantified Boolean formula is valid (i.e., an instance x is mapped by this reduction to a formula of the form $\exists y_1 \cdots \exists y_{m(x)} \phi_x(y_1, ..., y_{m(x)})$).

The conjectured intractability of $\mathcal{NP}$ seems due to the long sequence of existential quantifiers. Of course, if somebody else (i.e., a "prover") were to provide us with an adequate assignment (of the y_i's) whenever such an assignment exists then we would be in good shape. That is, we can efficiently verify proofs of validity of existentially quantified Boolean formulae.

But what if we want to verify the validity of a universally quantified Boolean formulae (i.e., formulae of the form $\forall y_1 \cdots \forall y_m \phi(y_1, ..., y_m)$). Here we seem to need the help of a totally different entity: we need a "refuter" that is guaranteed to provide us with a refutation whenever such exist, and we need to believe that if we were not presented with such a refutation then it is the case that no refutation exists (and hence the universally quantified formulae is valid). Indeed, this new setting (of a "refutation system") is fundamentally different from the setting of a proof system: In a proof system we are only convinced by proofs (to assertions) that we have verified by ourselves, whereas in the "refutation system" we trust the "refuter" to provide evidence against false assertions.[1] Furthermore, there seems to be no way of converting one setting (e.g., the proof system) into another (resp., the refutation system).

Taking an additional step, we may consider a more complicated system in which we use two agents: a "supporter" that tries to provide evidence in favor of an assertion and an "objector" that tries to refute it. These two agents conduct a debate (or an argument) in our presence, exchanging messages with the goal of making us (the referee) rule their way. The assertions that can be proven in this system take the form of general quantified formulae with alternating sequences of quantifiers, where the number of alternations equals the number of rounds of interaction in the said system. We stress that the exact length of each sequence of quantifiers of the same type does not matter, what matters is the number of alternations, denoted k.

The aforementioned system of alternations can be viewed as a two-party game, and we may ask ourselves which of the two parties has a k-move winning strategy. In general, we may consider any (0-1 zero-sum) two-party game, in which the game's position can be efficiently updated (by any given move) and evaluated. For such a fixed game, given an initial position, we may ask whether the first party has a (k-move) winning strategy. It seems that answering this question for some k does not help in answering it for $k+1$. We now turn to formalize the foregoing discussion.

2.1 Alternation of quantifiers

In the following definition, the aforementioned propositional formula ϕ_x is replaced by the input x itself. (Correspondingly, the combination of the Karp-reduction and a formula evaluation algorithm are replaced by the verification algorithm V (see Exercise 17).) This is done in order to make the comparison to the definition of $\mathcal{NP}$ more transparent (as well as to fit the standard presentations). We also replace a sequence of Boolean quantifiers of the same type by a single corresponding quantifier that quantifies over a string of corresponding length.

[1] More formally, in proof systems the soundness condition relies only on the actions of the verifier, whereas completeness also relies on the prover using an adequate strategy. In contrast, in "refutation system" the soundness condition relies on the proper actions of the refuter, whereas completeness does not depend on the refuter's actions.

Definition 8 (the class Σ_k): *For a natural number k, a decision problem $S \subseteq \{0,1\}^*$ is in Σ_k if there exists a polynomial p and a polynomial time algorithm V such that $x \in S$ if and only if*

$$\exists y_1 \in \{0,1\}^{p(|x|)} \forall y_2 \in \{0,1\}^{p(|x|)} \exists y_3 \in \{0,1\}^{p(|x|)} \cdots Q_k y_k \in \{0,1\}^{p(|x|)} \quad \text{s.t. } V(x, y_1, ..., y_k) = 1$$

where Q_k is an existential quantifier if k is odd and is a universal quantifier otherwise.

Note that $\Sigma_1 = \mathcal{NP}$ and $\Sigma_0 = \mathcal{P}$. The Polynomial-time Hierarchy, denoted $\mathcal{PH}$, is the union of all the aforementioned classes (i.e., $\mathcal{PH} = \cup_k \Sigma_k$), and Σ_k is often referred to as the k^{th} level of $\mathcal{PH}$. The levels of the Polynomial-time Hierarchy can also be defined inductively, by defining Σ_{k+1} based on $\Pi_k \stackrel{\text{def}}{=} \text{co}\Sigma_k \stackrel{\text{def}}{=} \{\{0,1\}^* \setminus S : S \in \Sigma_k\}$.

Proposition 9 *The set S is in Σ_{k+1} if and only if there exists a polynomial p and a set $S' \in \Pi_k$ such that $S = \{x : \exists y \in \{0,1\}^{p(|x|)} \text{ s.t. } (x,y) \in S'\}$.*

Proof: Suppose that S is in Σ_{k+1} and let p and V be as in Definition 8. Then define S' as the set of pairs (x,y) such that $|y| = p(|x|)$ and $\forall y_1 \in \{0,1\}^{p(|x|)} \exists y_2 \in \{0,1\}^{p(|x|)} \cdots Q_k y_k \in \{0,1\}^{p(|x|)}$ s.t. $V(x,y,y_1,...,y_k) = 1$. Note that $x \in S$ if and only if there exists $y \in \{0,1\}^{p(|x|)}$ such that $(x,y) \in S'$, and that $S' \in \Pi_k$ (by replacing V with $1 - V$).

On the other hand, suppose that for some polynomial p and a set $S' \in \Pi_k$ it holds that $S = \{x : \exists y \in \{0,1\}^{p(|x|)} \text{ s.t. } (x,y) \in S'\}$. Then for some p' and V' it holds that $(x,y) \in S'$ if and only if $|y| = p(|x|)$ and $\forall y_1 \in \{0,1\}^{p(|x|)} \exists y_2 \in \{0,1\}^{p(|x|)} \cdots Q_k y_k \in \{0,1\}^{p(|x|)}$ s.t. $V'(x,y,y_1,...,y_k) \neq 1$. By suitable encoding (of all y's to the same length) and a trivial modification of V' we conclude that $S \in \Sigma_{k+1}$. ■

Determining the winner in k-move games. Definition 8 can be interpreted as capturing the complexity of determining the winner in certain *efficient two-party game*. Specifically, we refer to two-party games that satisfy the following three conditions:

1. The parties alternate in taking moves that effect the game's (global) position, where each move has a description length that is bounded by a polynomial in the length of the current position.
2. The current position can be updated in polynomial-time based on the previous position and the current party's move.[2]
3. The winner in each position can be determined in polynomial-time.

A set $S \in \Sigma_k$ can be viewed as the set of initial positions (in a suitable game) for which the first party has a k-move winning strategy. Specifically, $x \in S$ if starting at the initial position x, there exists move y_1 for the first party, such that for every response move y_2 of the second party, there exists move y_3 for the first party, etc, such that after k moves the parties reach a position in which the first party wins, where the final position as well as which party wins in it are determined by the predicate V (in Definition 8). That is, $V(x, y_1, ..., y_k) = 1$ if the position that is reached when starting from position x and taking the move sequence $y_1, ..., y_k$ is a winning position for the first party.

[2]Note that, since we consider a constant number of moves, the length of all possible final positions is bounded by a polynomial in the length of the initial position, and thus all items have an equivalent form in which one refers to the complexity as a function of the length of the initial position. The latter form allows for a smooth generalization to games with a polynomial number of moves, where it is essential to state all complexities in terms of the length of the initial position.

The collapsing effect of some equalities. Extending the intuition behind the $\mathcal{NP} \neq \text{co}\mathcal{NP}$ conjecture, it is conjectured that $\Sigma_k \neq \Pi_k$ for every $k \in \mathbb{N}$. The failure of this conjecture causes the collapse of the Polynomial-time Hierarchy to the corresponding level, which violates the conjecture that $\Sigma_{k+1} \neq \Sigma_k$ (which, in turn, may seem as an extension of the conjecture that $\mathcal{NP} \neq \mathcal{P}$).

Proposition 10 *For every k, if $\Sigma_k = \Pi_k$ then $\Sigma_{k+1} = \Sigma_k$, which in turn implies $\mathcal{PH} = \Sigma_k$.*

Proof: Assuming that $\Sigma_k = \Pi_k$, we consider any set S in Σ_{k+1}. By Proposition 9, there exists a polynomial p and a set $S' \in \Pi_k$ such that $S = \{x : \exists y \in \{0,1\}^{p(|x|)} \text{ s.t. } (x,y) \in S'\}$. Using the hypothesis, we infer that $S' \in \Sigma_k$, and it follows that $S \in \Sigma_k$ (by collapsing two adjacent existential quantifiers). We also note that $\Sigma_{k+1} = \Sigma_k$ (or, equivalently, $\Pi_{k+1} = \Pi_k$) implies $\Sigma_{k+2} = \Sigma_{k+1}$ (again by using Proposition 9), and thus $\Sigma_{k+1} = \Sigma_k$ implies $\mathcal{PH} = \Sigma_k$. ■

Decision problems that are Cook-reductions to NP. We review two types of optimization problems, which (under some natural conditions) are computationally equivalent (under Cook reductions). One type of problems referred to finding a solution that have a value exceeding some given threshold, whereas the second type called for finding optimal solutions. Typically, NP-complete optimization problems are problems of the first type, and the corresponding versions of the second type are believed not to be in NP. For example, the problem of deciding whether or not a given graph G has a clique of a given size K is NP-complete. In contract, the problem of deciding whether or not K is the maximum clique size of the graph G is not known (and quite unlikely) to be in $\mathcal{NP}$, although it is Cook-reducible to $\mathcal{NP}$. Thus, the class of decision problems that are Cook-reducible to $\mathcal{NP}$ contains many natural problems that are unlikely to be in $\mathcal{NP}$. The Polynomial-time Hierarchy contains all these problems, because Σ_2 contains all problems that are Cook-reductions to $\mathcal{NP}$ (see Exercise 15).

Complete problems and a relation to AC0. We note that quantified Boolean formulae with a bounded number of quantifier alternation provide complete problems for the various levels of the Polynomial-time Hierarchy (see Exercise 17). We also note the correspondnace between these formulae and (highly uniform) constant-depth circuits of unbounded fan-in that get as input the truth-table of the underlying (quantifier-free) formula (see Exercise 18).

2.2 Non-deterministic oracle machines

The Polynomial-time Hierarchy is commonly defined in terms of non-deterministic polynomial-time (oracle) machines that are given oracle access to a set of the lower level of the same hierarchy. Such machines are defined by combining the definitions of non-deterministic (polynomial-time) machines and oracle machines. Specifically, for an oracle $f : \{0,1\}^* \to \{0,1\}^*$, a non-deterministic machine M, and a string x, one considers the question of whether or not there exists an accepting (non-deterministic) computation of M on input x and access to the oracle f. The class of sets that can be accepted by non-deterministic polynomial-time (oracle) machines with access to f is denoted $\mathcal{NP}^f$. (We note that this notation makes sense because we can associate the class $\mathcal{NP}$ with a collection of machines that lends itself to be extended to oracle machines.) For any class of decision problems $\mathcal{C}$, we denote by $\mathcal{NP}^{\mathcal{C}}$ the union of $\mathcal{NP}^f$ taken over all decision problems f in $\mathcal{C}$. The following result provides an alternative definition of the Polynomial-time Hierarchy.

Proposition 11 *For every $k \in \mathbb{N}$, it holds that $\Sigma_{k+1} = \mathcal{NP}^{\Sigma_k}$.*

Proof: The proof of Proposition 11 follows the ideas of the proof of Proposition 9. The first direction is straightforward: For any $S \in \Sigma_{k+1}$, let $S' \in \Pi_k$ and p be as in Proposition 11, and consider a non-deterministic machine that on input x generates $y \in \{0,1\}^{p(|x|)}$ and accepts if and only if $(x,y) \in S'$. Thus, $S \in \mathcal{NP}^{\Pi_k} = \mathcal{NP}^{\Sigma_k}$ (because we can flip the answer given by the oracle).[3]

For the opposite direction, suppose that the non-deterministic polynomial-time oracle machine M accepts S when given oracle access to $S' \in \Sigma_k$. Note that (unlike the machine constructed in the foregoing argument) machine M may issue several queries to S', and these queries may be determined based on previous oracle answers. To simplify the argument, we assume without loss of generality, that at the very beginning of its execution M guesses (non-deterministic) all oracle answers and accepts only if the actual answers match its guesses. Thus, M's queries to the oracle are determined by its input, denoted x, and its non-deterministic choices, denoted y. We denote by $q^{(i)}(x,y)$ the i^{th} query made by M (on input x and non-deterministic choices y), and by $a^{(i)}(x,y)$ the corresponding (a priori) guessed answer (which is a bit in y). Thus, M accepts x if and only if there exists $y \in \{0,1\}^{\mathrm{poly}(|x|)}$ such that the following two conditions hold:

1. Machine M accepts x on input x and non-deterministic choices y, provided that all oracle answers fit the correspondingly guessed answers $a^{(i)}(x,y)$.

2. All oracle answers fit the correspondingly guessed answers $a^{(i)}(x,y)$; that is, $a^{(i)}(x,y)=1$ if and only if $q^{(i)}(x,y) \in S'$, for every $i = 1,...,q(x,y)$, where $q(x,y) = \mathrm{poly}(|x|)$ is the number of queries made by M.

Denoting the first event by $A(x,y)$ and denoting the verification algorithm of S' by V', it holds that $x \in S$ if and only if

$$\exists y \left(A(x,y) \land \bigwedge_{i=1}^{q(x,y)} \left(((a^{(i)}(x,y)=1) \Leftrightarrow \exists y_1^{(i)} \forall y_2^{(i)} \cdots Q_k y_k^{(i)}\, V'(q^{(i)}(x,y), y_1^{(i)}, ..., y_k^{(i)})=1) \right) \right)$$

The proof is completed by observing that the foregoing expression can be rearranged to fit the definition of Σ_{k+1}. Details follow.

Starting with the foregoing expression, we first pull all quantifiers outside, and obtain a quantified expression with $k+1$ alternations, starting with an existential quantifier.[4] (We get $k+1$ alternations rather than k, because $a^{(i)}(x,y)=0$ introduces an expression of the form $\neg \exists y_1^{(i)} \forall y_2^{(i)} \cdots Q_k y_k^{(i)}\, V'(q^{(i)}(x,y), y_1^{(i)}, ..., y_k^{(i)}) = 1$, which in turn is equivalent to the expression $\forall y_1^{(i)} \exists y_2^{(i)} \cdots \overline{Q}_k y_k^{(i)}\, \neg V'(q^{(i)}(x,y), y_1^{(i)}, ..., y_k^{(i)})=1$).) Once this is done, we may incorporate the computation of all the $q^{(i)}(x,y)$'s (and $a^{(i)}(x,y)$'s) as well as the polynomial number of invocations of V' (and other logical operations) into the new verification algorithm V. It follows that $S \in \Sigma_{k+1}$. ■

A general perspective – what does ${\mathcal{C}_1}^{\mathcal{C}_2}$ mean? By the above discussion it should be clear that the class ${\mathcal{C}_1}^{\mathcal{C}_2}$ can be defined for two complexity classes $\mathcal{C}_1$ and $\mathcal{C}_2$, *provided that $\mathcal{C}_1$ is associated with a class of machines that extends naturally to allow for oracle access.* Actually, the class ${\mathcal{C}_1}^{\mathcal{C}_2}$

[3]Don't get confused by the fact that the class of oracles may not be closed under complementation. From the point of view of the oracle machine, the oracle is merely a function, and the machine may do with its answer whatever it pleases (and in particular negate it).

[4]For example, note that for predicates P_1 and P_2, the expression $\exists y\,(P_1(y) \Leftrightarrow \exists z\, P_2(y,z))$ is equivalent to the expression $\exists y\,((P_1(y) \land \exists z\, P_2(y,z)) \lor ((\neg P_1(y) \land \neg \exists z\, P_2(y,z)))$, which in turn is equivalent to the expression $\exists y \exists z' \forall z''\,((P_1(y) \land P_2(y,z')) \lor ((\neg P_1(y) \land \neg P_2(y,z'')))$. Note that pulling the quantifiers outside in $\land_{i=1}^{t} \exists y^{(i)} \forall z^{(i)} P(y^{(i)}, z^{(i)})$ yields an expression of the type $\exists y^{(1)}, ..., y^{(t)} \forall z^{(1)}, ..., z^{(t)} \land_{i=1}^{t} P(y^{(i)}, z^{(i)})$.

is not defined based on the class $\mathcal{C}_1$ but rather by analogy to it. Specifically, suppose that $\mathcal{C}_1$ is the class of sets recognizable by machines of certain type (e.g., deterministic or non-deterministic) with certain resource bounds (e.g., time and/or space bounds). Then we consider analogous oracle machines (i.e., of the same type and with the same resource bounds), and say that $S \in {\mathcal{C}_1}^{\mathcal{C}_2}$ if there exists such an oracle machine M_1 and a set $S_2 \in \mathcal{C}_2$ such that $M_1^{S_2}$ accepts the set S.

Decision problems that are Cook-reductions to NP, revisited. Using the foregoing notation, the class of decision problems that are Cook-reductions to $\mathcal{NP}$ is denoted $\mathcal{P}^{\mathcal{NP}}$, and thus is a subset of $\mathcal{NP}^{\mathcal{NP}} = \Sigma_2$ (see Exercise 19). In contrast, recall that the class of decision problems that are Karp-reductions to $\mathcal{NP}$ equals $\mathcal{NP}$.

2.3 The P/poly-versus-NP Question and PH

As stated in Section 1, a main motivation for the definition of $\mathcal{P}$/poly is the hope that it can serve to separate $\mathcal{P}$ from $\mathcal{NP}$ (by showing that $\mathcal{NP}$ is not even contained in $\mathcal{P}$/poly, which is a (strict) superset of $\mathcal{P}$). In light of the fact that $\mathcal{P}$/poly extends far beyond $\mathcal{P}$ (and in particular contains undecidable problems), one may wonder if this approach does not run the risk of $\mathcal{NP}$ being in $\mathcal{P}$/poly (even if $\mathcal{P} \neq \mathcal{NP}$). Ideally, we would like to know that $\mathcal{NP} \subset \mathcal{P}$/poly may occur only if $\mathcal{P} = \mathcal{NP}$ (which means that the Polynomial-time Hierarchy collapses to its zero level). The following result may seem to get close, showing that $\mathcal{NP} \subset \mathcal{P}$/poly may occur only if the Polynomial-time Hierarchy collapses to its second level.

Theorem 12 *If $\mathcal{NP} \subset \mathcal{P}$/poly then $\Sigma_2 = \Pi_2$.*

Recall that $\Sigma_2 = \Pi_2$ implies $\mathcal{PH} = \Sigma_2$ (see Proposition 10). Thus, an unexpected behavior of the non-uniform complexity class $\mathcal{P}$/poly implies an unexpected behavior of in the world of uniform complexity (i.e., the ability to reduce any constant number of quantifier alternations into two).

Proof: (We present the proof in terms of machines that take advice, although a presentation in terms of circuits may be somewhat less cumbersome.)

Using Proposition 9, for every $S \in \Pi_2$ there exists a polynomial p and a set $S' \in \mathcal{NP}$ such that $S = \{x : \forall y \in \{0,1\}^{p(|x|)}\ (x,y) \in S'\}$. Using the theorem's hypothesis, it follows that $S' \in \mathcal{P}$/poly, which means that there exists polynomially bounded advice that allow for deciding S' in polynomial-time. That is, there exists a polynomial-time algorithm A and an advice sequence $(a_n)_{n\in\mathbb{N}}$ such that $|a_n| = \text{poly}(n)$ and $A(a_n, \cdot)$ correctly decides the membership of n-bit long strings in S'. Thus, $x \in S$ if and only if there exists a "correct advice" $a \in \{0,1\}^{\text{poly}(|x|)}$ for $(|x|+p(|x|))$-bit long inputs such that for every $y \in \{0,1\}^{p(|x|)}$ it holds that $A(a,(x,y)) = 1$, where the correctness condition means that $A(a,\cdot)$ decides correctly the membership of $(|x|+p(|x|))$-bit long strings in S'. Note that the conditions made have the right form: *there exists a string a such that for all y and z some condition holds* (i.e., $A(a,(x,y)) = 1$ and $A(a,z) = \chi_{S'}(z)$, where $\chi_{S'}(z) = 1$ if $z \in S'$ and $\chi_{S'}(z) = 0$ otherwise).

The problem is that part of the aforementioned condition is not checkable in polynomial-time. We refer to the need to check that the advice is correct (i.e., that it allows to correctly decide S' on $(|x|+p(|x|))$-bit long inputs). Suppose, for a moment, that S' is downwards self-reducible; that is, that deciding whether $z \in S'$ can be reduced to deciding membership in S' of shorter (than z) strings. Then, referring to the aforementioned algorithm A and polynomial p, we revise the foregoing suggestion and assert that $x \in S$ if and only if there exists a sequence of poly($|x|$)-bit long strings $a_1, ..., a_m$, where $m = (|x|+p(|x|))$, such that the following two conditions hold

1. For every $y \in \{0,1\}^{p(|x|)}$, it holds that $A(a_m,(x,y)) = 1$.

2. For $i = 1, ..., m$, for every $z \in \{0,1\}^i$ it holds that $A(a_i, z) = 1$ if and only if $z \in S'$.

   Using downwards self-reducible this condition can be expressed by referring to the values of $A(a_j, z')$ for some $j < i$ and $z' \in \{0,1\}^j$. Specifically, we check the value of $A(a_i, z) = 1$ against the value obtained by reducing the question of whether $z \in S'$ to the membership of shorter strings in S', where the latter decisions are made by relying on the hypothesis that we have correct advice for shorter strings. That is, we verify the correctness of the advice a_i, by relying on the correctness of the advice $a_1, ..., a_{i-1}$. (Needless to say, the correctness of the advice a_1 can be verified in a constant number of steps by using a brute force algorithm for deciding membership in S'.)

There is a minor problem with this plan, because we have assumed that S' is downwards self-reducible, while S' may be an arbitrary set in $\mathcal{NP}$. The solution is to reduce S' to SAT, and rely on the fact that SAT is downwards self-reducible. The latter fact is implicit in the proof of the self-reducibility of SAT; that is, $\phi \in$ SAT if and only if either $\phi_{x_1=0} \in$ SAT or $\phi_{x_1=1} \in$ SAT, where $\phi_{x_1=\sigma}$ denotes the formula obtained from ϕ by setting the first variable (denoted x_1) to σ and simplifying the formula in the obvious way (thus reducing its length).

For simplicity, in the rest of the proof, we rely on the fact that S' is Karp-reducible to SAT (rather than only Cook-reducible to it). Specifically, let f be a Karp-reduction of S' to SAT. Thus, $x \in S$ if and only if $\forall y \in \{0,1\}^{p(|x|)}$ $\phi_{x,y} \in$ SAT. where $\phi_{x,y} \stackrel{\rm def}{=} f(x,y)$. Using the hypothesis, we have SAT $\in \mathcal{P}/\text{poly}$, and thus there exists a polynomial-time algorithm A that solves SAT using advice of polynomial length. Now, we assert that $x \in S$ if and only if there exists a sequence of poly($|x|$)-bit long strings $a_1, ..., a_m$, where $m = \text{poly}(|x|)$ satisfies $m \geq \max_{y \in \{0,1\}^{p(|x|)}}\{|f(x,y)|\}$, such that the following two conditions hold:

1. For every $y \in \{0,1\}^{p(|x|)}$, it holds that $A(a_{|f(x,y)|}, f(x,y)) = 1$.

2. For $i = 1, ..., m$, for every $\phi \in \{0,1\}^i$ it holds that $A(a_i, \phi) = 1$ if and only if $\phi \in$ SAT. This condition is checked as follows.

   For every (non-empty) formula ϕ, we denote by ϕ_σ the formula resulting from ϕ by setting its first variable to σ and making the obvious simplifications (i.e., omitting constants from clauses and omitting empty clauses). Thus, $|\phi_\sigma| < |\phi|$.

   For every $i > 1$ and $\phi \in \{0,1\}^i$, we check that $A(a_i, \phi) = 1$ if and only if either $A(a_{|\phi_0|}, \phi_0) = 1$ or $A(a_{|\phi_1|}, \phi_1) = 1$. For $\phi \in \{0,1\}^1$, we check that $A(a_1, \phi) = 1$ if and only if ϕ is satisfiable (which can be determined in constant time). (Indeed, for any threshold $t = O(\log|x|)$, we could have distinguished the cases $i > t$ and $i \leq t$, where the latter can be handled in time poly(2^t).)

Observe that the expression obtained for membership in S is indeed of the Σ_2-form. The theorem follows. ■

A technical perspective. The proof of Theorem 12 implies that *any set* $S' \in \mathcal{P}/\text{poly}$ *that is downwards self-reducible is in* Σ_2. The point is that the existential quantifier selects advice that are checked for correctness using the universal quantifier, while relying on the downwards self-reducibility process.

Notes

The class $\mathcal{P}$/poly was defined by Karp and Lipton [4] as part of a general formulation of "machines which take advise" [4]. They have noted the equivalence to the traditional formulation of polynomial-size circuits as well as the effect of uniformity (Proposition 4).

The Polynomial-Time Hierarchy ($\mathcal{PH}$) was introduced by Stockmeyer [5]. A third equivalent formulation of $\mathcal{PH}$ (via "alternating machines") can be found in [1].

The effect of $\mathcal{NP} \subset \mathcal{P}$/poly on the Polynomial-time hierarchy (i.e., Theorem 12) was observed by Karp and Lipton [4]. This interesting connection between non-uniform and uniform complexity provides the main motivation for presenting $\mathcal{P}$/poly and $\mathcal{PH}$ in the same lecture. We (gain) call the reader's attention to the proof of Theorem 12, and specifically to the inspiring use of downwards self-reducibility.

Exercise 13 (sparse sets) A set $S \subset \{0,1\}^*$ is called sparse if there exists a polynomial p such that $|S \cap \{0,1\}^n| \leq p(n)$ for every n.

1. Prove that any sparse set is in $\mathcal{P}$/poly.
2. Prove that a set is in $\mathcal{P}$/poly if and only if it is Cook-reducible to some sparse set.

Guideline: For the forward direction of Part 2, encode the advice sequence $(a_n)_{n\in\mathbb{N}}$ as a sparse set $\{(1^n, i, \sigma_{n,i}) : n \in \mathbb{N},\, i \leq |a_n|\}$, where $\sigma_{n,i}$ is the i^{th} bit of a_n. For the opposite direction, note that on input x the Cook-reduction makes queries of length at most poly($|x|$), and so all the relevant strings in the target of the reduction can be encoded in the n^{th} advice.

Exercise 14 (advise hierarchy) Prove that for any two functions $\ell, \delta : \mathbb{N} \to \mathbb{N}$ such that $\ell(n) < 2^{n-1}$ and δ is unbounded, it holds that $\mathcal{P}/\ell$ is strictly contained in $\mathcal{P}/(\ell + \delta)$.
(Hint: for every sequence $\overline{a} = (a_n)_{n\in\mathbb{N}}$ such that $|a_n| = \ell(n) + \delta(n)$, consider the set $S_{\overline{a}}$ where $x \in S$ if and only if x is the i^{th} string of length $|x|$ and $i \leq |a_{|x|}|$ and the i^{th} bit in $a_{|x|}$ is 1.)

Exercise 15 Prove that Σ_2 contains all sets that are Cook-reducible to $\mathcal{NP}$.
(Hint: This is quite obvious when using the definition of Σ_2 as presented in Section 2.2; see Exercise 19.)

Exercise 16 (the class Π_i) Prove that for any natural number k, a decision problem $S \subseteq \{0,1\}^*$ is in Π_k if there exists a polynomial p and a polynomial time algorithm V such that $x \in S$ if and only if

$$\forall y_1 \in \{0,1\}^{p(|x|)} \exists y_2 \in \{0,1\}^{p(|x|)} \forall y_3 \in \{0,1\}^{p(|x|)} \cdots Q_k y_k \in \{0,1\}^{p(|x|)} \quad \text{s.t. } V(x, y_1, ..., y_k) = 1$$

where Q_k is a universal quantifier if k is odd and is an existential quantifier otherwise.
(Recall that Π_k was defined as coΣ_k, which in turn is defined as $\{\{0,1\}^* \setminus S : S \in \Sigma_k\}$.)

Exercise 17 (complete problems for $\mathcal{PH}$) A k-alternating quantified Boolean formula is a quantified Boolean formula with up to k alternations between existential and universal quantifiers, starting with an existential quantifier. For example, $\exists z_1 \exists z_2 \forall z_3 \phi(z_1, z_2, z_3)$ (where the z_i's are Boolean variables) is a 2-alternating quantified Boolean formula. Prove that the problem of deciding whether or not a k-alternating quantified Boolean formula is valid is Σ_k-complete. That is, denoting the aforementioned problem by `kQBF`, prove that `kQBF` is in Σ_k and every problem in Σ_k is Karp-reducible to `kQBF`.

Exercise 18 (on the relation between $\mathcal{PH}$ and $\mathcal{AC}^0$) Note that there is an obvious analogy between $\mathcal{PH}$ and constant depth circuits of unbounded fan-in, where existential (resp., universal) quantifiers are represented by "large" $\bigvee$ (rep., $\bigwedge$) gates. To articulate this relationship, consider the following definitions.

- A family of circuits $\{C_N\}$ is called highly uniform if there exists a polynomial-time algorithm that answers local queries regarding the structure of the relevant circuit. Specifically, on input (N, u, v), the algorithm determines the type of gates represented by the vertices u and v in C_N as well as whether there exists a directed edge from u to v. Note that this algorithm operates in time that polylogarithmic in the size of C_N.

  We focus on family of polynomial-size circuits, meaning that the size of C_N is polynomial in N, which in turn represents the number of inputs to C_N.

- Fixing a polynomial p, a *p-succinctly represented input* $X \in \{0,1\}^N$ is a circuit c_X of size at most $p(\log_2 N)$ such that for every $i \in [N]$ it holds that $c_X(i)$ equals the i^{th} bit of X.

- For a fixed family of highly uniform circuits $\{C_N\}$ and a fixed polynomial p, the problem of evaluating a succinctly represented input is defined as follows. Given p-succinct representation of an input $X \in \{0,1\}^N$, determine whether or not $C_N(X) = 1$.

For every k and every $S \in \Sigma_k$, show that there exists a family of highly uniform unbounded fan-in circuits of depth k and polynomial-size such that S is Karp-reducible to evaluating a succinctly represented input (with respect to that family of circuits). That is, the reduction should map an instance $x \in \{0,1\}^n$ to a p-succinct representation of some $X = X_x \in \{0,1\}^N$ such that $x \in S$ if and only if $C_N(X) = 1$. (Note that X is represented by a circuit c_X of size at most $p(\log_2 N)$, and that it must hold that $|c_X| \leq \text{poly}(n)$ and thus $N \leq \exp(\text{poly}(n))$.)[5]

Guideline: Let $S \in \Sigma_k$ and let V be the corresponding verification algorithm as in Definition 8. That is, $x \in S$ if and only if $\exists y_1 \forall y_2 \cdots Q_k y_k$, where each $y_i \in \{0,1\}^{\text{poly}(|x|)}$ such that $V(x, y_1, ..., y_k) = 1$. Then, for $m = \text{poly}(|x|)$ and $N = 2^{k \cdot m}$, consider the circuit $C_N(Z) = \bigvee_{i_1 \in [2^m]} \bigwedge_{i_2 \in [2^m]} \cdots Q'_{i_k \in [2^m]} Z_{i_1, i_2, ..., i_k}$, and the problem of evaluating it at the input consisting of the truth-table of $V(x, \cdots)$ (i.e., when setting $Z_{i_1, i_2, ..., i_k} = V(x, i_1, ..., i_k)$). Note that the size of C_N is $O(N)$.[6]

Exercise 19 Verify the following facts:

1. For every $k \geq 1$, it holds that $\Sigma_k \subseteq \mathcal{P}^{\Sigma_k} \subseteq \Sigma_{k+1}$.

   (Note that, for any complexity class $\mathcal{C}$, the class $\mathcal{P}^{\mathcal{C}}$ is the class of sets that are Cook-reducible to some set in $\mathcal{C}$. In particular, $\mathcal{P}^{\mathcal{P}} = \mathcal{P}$.)
2. For every $k \geq 1$, $\Pi_k \subseteq \mathcal{P}^{\Pi_k} \subseteq \Pi_{k+1}$.

   (Hint: For any complexity class $\mathcal{C}$, it holds that $\mathcal{P}^{\mathcal{C}} = \mathcal{P}^{\text{co}\mathcal{C}}$ and $\mathcal{P}^{\mathcal{C}} = \text{co}\mathcal{P}^{\mathcal{C}}$.)
3. For every $k \geq 1$, it holds that $\Sigma_k \subseteq \Pi_{k+1}$ and $\Pi_k \subseteq \Sigma_{k+1}$. Thus, $\mathcal{PH} = \cup_k \Pi_k$.

[5] Assuming $\mathcal{P} \neq \mathcal{NP}$, it cannot be that $N \leq \text{poly}(n)$ (because circuit evaluation can be performed in time polynomial in the size of the circuit).

[6] Advanced comment: the limitations of $\mathcal{AC}^0$ circuits imply limitations on the functions of the truth-table of a *generic* V that such the aforementioned circuits C_N can compute. Unfortunately, these limitations do not seem to provide useful information on the limitations of functions that are confined to truth-tables that have succinct representation (as in the case of the actual V). This fundamental problem is "resolved" in the context of "relativization" by providing V with oracle access to an arbitrary input of length N (or so); cf. [3].

Exercise 20 Referring to the notion of downwards self-reducibility (as defined in the proof of Theorem 12), prove that if $\mathcal{P} = \mathcal{NP}$ (resp., $\mathcal{NP} = \text{co}\mathcal{NP}$) then any set $S' \in \mathcal{P}/\text{poly}$ that is downwards self-reducible is in $\mathcal{P}$ (resp., in $\mathcal{NP}$).
(Hint: any set $S' \in \mathcal{P}/\text{poly}$ that is downwards self-reducible is in Σ_2.)

Exercise 21 In continuation to Part 2 of Exercise 13, we consider the class of sets that are Karp-reducible to a sparse set. It can be proved that this class contains `SAT` if and only if $\mathcal{P} = \mathcal{NP}$ (see [2]). Here, we only consider the special case in which the sparse set is contained in another sparse set that is in $\mathcal{P}$ (e.g., the latter set may be $\{1\}^*$, in which case the former set may be an arbitrary unary set). Specifically, we claim that

if `SAT` *is Karp-reducible to a set* $S \subseteq G$ *such that* $G \in \mathcal{P}$ *and* G *is sparse then* `SAT` $\in \mathcal{P}$.

Using the hypothesis, we outline a polynomial-time procedure for solving the search problem of SAT, and leave providing the details as an exercise. The procedure conducts a DFS on the tree of all possible partial truth assignment to the input formula, while truncating the search at nodes that are roots of sub-trees that were already proved to contain no satisfying assignment (at the leaves).[7]

Guideline: The key observation is that each internal node (which yields a formula derived from the initial formulae by instantiating the corresponding partial truth assignment) is mapped by the reduction either to a string not in G (in which case we conclude that the sub-tree contains no satisfying assignments and backtrack from this node) or to a string in G. In the latter case, unless we already know that this string is not in S, we start a scan of the sub-tree rooted at this node. but once we backtrack from this internal node, we know that the corresponding element of G is not in S, and we will never extend a node mapped to this element again. Also note that once we reach a leaf, we can check by ourselves whether or not it corresponds to a satisfying assignment to the initial formula.

(Hint: When analyzing the forgoing procedure, note that on input an n-variable formulae ϕ the number of times we start to scan a sub-tree is at most $n \cdot |\cup_{i=1}^{\text{poly}(|\phi|)} \{0,1\}^i \cap (G \setminus S)|$.)

References

[1] A.K. Chandra, D.C. Kozen and L.J. Stockmeyer. Alternation. *Journal of the ACM*, Vol. 28, pages 114–133, 1981.

[2] S. Fortune. A Note on Sparse Complete Sets. *SIAM Journal on Computing*, Vol. 8, pages 431–433, 1979.

[3] M.L. Furst, J.B. Saxe, and M. Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory*, Vol. 17 (1), pages 13–27, 1984. Preliminary version in *22nd FOCS*, 1981.

[4] R.M. Karp and R.J. Lipton. Some connections between nonuniform and uniform complexity classes. In *12th ACM Symposium on the Theory of Computing*, pages 302-309, 1980.

[5] L.J. Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, Vol. 3, pages 1–22, 1977.

LaTeX Warning: Citation 'Stock77' on page 11 undefined on input LaTeX Warning: Citation 'KL' on page 11 undefined on input line

[7] For an n-variable formulae, the leaves of the tree correspond to all possible n-bit long strings, and an internal node corresponding to τ is the parent of nodes corresponding to $\tau 0$ and $\tau 1$.