Chapter 3

Variations on P and NP

Cast a cold eye
On life, on death.
Horseman, pass by!

W.B. Yeats, Under Ben Bulben

In this chapter 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; still, the resulting classes are referred to quite frequently in the literature.

Summary: Non-uniform polynomial-time (P/poly) captures efficient computations that are carried out by devices that can each only handle inputs of a specific length. 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 quantified Boolean formulae 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. This result is commonly interpreted as supporting the common belief that non-uniformity is irrelevant to the P-vs-NP Question; that is, although P/poly extends beyond the class P, is is believed that P/poly does not contain NP.

Except for the latter result, which is presented in Section 3.2.3, the treatments of P/poly (in Section 3.1) and of the Polynomial-time Hierarchy (in Section 3.2) are independent of one another.

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

In this section we consider two formulations of the notion of non-uniform polynomial-time, based on the two models of non-uniform computing devices that were presented in Section 1.2.4. That is, we specialize the treatment of non-uniform computing devices, provided in Section 1.2.4, to the case of polynomially bounded complexities. It turns out that both (polynomially bounded) formulations allow for solving the same class of computational problems, which is a strict superset of the class of problems solvable by polynomial-time algorithms.

The two models of non-uniform computing devices are Boolean circuits and "machines that take advice" (cf. §1.2.4.1 and §1.2.4.2, respectively). 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) advice of polynomially bounded length.

The main motivation for considering non-uniform polynomial-size circuits is that their computational 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 (i.e., uniformity) that seems secondary[1] and is not well-understood may turn out fruitful. In particular, the (non-uniform) circuit model facilitates a low-level analysis of the evolution of a computation, and allow for the application of combinatorial techniques. The benefit of this approach has been demonstrated in the study of restricted classes of circuits (see Sections B.2.2 and B.2.3).

The main motivation for considering polynomial-time algorithms that take polynomially bounded advice is that such devices are useful in modeling auxiliary information that is available to possible efficient strategies that are of interest to us. We mention two such settings. In cryptography (see Appendix C), the advice is used for accounting for auxiliary information that is available to an adversary. In the context of derandomization (see Section 8.3), the advice is used for accounting for the main input to the randomized algorithm. In addition, 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.

3.1.1 Boolean Circuits

We refer the reader to §1.2.4.1 for a 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 result stated in §1.2.4.1:

Theorem 3.1 (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 $C(x)$.*

[1] The common belief is that the issue of non-uniformity is irrelevant to the P-vs-NP Question; that is, that resolving the latter question by proving that $\mathcal{P} \neq \mathcal{NP}$ is not easier than proving that NP does not have polynomial-size circuits. For further discussion see Appendix B.2 and Section 3.2.3.

Recall that the algorithm works by performing the "value-determination" process that underlies the definition of the computation of the circuit on a given input. This process assigns values to each of the circuit vertices based on the values of its children (or the values of the corresponding bit of the input, in the case of an input-terminal vertex).

Circuit size as a complexity measure. We recall the definitions of circuit complexity presented in to §1.2.4.1: 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 thus we 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 3.2 (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.

An interesting feature of Definition 3.2 is that, unlike in the case of uniform model of computation, circuit complexity is the actual complexity of the function rather than an upper-bound on its complexity (cf. §1.2.3.4 and Section 4.2.1). This is a consequence of the fact that the circuit model has no "free parameters" (e.g., the finite algorithm in use)[2] and that the issue of constructibility of complexity measures (cf., e.g., Definition 4.2) is irrelevant to it.

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 $s_f(n) \leq p(n)$, 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 3.3 (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*.*

[2] **Advanced comment:** Note that such "free parameters" underly linear speedup results such as Exercise 4.7, which in turn prevent the specification of the exact complexities of functions.

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). For further discussion see Section 5.2.3.

Proposition 3.4 *If a problem is solvable by a uniform family of polynomial-size circuits then it is solvable by a polynomial-time algorithm.*

As was hinted in §1.2.4.1, the converse holds as well. The latter fact follows easily from the proof of Theorem 2.20 (see also the proof of Theorem 3.6).

Proof: On input x, the algorithm operates in two stages. In the first stage, it invokes the algorithm guaranteed by the uniformity condition, on input $n \stackrel{\text{def}}{=} |x|$, and obtains the circuit C_n. Next, it invokes the circuit evaluation algorithm (asserted in Theorem 3.1) 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 (i.e., polynomial 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). ■

3.1.2 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). Thus, we specialize Definition 1.12 to the case of polynomial-time algorithms.

Definition 3.5 (non-uniform polynomial-time and $\mathcal{P}$/poly): *We say that a function f is* computed in polynomial-time with advice of length $\ell : \mathbb{N} \to \mathbb{N}$ *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 (see Theorem 3.7). and allowing advice of length comparable to the time complexity yields a formulation equivalent to circuit complexity (see Theorem 3.6). 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. (Indeed, 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 3.6 *A decision problem is in* $\mathcal{P}/\mathrm{poly}$ *if and only if it can be 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 Sketch: Suppose that a problem can be solved by a polynomial-time algorithm A using the polynomially bounded advice sequence $(a_n)_{n\in\mathbb{N}}$. We obtain a family of polynomial-size circuits that solves the same problem by adapting the proof of Theorem 2.20. Specifically, we 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 Theorem 2.20, where it holds that $C_x(y) = M_R(x, y)$ for every y of adequate length).

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 3.1, while using the circuit's description as advice. That is, we use the fact that a circuit of size s can be described by a string 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 $|S \cap \{0,1\}^n| \leq p(n)$. We note that $\mathcal{P}/\mathrm{poly}$ equals the class of sets that are Cook-reducible to a sparse set (see Exercise 3.2). 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 3.12).

The power of $\mathcal{P}/\mathrm{poly}$**.** In continuation to Theorem 1.13 (which focuses on advice and ignores the time complexity of the machine that takes this advice), we prove the following (stronger) result.

Theorem 3.7 (the power of advice, revisited): *The class* $\mathcal{P}/1 \subseteq \mathcal{P}/\text{poly}$ *contains* $\mathcal{P}$ *as well as some undecidable problems.*

Actually, $\mathcal{P}/1 \subset \mathcal{P}/\text{poly}$. 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 3.3.

Proof: Clearly, $\mathcal{P} = \mathcal{P}/0 \subseteq \mathcal{P}/1 \subseteq \mathcal{P}/\text{poly}$. To prove that $\mathcal{P}/1$ contains some undecidable problems, we review the proof of Theorem 1.13. The latter proof established the existence of uncomputable Boolean function that only depend on their input length. That is, 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 $x \in S$. ■

3.2 The Polynomial-time Hierarchy (PH)

We start with an informal motivating discussion, which will be made formal in Section 3.2.1.

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 (to 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.[3] Furthermore, there seems

[3]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

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 efficiently 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 type of question for some fixed k does not necessarily allow answering it for $k+1$. We now turn to formalize the foregoing discussion.

3.2.1 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 3.7).) 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 all strings of the corresponding length.

Definition 3.8 (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*

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

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

"refutation system" the soundness condition relies on the proper actions of the refuter, whereas completeness does not depend on the refuter's actions.

can also be defined inductively, by defining Σ_{k+1} based on $\Pi_k \stackrel{\text{def}}{=} \text{co}\Sigma_k$, where $\text{co}\Sigma_k \stackrel{\text{def}}{=} \{\{0,1\}^* \setminus S : S \in \Sigma_k\}$ (cf. Eq. (2.4)).

Proposition 3.9 *For every $k \geq 0$, a 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 3.8. Then define S' as the set of pairs (x,y) such that $|y| = p(|x|)$ and

$$\forall z_1 \in \{0,1\}^{p(|x|)} \exists z_2 \in \{0,1\}^{p(|x|)} \cdots Q_k z_k \in \{0,1\}^{p(|x|)} \text{ s.t. } V(x,y,z_1,...,z_k) = 1\,.$$

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 $S' \in \Pi_k$ (see Exercise 3.6).

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

$$\forall z_1 \in \{0,1\}^{p'(|x|)} \exists z_2 \in \{0,1\}^{p'(|x|)} \cdots Q_k z_k \in \{0,1\}^{p'(|x|)} \text{ s.t. } V'(x,y,z_1,...,z_k) \neq 1$$

(see Exercise 3.6 again). By suitable encoding (of y and the z_i's as strings of length $\max(p(|x|), p'(|x|))$) and a trivial modification of V', we conclude that $S \in \Sigma_{k+1}$. ■

Determining the winner in k-move games. Definition 3.8 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.[4]

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 a move y_1 for the first party, such that for every response move y_2 of the second party, there exists a move y_3 for the first party, etc, such that after k moves the parties reach a position in which the first party wins,

[4]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 (as in Section 5.4), where it is essential to state all complexities in terms of the length of the initial position.

where the final position as well as which party wins in it are determined by the predicate V (in Definition 3.8). 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.

The collapsing effect of some equalities. Extending the intuition that underlies the $\mathcal{NP} \neq \text{co}\mathcal{NP}$ conjecture, it is commonly 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.

Proposition 3.10 *For every $k \geq 1$, if $\Sigma_k = \Pi_k$ then $\Sigma_{k+1} = \Sigma_k$, which in turn implies $\mathcal{PH} = \Sigma_k$.*

The converse also holds (i.e., $\mathcal{PH} = \Sigma_k$ implies $\Sigma_{k+1} = \Sigma_k$ and $\Sigma_k = \Pi_k$). Needless to say, Proposition 3.10 does not seem to hold for $k = 0$.

Proof: Assuming that $\Sigma_k = \Pi_k$, we first show that $\Sigma_{k+1} = \Sigma_k$. For any set S in Σ_{k+1}, by Proposition 3.9, 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 so (using Proposition 3.9 and $k \geq 1$) there exists a polynomial p' and a set $S'' \in \Pi_{k-1}$ such that $S' = \{x' : \exists y' \in \{0,1\}^{p'(|x'|)} \text{ s.t. } (x',y') \in S''\}$. It follows that

$$S = \{x : \exists y \in \{0,1\}^{p(|x|)} \exists z \in \{0,1\}^{p'(|(x,y)|)} \text{ s.t. } ((x,y),z) \in S''\}.$$

By collapsing the two adjacent existential quantifiers (and using Proposition 3.9 yet again), we conclude that $S \in \Sigma_k$. This proves the first part of the proposition.

Turning to the second part, we 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 3.9), and similarly $\Sigma_{j+2} = \Sigma_{j+1}$ for any $j \geq k$. Thus, $\Sigma_{k+1} = \Sigma_k$ implies $\mathcal{PH} = \Sigma_k$. ■

Decision problems that are Cook-reductions to NP. The Polynomial-time Hierarchy contains all decision problems that are Cook-reductions to $\mathcal{NP}$ (see Exercise 3.4). As shown next, the latter class contains many natural problems. Recall that in Section 2.2.2 we defined two types of optimization problems and showed that under some natural conditions these two types are computationally equivalent (under Cook reductions). Specifically, one type of problems referred to finding solutions that have a value *exceeding some given threshold*, whereas the second type called for finding *optimal solutions*. In Section 2.3 we presented several problems of the first type, and proved that they are NP-complete. We note that corresponding versions of the second type are believed not to be in NP. For example, we discussed the problem of deciding whether or not a given graph G has a clique of a given size K, and showed that it 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.

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 3.7). We also note the correspondence 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 3.8).

3.2.2 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 in the lower level of the same hierarchy. Such machines are defined by combining the definitions of non-deterministic (polynomial-time) machines (cf. Definition 2.7) and oracle machines (cf. Definition 1.11). Specifically, for an oracle $f : \{0,1\}^* \to \{0,1\}^*$, a non-deterministic oracle 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 3.11 *For every $k \geq 1$, it holds that $\Sigma_{k+1} = \mathcal{NP}^{\Sigma_k}$.*

Proof: The first direction (i.e., $\Sigma_{k+1} \subseteq \mathcal{NP}^{\Sigma_k}$) is almost straightforward: For any $S \in \Sigma_{k+1}$, let $S' \in \Pi_k$ and p be as in Proposition 3.9; that is, $S = \{x : \exists y \in \{0,1\}^{p(|x|)} \text{ s.t. } (x,y) \in S'\}$. Consider the non-deterministic oracle machine that, on input x, non-deterministically generates $y \in \{0,1\}^{p(|x|)}$ and accepts if and only if (the oracle indicates that) $(x,y) \in S'$. This machine demonstrates that $S \in \mathcal{NP}^{\Pi_k} = \mathcal{NP}^{\Sigma_k}$, where the equality holds by letting the oracle machine flip each (binary) answer that is provided by the oracle.[5]

For the opposite direction (i.e., $\mathcal{NP}^{\Sigma_k} \subseteq \Sigma_{k+1}$), let M be a non-deterministic polynomial-time oracle machine that 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 machine 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

[5] Do not 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).

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, when for every i it holds that the i^{th} oracle query made by M is answered by the value $a^{(i)}(x,y)$. We stress that we do not assume here that these answers are consistent with S'; we merely refer to the decision of M on a given input, when it makes a specific sequence of non-deterministic choices, and is given specific oracle answers.

2. Each bit $a^{(i)}(x,y)$ is consistent with S'; that is, for every i, it holds that $a^{(i)}(x,y)=1$ if and only if $q^{(i)}(x,y) \in S'$.

Denoting the first event by $A(x,y)$ and letting $q(x,y) \leq \mathrm{poly}(|x|)$ denote the number of queries made by M, it follows that $x \in S$ if and only if

$$\exists y \left(A(x,y) \wedge \bigwedge_{i=1}^{q(x,y)} \left((a^{(i)}(x,y)=1) \Leftrightarrow (q^{(i)}(x,y) \in S') \right) \right).$$

Denoting the verification algorithm of S' by V', it holds that $x \in S$ if and only if

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

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.[6] (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 foregoing 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*

[6] 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 $\exists y \, ((P_1(y) \wedge \exists z \, P_2(y,z)) \vee ((\neg P_1(y) \wedge \neg \exists z \, P_2(y,z)))$, which in turn is equivalent to the expression $\exists y \exists z' \forall z'' \, ((P_1(y) \wedge P_2(y,z')) \vee ((\neg P_1(y) \wedge \neg P_2(y,z''))$. Note that pulling the quantifiers outside in $\wedge_{i=1}^{t} \exists y^{(i)} \forall z^{(i)} P(y^{(i)}, z^{(i)})$ yields an expression of the type $\exists y^{(1)}, ..., y^{(t)} \forall z^{(1)}, ..., z^{(t)} \wedge_{i=1}^{t} P(y^{(i)}, z^{(i)})$.

in a way that allows for oracle access. Actually, the class ${\mathcal{C}_1}^{\mathcal{C}_2}$ *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 that are recognizable (or rather accepted) 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 an adequate oracle machine M_1 (i.e., of this type and resource bounds) 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 3.9). In contrast, recall that the class of decision problems that are Karp-reductions to $\mathcal{NP}$ equals $\mathcal{NP}$.

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

As stated in Section 3.1, 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 asking too much (because it may be that $\mathcal{NP}$ is in $\mathcal{P}$/poly even if $\mathcal{P} \neq \mathcal{NP}$). The common feeling is that the added power of non-uniformity is irrelevant with respect to the P-vs-NP Question. Ideally, we would like to know that $\mathcal{NP} \subset \mathcal{P}$/poly may occur only if $\mathcal{P} = \mathcal{NP}$ (which may be phrased as saying that the Polynomial-time Hierarchy collapses to its zero level). The following result seems to get close to such an implication, showing that $\mathcal{NP} \subset \mathcal{P}$/poly may occur only if the Polynomial-time Hierarchy collapses to its second level.

Theorem 3.12 *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 3.10). Thus, an unexpected behavior of the non-uniform complexity class $\mathcal{P}$/poly implies an unexpected behavior in the world of uniform complexity (i.e., the ability to reduce any constant number of quantifier alternations to two quantifier alternations).

Proof: Using the hypothesis (i.e., $\mathcal{NP} \subset \mathcal{P}$/poly) and starting with an arbitrary set $S \in \Pi_2$, we shall show that $S \in \Sigma_2$. Loosely speaking, $S \in \Pi_2$ means that $x \in S$ if and only if for all y there exists a z such that some (fixed) polynomial-time verifiable condition regarding (x, y, z) holds. Note that the residual condition regarding (x, y) is of the NP-type, and thus (by the hypothesis) it can be verified by a polynomial-size circuit. This suggests saying that $x \in S$ if and only if there exists an *adequate* circuit C such that for all y it holds that $C(x, y) = 1$. Thus, we managed to switch the order of the universal and existential quantifiers. Specifically, the resulting assertion is of the desired Σ_2-type provided that we can either verify the *adequacy condition* in co$\mathcal{NP}$ (or even in Σ_2) or keep out of trouble even

in the case that $x \notin S$ and C is inadequate. In the following proof we implement the latter option by observing that the hypothesis yields small circuits for NP-search problems (and not only for NP-decision problems). Specifically, we obtain (small) circuits that, given (x,y), find an NP-witness for (x,y) (whenever such a witness exists), and rely on the fact that we can efficiently verify the correctness of NP-witnesses. (The alternative approach of providing a coNP-type procedure for verifying the adequacy of the circuit is pursued in Exercise 3.11.)

Let S be an arbitrary set in Π_2. Then, by Proposition 3.9, there exists a polynomial p and a set $S' \in \mathcal{NP}$ such that $S = \{x : \forall y \in \{0,1\}^{p(|x|)}\ (x,y) \in S'\}$. Let $R' \in \mathcal{PC}$ be the witness-relation corresponding to S'; that is, there exists a polynomial p', such that $x' = \langle x,y\rangle \in S'$ if and only if there exists $z \in \{0,1\}^{p'(|x'|)}$ such that $(x',z) \in R'$. It follows that

$$S = \{x : \forall y \in \{0,1\}^{p(|x|)} \exists z \in \{0,1\}^{p'(|\langle x,y\rangle|)}\ (\langle x,y\rangle, z) \in R'\}.$$

By the reduction of $\mathcal{PC}$ to $\mathcal{NP}$ (see the proof of Theorem 2.6 and further discussion in Section 2.2.1), the theorem's hypothesis (i.e., $\mathcal{NP} \subseteq \mathcal{P}/\mathrm{poly}$) implies the existence of polynomial-size circuits for solving the search problem of R'. Using the existence of these circuits, it follows that for any $x \in S$ there exists a small circuit C' such that for every y it holds that $C'(x,y) \in R'(x,y)$, whereas for any $x \notin S$ there exists a y such that $\langle x,y\rangle \notin S'$ and hence $C'(x,y) \notin R'(x,y)$ for any circuit C' (for the trivial reason that $R'(x,y) = \emptyset$). But let us first spell-out what we mean by polynomial-size circuits for solving a search problem as well as further justify their existence for the search problem of R'.

In Section 3.1, we have focused on polynomial-size circuits that solve decision problems. However, the definition sketched in Section 3.1.1 also applies to solving search problems, provided that an appropriate encoding is used for allowing solutions of possibly varying lengths (for instances of fixed length) to be presented as strings of fixed length. Next observe that combining the Cook-reduction of $\mathcal{PC}$ to $\mathcal{NP}$ with the hypothesis $\mathcal{NP} \subseteq \mathcal{P}/\mathrm{poly}$, implies that $\mathcal{PC}$ is Cook-reducible to $\mathcal{P}/\mathrm{poly}$. In particular, this implies that any search problem in $\mathcal{PC}$ can be solved by a family of polynomial-size circuits. Note that the resulting circuit that solves n-bit long instances of such a problem may incorporate polynomially (in n) many circuits, each solving a decision problem for m-bit long instances, where $m \in [\mathrm{poly}(n)]$. Needless to say, the size of the resulting circuit that solves the search problem of the aforementioned $R' \in \mathcal{PC}$ (for instances of length n) is upper-bounded by $\mathrm{poly}(n) \cdot \sum_{m=1}^{\mathrm{poly}(n)} \mathrm{poly}(m)$.

It follows that $x \in S$ if and only if *there exists a* $\mathrm{poly}(|x| + p(|x|))$*-size circuit* C' *such that for all* $y \in \{0,1\}^{p(|x|)}$ *it holds that* $(\langle x,y\rangle, C'(x,y)) \in R'$. Note that in the case that $x \in S$ we use the circuit C' that is guaranteed for inputs of length $|x| + p(|x|)$ by the foregoing discussion[7], whereas in the case that $x \notin S$ it does not matter which circuit C' is used (because in that case there exists a y such that for all z it holds that $(\langle x,y\rangle, z)) \notin R'$).

The key observation regarding the foregoing condition (i.e., $\exists C' \forall y\ (\langle x,y\rangle, C'(x,y)) \in R'$) is that it is of the desired form (of a Σ_2 statement). Specifically, consider

[7] Thus, C' may actually depend only on $|x|$, which in turn determines $p(|x|)$.

the polynomial-time verification procedure V that given x, y and the description of the circuit C', first computes $z \leftarrow C'(x,y)$ and accepts if and only if $(\langle x,y\rangle, z) \in R'$, where the latter condition can be verified in polynomial-time (because $R' \in \mathcal{PC}$). Denoting the description of a potential circuit by $\langle C'\rangle$, the aforementioned (polynomial-time) computation of V is denoted $V(x,\langle C'\rangle,y)$, and indeed $x \in S$ if and only if

$$\exists \langle C'\rangle \in \{0,1\}^{\mathrm{poly}(|x|+p(|x|))} \forall y \in \{0,1\}^{p(|x|)}\ V(x,\langle C'\rangle,y) = 1.$$

Having established that $S \in \Sigma_2$ for an arbitrary $S \in \Pi_2$, we conclude that $\Pi_2 \subseteq \Sigma_2$. The theorem follows (by applying Exercise 3.9.4). ■

Chapter Notes

The class $\mathcal{P}/\mathrm{poly}$ was defined by Karp and Lipton [132] as part of a general formulation of "machines which take advice" [132]. They also noted the equivalence to the traditional formulation of polynomial-size circuits as well as the effect of uniformity (Proposition 3.4).

The Polynomial-Time Hierarchy ($\mathcal{PH}$) was introduced by Stockmeyer [205]. A third equivalent formulation of $\mathcal{PH}$ (via so-called "alternating machines") can be found in [49].

The implication of the failure of the conjecture that $\mathcal{NP}$ is not contained in $\mathcal{P}/\mathrm{poly}$ on the Polynomial-time Hierarchy (i.e., Theorem 3.12) was discovered by Karp and Lipton [132]. This interesting connection between non-uniform and uniform complexity provides the main motivation for presenting $\mathcal{P}/\mathrm{poly}$ and $\mathcal{PH}$ in the same chapter.

Exercises

Exercise 3.1 (a small variation on the definitions of $\mathcal{P}/\mathrm{poly}$) Using an adequate encoding of strings of length smaller than n as n-bit strings (e.g., $x \in \cup_{i<n}\{0,1\}^i$ is encoded as $x01^{n-|x|-1}$), define circuits (resp., machines that take advice) as devices that can handle inputs of various lengths up to a given bound (rather than as devices that can handle inputs of a fixed length). Show that the class $\mathcal{P}/\mathrm{poly}$ remains invariant under this change (and Theorem 3.6 remains valid).

Exercise 3.2 (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 $\mathcal{P}/\mathrm{poly}$. Note that a sparse set may be undecidable.
2. Prove that a set is in $\mathcal{P}/\mathrm{poly}$ 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 $\text{poly}(|x|)$, and so emulating the reduction on an input of length n only requires knowledge of all the strings that are in the sparse set and have length at most $\text{poly}(n)$.

Exercise 3.3 (advice 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 $\mathcal{P}/(\ell + \delta)$.

Guideline: For every sequence $\overline{a} = (a_n)_{n \in \mathbb{N}}$ such that $|a_n| = \ell(n) + \delta(n)$, consider the set $S_{\overline{a}}$ that encodes $\overline{a}$ such that $x \in S_{\overline{a}} \cap \{0,1\}^n$ if and only if the $\text{idx}(x)^{\text{th}}$ bit in a_n equals 1 (and $\text{idx}(x) \leq |a_n|$), where $\text{idx}(x)$ denotes the index of x in $\{0,1\}^n$. For more details see Section 4.1.

Exercise 3.4 Prove that Σ_2 contains all sets that are Cook-reducible to $\mathcal{NP}$.

Guideline: This is quite obvious when using the definition of Σ_2 as presented in Section 3.2.2; see Exercise 3.9. Alternatively, the fact can be proved by using *some* of the ideas that underlie the proof of Theorem 2.33, while noting that a conjunction of NP and coNP assertions forms an assertion of type Σ_2 (see also the second part of the proof of Proposition 3.11).

Exercise 3.5 Let $\Delta = \mathcal{NP} \cap \text{co}\mathcal{NP}$. Prove that Δ equals the class of decision problems that are Cook-reducible to Δ (i.e., $\Delta = \mathcal{P}^{\Delta}$).

Guideline: See proof of Theorem 2.33.

Exercise 3.6 (the class Π_i) Recall that Π_k is defined to equal $\text{co}\Sigma_k$, which in turn is defined to equal $\{\{0,1\}^* \setminus S : S \in \Sigma_k\}$. 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

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

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

Exercise 3.7 (complete problems for the various levels of $\mathcal{PH}$) 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 under Karp-reductions. That is, denoting the aforementioned problem by `kQBF`, prove that `kQBF` is in Σ_k and that every problem in Σ_k is Karp-reducible to `kQBF`.

Exercise 3.8 (on the relation between $\mathcal{PH}$ and $\mathcal{AC}^0$) Note that there is an obvious analogy between $\mathcal{PH}$ and constant-depth polynomial-size circuits of unbounded fan-in, where existential (resp., universal) quantifiers are represented by "large" $\bigvee$ (resp., $\bigwedge$) 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** $Z \in \{0,1\}^N$ is a circuit c_Z of size at most $p(\log_2 N)$ such that for every $i \in [N]$ it holds that $c_Z(i)$ equals the i^{th} bit of Z.

- 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 $Z \in \{0,1\}^N$, determine whether or not $C_N(Z) = 1$.*

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 $Z \in \{0,1\}^N$ such that $x \in S$ if and only if $C_N(Z) = 1$. (Note that Z is represented by a circuit c_Z of size at most $p(\log_2 N)$, and that it follows that $|c_Z| \leq \text{poly}(n)$ and thus $N \leq \exp(\text{poly}(n))$.)[8]

Guideline: Let $S \in \Sigma_k$ and let V be the corresponding verification algorithm as in Definition 3.8. 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 fixed 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 C_N at an input consisting of the truth-table of $V(x, \cdots)$ (i.e., when setting $Z_{i_1, i_2, ..., i_k} = V(x, i_1, ..., i_k)$, where $[2^m] \equiv \{0,1\}^m$). Note that the size of C_N is $O(N)$.[9]

Exercise 3.9 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, $\mathcal{P}^{\mathcal{P}} = \mathcal{P}$.)

[8] 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).

[9] **Advanced comment:** the computational limitations of $\mathcal{AC}^0$ circuits (see, e.g., [79, 111]) imply limitations on the functions of a *generic* input Z that the aforementioned circuits C_N can compute. Actually, these limitations apply also to $Z = h(Z')$, where $Z' \in \{0,1\}^{N^{\Omega(1)}}$ is generic and each bit of Z equals either some fixed bit in Z' or its negation. Unfortunately, these computational limitations do not seem to provide useful information on the limitations of functions of inputs Z that have succinct representation (as obtained by setting $Z_{i_1, i_2, ..., i_k} = V(x, i_1, ..., i_k)$, where V is a fixed polynomial-time algorithm and only $x \in \{0,1\}^{\text{poly}(\log N)}$ varies). 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. [79].

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 $\mathcal{P}^{\mathcal{C}} = \mathcal{P}^{\mathrm{co}\mathcal{C}}$ and $\mathcal{P}^{\mathcal{C}} = \mathrm{co}\mathcal{P}^{\mathcal{C}}$.)
3. For every $k \geq 1$, it holds that $\Sigma_k \subseteq \Pi_{k+1}$ and $\Pi_k \subseteq \Sigma_{k+1}$. Thus, $\mathcal{PH} = \cup_k \Pi_k$.
4. For every $k \geq 1$, if $\Sigma_k \subseteq \Pi_k$ (resp., $\Pi_k \subseteq \Sigma_k$) then $\Sigma_k = \Pi_k$.

   (Hint: For any $S \in \Pi_k$ (resp., $S \in \Sigma_k$), apply the hypothesis to $\{0,1\}^* \setminus S$.)

Exercise 3.10 In continuation to Exercise 3.7, prove that following claims:

1. SAT is computationally equivalent to 1QBF.
2. For every $k \geq 1$, it holds that $\mathcal{P}^{\Sigma_k} = \mathcal{P}^{\mathtt{kQBF}}$ and $\Sigma_{k+1} = \mathcal{NP}^{\mathtt{kQBF}}$.

   Guideline: Prove that if S is $\mathcal{C}$-complete then $\mathcal{P}^{\mathcal{C}} = \mathcal{P}^S$. Note that $\mathcal{P}^{\mathcal{C}} \subseteq \mathcal{P}^S$ uses the polynomial-time reductions of $\mathcal{C}$ to S, whereas $\mathcal{P}^S \subseteq \mathcal{P}^{\mathcal{C}}$ uses $S \in \mathcal{C}$.

Exercise 3.11 (an alternative proof of Theorem 3.12) In continuation to the discussion in the proof of Theorem 3.12, use the following guidelines to provide an alternative proof of Theorem 3.12.

1. First, prove that if S' is downwards self-reducible (as defined in Exercise 2.13) then the correctness of circuits deciding S' can be decided in $\mathrm{co}\mathcal{NP}$. Specifically, denoting by χ the characteristic function of S', show that the set

   $$\mathrm{ckt}_\chi \stackrel{\mathrm{def}}{=} \{(1^n, \langle C \rangle) : \forall w \in \{0,1\}^n \; C(w) = \chi(w)\}$$

   is in $\mathrm{co}\mathcal{NP}$.

   Guideline: Using the more flexible formulation suggested in Exercise 3.1, it suffices to verify that, for every $i < n$ and every i-bit string w, the value $C(w)$ equals the output of the downwards self-reduction on input w when obtaining answers according to C. Thus, for every $i < n$, the correctness of C on inputs of length i follows from its correctness on inputs of length less than i. Needless to say, the correctness of C on the empty string (or on all inputs of some constant length) can be verified by comparison to the fixed value of χ on the empty string (resp., the values of χ on a constant number of strings).
2. Recalling that SAT is downwards self-reducible and that $\mathcal{NP}$ reduces to SAT, derive Theorem 3.12 as a corollary of Part 1.

Exercise 3.12 In continuation to Part 2 of Exercise 3.2, 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 [77]). Here, we only consider the special case in which the sparse set is contained in a polynomial-time decidable set that is itself sparse (e.g., the latter set may be $\{1\}^*$, in which case the former set may be an arbitrary unary set). Actually, prove the following seemingly stronger claim:

> *if* SAT *is Karp-reducible to a set* $S \subseteq G$ *such that* $G \in \mathcal{P}$ *and* $G \setminus S$ *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 the task of 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 demonstrated to contain no satisfying assignment (at the leaves).[10]

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 Karp-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*. However, once we backtrack from this internal node, we know that the corresponding element of G is not in S, and we will never scan again a sub-tree rooted at a node that is mapped to this element. 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 $n \cdot |\cup_{i=1}^{\mathrm{poly}(|\phi|)} \{0,1\}^i \cap (G \setminus S)|$.)

[10] 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 $\tau 1$.