Complexity Zoology

By

ROBERT JOSEPH SANDERS, JR.

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

MATHEMATICS

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Greg Kuperberg, Chair

Eric Babson

Bruno Nachtergaele

Committee in Charge

2019

Contents

Complexity Zoology

Abstract

Complexity Zoology is a program that deduces complexity class inclusions and oracle separations from an initial data set. It is designed to aid the process of compiling information about complexity classes and the relationships between them. This thesis is a description of Complexity Zoology and how it works, along with a survey of complexity theory conducted with the assistance of the Complexity Zoology software. We begin with an overview of commonly used computational models and of the theory of complexity class operators. Then, the algorithm underlying Complexity Zoology is explained. Finally, we survey the landscape of complexity theory through the lens of Complexity Zoology's data set. We include both a detailed survey involving a smaller number of classes and a bird's-eye view involving the full set of classes in Zoology's input file.

Acknowledgments

Thanks are owed to several for making both Complexity Zoology and this thesis possible. Much of this project's data set is based on email correspondence between Greg Kuperberg, Scott Aaronson, and Lance Fortnow. Other contributors to these conversations include Lijie Chen, Avishay Tal, Justin Thaler, Prashant Vasudevan, John Watrous, and Jiapeng Zhang.

I also thank Greg Kuperberg for his extensive help with many aspects of the project: adding to the data set, clarifying key concepts and arguments, facilitating conversations with experts in complexity theory, and writing the original version of Complexity Zoology that served as the outline for this new version. Finally, I thank my thesis committee—Eric Babson, Greg Kuperberg, and Bruno Nachtergaele—for their time and attention.

CHAPTER 1

Introduction

This document is a description of a computer program called *Complexity Zoology*. The program is an *expert system*: it is equipped with a database of information about which complexity classes are subsets of other complexity classes along with an inference engine that the program uses to deduce new conclusions from the existing information. Complexity Zoology then outputs a diagram of the relationships between the complexity classes in the input file.

Complexity Zoology takes its name from the Complexity Zoo, an online wiki of information about complexity classes written and maintained by Scott Aaronson [**Aar19**]. This version of Complexity Zoology was written from scratch; however, some of the design choices—particularly the input syntax and the functionality of the output diagram—are based on an earlier version by Greg Kuperberg.

The purpose and motivation of Complexity Zoology is to be a partially automated survey of complexity theory. The scope of a field as well-developed as complexity theory is so large that it can be difficult to quickly determine the status of a proposition of the form "the complexity class C_1 is contained in the complexity class C_2." Complexity Zoology aims to help with this problem, at least for the most fundamental complexity classes. More specifically, the project aims to:

- summarize a large portion of the known complexity class inclusions and oracle separations;
- identify redundant results (i.e., the results that follow logically from other known results);
- answer questions about complexity class relations automatically when the answer is a corollary of established results.

Complexity Zoology has proved itself useful in identifying when a result is a corollary of other results and in falsifying conjectures, but the demonstrated strength of the program has been the identification of stimulating open problems. Among the questions that the project has raised are these:

(1) We know that $\mathsf{NP} \subseteq \mathsf{IP} \subseteq \mathsf{MIP}$, and these classes are believed to be distinct. Is there an oracle relative to which $\mathsf{MIP} \not\subseteq \mathsf{IP}$? Does $\mathsf{IP} = \mathsf{NP}$ relative to the random oracle?

(2) Is there an oracle relative to which $\mathsf{NISZK} \not\subseteq \mathsf{coNIQSZK}$?

(3) Is there an oracle relative to which $\mathsf{SZK} \not\subseteq S_2\mathsf{P}$?

(4) Is there an oracle relative to which $\mathsf{BQP} \not\subseteq \mathsf{IP}$?

To make Complexity Zoology an effective tool, it has been necessary to limit its scope in a few ways. First, the system's expertise does not lie in reasoning about complexity theory as such. It knows nothing of the standard techniques used to prove results in complexity theory, such as diagonalization. It does not even understand what complexity classes are: common classes such as P, NP, and BPP are understood only in terms of their relationships to other complexity classes. Instead, the strength of Complexity Zoology consists of understanding results and open problems in the field. For example, if it is known that $\mathsf{C}_1 \subseteq \mathsf{C}_2$ is proven and $\mathsf{C}_1 \subseteq \mathsf{C}_3$ is an open question, then Complexity Zoology can conclude that $\mathsf{C}_2 \subseteq \mathsf{C}_3$ is unproven – either it has been disproven, or it is itself open. In essence, the system can be thought of as a diligent student conducting a broad overview of the field, drawing connections between results and attempting to fill in all possible gaps without examining the details.

Second, the project has been deliberately limited to a core collection of important complexity classes. The program has the capacity to identify critical gaps in its knowledge, and by choosing a conservative list of classes we increase the chance that the questions Complexity Zoology asks are of theoretical interest. The system's input syntax makes it easy to add and remove classes as needed, so adjustments can be made as necessary.

CHAPTER 2

Complexity Classes and Operators

This chapter summarizes the foundational concepts that underlie classical and quantum complexity theory. The general reference used for these ideas is the text *Computational Complexity* by Arora and Barak [**AB09**]. Throughout this thesis, we also use a survey of Watrous as a secondary reference for quantum computation [**Wat09**].

2.1. Models of Computation

2.1.1. Complexity Classes and Decision Problems. A computational problem is a question about a particular object of input, such as "Is the input x a prime number?" or "What is the greatest common divisor of the integers x and y?" It is assumed that the input to the question, as well as the answer, can be encoded as a finite string of zeros and ones—i.e., as an element of Σ^*. (We denote by Σ the set $\{0,1\}$ and by S^* the set of all finite strings whose characters consist of elements of the set S.) Thus, a computational problem can be modeled as a function $f : \Sigma^* \to \Sigma^*$.

In general, a complexity class is a set of computational problems $f : \Sigma^* \to \Sigma^*$, usually interpreted as the set of all problems that are tractable within a specified computational model. For this project, we are interested in *decision problems*, which are computational problems whose answer is either yes or no (encoded as 1 and 0, respectively). Within the functional framework, decision problems are formalized as functions $d : \Sigma^* \to \Sigma$. Such functions can be identified with the set $\{x \in \Sigma^* : d(x) = 1\}$. For this reason, we will identify decision problems with *languages*, or subsets $\mathscr{L}$ of Σ^*. We will also conflate a language with its corresponding decision function when it is convenient to do so.

2.1.2. Notation and Conventions. At this point, it will be useful to identify some conventions. The set $\mathbb{N}$ of natural numbers is assumed to contain zero. We use big-O notation to describe the size of functions: for a pair of functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$, we write $f = O(g)$ (or sometimes $f(n) = O(g(n))$) if there exists a real constant C such that $f(n) \leq Cg(n)$ for every $n \in \mathbb{N}$. We also write $f(n) = O(n^*)$ to indicate that there exists $k \in \mathbb{N}$ such that $f(n) = O(n^k)$. Exponents are used to write strings having the same character repeated multiple times, so that $0^4 = 0000$, for example.

2.1.3. Classical Computation. The most commonly used model of computation in classical complexity theory is the *Turing machine*. A (deterministic) *Turing machine* with $k \geq 2$ tapes, abbreviated TM, is a triple $M = (\Gamma, Q, \delta)$ containing the following data:

(i) A set Γ of *symbols*, called the *alphabet*, which must include the blank symbol $\Box$, the start symbol $\triangleright$, and the numerals 0 and 1;
(ii) A set Q of *states* of M, one of which is the starting state $q_{\texttt{start}}$, and another of which is the halting state $q_{\texttt{halt}}$;
(iii) A *transition function* $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathsf{L}, \mathsf{S}, \mathsf{R}\}^k$ satisfying

$$\delta(q_{\texttt{halt}}, s_1, \ldots, s_k) = (q_{\texttt{halt}}, s_2, \ldots, s_k, \mathsf{S}, \ldots, \mathsf{S})$$

for every $s_i \in \Gamma$.

A *configuration* for a Turing machine $M = (\Gamma, Q, \delta)$ with k tapes is a tuple $c = (q, i_1, \ldots, i_k, t)$, where $q \in Q$, $i_1, \ldots, i_k$ are natural numbers, and $t : \{1, \ldots, k\} \times \mathbb{N} \to \Gamma$ is a *tape function* such that $t(j, \ell) = \Box$ for all sufficiently large ℓ.

An *initial configuration* is a configuration $c = (q_{\texttt{start}}, 0, \ldots, 0, t)$ satisfying the following conditions:

(i) $t(j, 0) = \triangleright$ for $j \in \{1, \ldots, k\}$;
(ii) $t(j, \ell) = \Box$ for $j > 1$ and $\ell \in \mathbb{N}$;
(iii) There exists a natural number $N \geq 1$ such that $t(1, \ell) \neq \Box$ for all $\ell < N$ and $t(1, \ell) = \Box$ for all $\ell \geq N$.

The string $t(1,1) \ldots t(1, N-1)$ is the *input*. If $N = 1$, the input is considered to be the empty string ε.

A *halted configuration* is a configuration with $q = q_{\texttt{halt}}$. If there exists a natural number N such that $t(k, \ell) \neq \Box$ for $\ell \leq N$ and $t(k, \ell) = \Box$ for $\ell > N$, then the string $t(k, 0) \ldots t(k, N)$ is the *output*; otherwise, ε is the output.

Let $c = (q, i_1, \ldots, i_k, t)$ and $c' = (q', i'_1, \ldots, i'_k, t')$ be configurations for $M = (\Gamma, Q, \delta)$. The configuration c' *succeeds* c ($c \to c'$) if

$$\delta(q, t(1, i_1), \ldots, t(k, i_k)) = (q', t'(2, i_2), \ldots, t'(k, i_k), \mathsf{X}_1, \ldots, \mathsf{X}_k),$$

where $t(j,\ell) = t'(j,\ell)$ if $j = 1$ or $\ell \notin \{i_2, \ldots, i_k\}$, and

$$i'_j = \max\{0, i_j - 1\} \text{ if } \mathsf{X}_j = \mathsf{L},$$

$$i'_j = i_j \text{ if } \mathsf{X}_j = \mathsf{S},$$

$$i'_j = i_j + 1 \text{ if } \mathsf{X}_j = \mathsf{R}.$$

A *computation* for a machine M is a sequence $c_0 \to c_1 \to \ldots \to c_n$, where c_0 is an initial configuration and c_n is a halted configuration. The input of c_0 and the output of c_n are the *input* and *output* of the computation, respectively. The machine M *computes* the function $f : \Sigma^* \to \Sigma$ if the computation for M with input x has output $f(x)$. We also write $M(x)$ to indicate the output of M with input x. If $M(x) = 1$, the machine *accepts* the input; if $M(x) = 0$, the machine *rejects* the input.

The *computation time* $T_M(x)$ of the machine M with input x is the length of the computation of M with input x. For a function $f : \mathbb{N} \to \mathbb{N}$, *$M$ runs in $f(n)$-time* if there exists a constant C such that $T_M(x) \leq C \cdot f(|x|)$ for every $x \in \Sigma^*$. M runs in *polynomial time* if it runs in $f(n)$-time for some $f(n) = O(n^*)$. The function class of all $f : \Sigma^* \to \Sigma^*$ that are polynomial-time computable is given by FP.

There are several variations of the Turing machine concept. In a *non-deterministic* Turing machine, there are two transition functions, and a computation is a tree of configurations rather than a sequence. A probabilistic Turing machine also has two transition functions, but at each stage of the computation the machine tosses a coin to decide which transition function it uses. These variations are often useful, but it is usually enough to use a standard, deterministic Turing machine along with an additional string $y \in \Sigma^*$ placed alongside the input. y can then represent either a sequence of random bits/coin tosses or a particular branch of a non-deterministic computation.

All Turing machines are assumed to have an alphabet consisting entirely of $\Sigma = \{0, 1\}$. Any additional symbols are assumed to be encoded in some way. For example, if the input is a tuple, such as $\langle x, y \rangle$ for $x, y \in \Sigma^*$, then we could encode the symbol 0 as 00, 1 as 01, and the separating comma as 10. We will also occasionally want to consider a natural number as being the input or output of a computational process. We therefore fix a bijection $e : \mathbb{N} \to \Sigma^*$ allowing us to identify natural numbers with the strings in Σ^*. For definiteness, we can take $e(n)$ to be the result of removing the initial 1 from the binary representation of $n + 1$.

Another classical model of computation is the *Boolean circuit*. A Boolean circuit is an acyclic graph with one or more sources (vertices with no incoming edges) and exactly one sink (a vertex with no outgoing edges). Each vertex that is not a source is labeled with $\wedge$, $\vee$, or $\neg$ and represent the *gates* of the circuit. Vertices labeled with $\neg$ must have a fanin—i.e., number of incoming edges—of 1. Each source is labeled with a unique element of $\{1,\ldots,n\}$, where n is the number of sources.

For a Boolean circuit C, a *value function* $v : V \to \Sigma$ is a function on the set V of vertices of C satisfying the following properties:

- If the vertex x is labeled with $\wedge$, then $v(x) = 1$ if and only if $v(y) = 1$ for every predecessor y of x.
- If the vertex x is labeled with $\vee$, then $v(x) = 1$ if and only if $v(y) = 1$ for some predecessor y of x.
- If the vertex x is labeled with $\neg$, then $v(x) = 1$ if and only if $v(y) = 0$, where y is the unique predecessor of x.

For each assignment of 0 or 1 to a source vertex, there is a unique value function that matches the assignment. (Since C is finite and acyclic, if this were not the case there would be a minimal vertex for which a unique value of v is not determined, but such a vertex is either a source or determined by its predecessors by minimality.)

Thus, given a Boolean circuit C, we can define a function $\bar{C} : \Sigma^n \to \Sigma$ according to the following procedure for $x \in \Sigma^n$, let $v : V \to \Sigma$ be the value function that assigns the kth bit of x to the source with the label k; then, set $\bar{C}(x) = v(y)$, where y is the sink of C. In this way, the circuit C can be thought of as computing the function $\bar{C}$.

Unlike Turing machines, Boolean circuits require the input to be of fixed length. A natural solution to this problem is to consider a family $\{C_n : n \geq 1\}$ of circuits, which is then considered to compute the function $\bar{C} : \Sigma^* \to \Sigma$ defined by $\bar{C}(x) = \bar{C}_n(x)$ for all $x \in \Sigma^*$, where n is the length of x. However, this model is actually more powerful than a Turing machine, because it can compute any unary language (languages consisting entirely of strings of 1s). When the size of C_n is limited to a polynomial of n, the resulting complexity class is the uncountable class P/poly rather than the usual polynomial-time computable class P. Intuitively, since there is a separate circuit for each possible input length n, it is possible to encode an amount of advice into each circuit that is a polynomial of n in length.

2.1.4. Quantum Computation. It remains to define a computational model for quantum computation. Rather than using classical bits, which must exist in a state of 0 or 1, quantum computation uses quantum bits, or *qubits*. A qubit is a normalized element $a|0\rangle + b|1\rangle$ of $\mathbb{C}^2$, where $a,b \in \mathbb{C}$ and $|0\rangle$ and $|1\rangle$ are orthonormal with respect to the usual inner product on $\mathbb{C}^2$ (so that $|a|^2 + |b|^2 = 1$). There are several possible models by which the qubits can be employed for computation; for definiteness, the quantum circuit will be our standard choice. Just as the 0-1 values are changed as they move through a classical circuit, the qubits of a quantum circuit are transformed unitarily as they pass through the vertices of a circuit.

DEFINITION 2.1.1. A *quantum circuit* is a finite acyclic graph satisfying the following properties:

- There are n sources, each of which is labeled with an element of the set $\{1,\ldots,n\}$.
- Each source has a fanout of 1, and each sink has a fanin of 1.
- If a vertex is neither a sink nor a source, then its fanin must equal its fanout, which can be at most 3.
- Each vertex x that is neither a sink nor a source is labeled with a 2^{k_x}-dimensional unitary transformation, where k_x is the fanin of x. The vertex is also labeled with a bijective function $f_x : E_i^x \to E_o^x$, where E_i^x is the set of incoming edges for x and E_o^x is the set of outgoing edges for x.

Note that, from each source, there is a canonical path to follow:

- From the source, follow the only edge forward.
- When arriving at the vertex x from the edge e, follow the edge $f_x(e)$ forward.
- The path ends when a sink is reached.

Each such path terminates at a different sink, and every sink is the terminal vertex of some such path. Thus, there is a natural bijection between sources and sinks, and we can assign the corresponding element of $\{1,\ldots,n\}$ to each of the sinks.

As with classical circuits, the input to a quantum circuit is a string of bits—in this case, a direct product of qubits of the form $|j\rangle$, $j = 0,1$ (we denote $|j_1\rangle \otimes \ldots \otimes |j_n\rangle = |j_1\rangle \ldots |j_n\rangle = |j_1 \ldots j_n\rangle$. The qubits then travel along the circuit according to the canonical paths, being transformed by the unitary maps at each vertex. Once each qubit reaches its sink, the initial bit string will have been transformed into a new state $\sum \alpha_{j_1 j_2 \cdots j_n} |j_1 j_2 \ldots j_n\rangle$. For a circuit C and initial state φ, we denote the final state by $\bar{C}(|\varphi\rangle)$.

To be explicit, this is the procedure by which the initial state $|\varphi\rangle$ is transformed into $\bar{C}(|\varphi\rangle)$:

(1) If necessary, add vertices so that all canonical paths have the same length and each vertex occurs at the same length along any canonical path passing through it. We do this by introducing new vertices with fanin 1 and labeling new non-terminal vertices with the identity transformation. Denote the new length of each canonical path by N.

(2) For $1 \leq k \leq N$, define a unitary transformation U_k:

   (a) Reorder the set $\{1, \ldots, n\}$ so that canonical paths passing through the same kth vertex are assigned adjacent numbers. Denote the resulting permutation of qubits by W.

   (b) Number the transformations labeling each kth vertex so that they appear in the same order indicated by the permutation W: $V_1, V_2, \ldots, V_{r_k}$.

   (c) Set $U_k = W^*(V_1 \otimes V_2 \otimes \ldots \otimes V_{r_k})W$.

(3) Set $\bar{C}(|\varphi\rangle) = U_N \ldots U_2 U_1 |\varphi\rangle$.

Finally, we obtain the result of the computation from $\bar{C}(|\varphi\rangle)$ by *measuring* it. If $\bar{C}(|\varphi\rangle) = \Sigma_x \alpha_x |x\rangle$, where the sum is over all elements of Σ^n, then the result of the measurements is $|x\rangle$ with probability $|\alpha_x|^2$.

To make meaningful discussion of quantum complexity theory possible, it is necessary to restrict the permitted operations to a finite set of *universal operations*. This means that any unitary matrix (of dimension ≥ 3) can be approximated to arbitrary precision by the finite collection of operators, in the same way that $\wedge$, $\vee$, and $\neg$ gates suffice for the purposes of classical computation. Moreover, this approximation can be accomplished efficiently:

THEOREM 2.1.1 (Solovay-Kitaev, [**Kit97**]). *There exists a finite set F of unitary operators, each having dimension ≤ 3, such that F is an efficient universal gate set, in the following sense:*

Let $d \geq 3$ be an integer, and let $\varepsilon > 0$. There exists a positive integer $\ell \leq 100(d \log 1/\varepsilon)^3$ such that for every $d \times d$ unitary matrix $U = (U_{jk})$, there exist unitary matrices $U_1, \ldots, U_\ell$ such that for each $j, k \in \{1, \ldots, d\}$,

$$|U_{jk} - (U_\ell \ldots U_1)_{jk}| < \varepsilon,$$

where each U_j corresponds to applying an operation from F to at most 3 of d qubits.

One possible choice of universal operators is the Hadamard gate $H = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right)$, the Toffoli gate $I_6 \oplus \left(\begin{smallmatrix}0 & 1\\ 1 & 0\end{smallmatrix}\right)$ (where I_6 denotes the 6×6 identity matrix), and the phase shift gate $\left(\begin{smallmatrix}1 & 0\\ 0 & i\end{smallmatrix}\right)$.

DEFINITION 2.1.2. Let $\{C_n : n \geq 1\}$ be a family of quantum circuits such that C_n has $m_n \geq n$ sources. $\{C_n : n \geq 1\}$ *computes* $f : \Sigma^* \to \Sigma$ *with probability* p if for every $x \in \Sigma^*$, when $\bar{C}_n(|x0^{m_n-n}\rangle)$ is measured, the first bit is equal to $f(x)$ with probability $\geq p$.

As in the classical case, to define the class of problems that are quantum computable in polynomial time, it is not only necessary to restrict the size of C_n to a polynomial of n, but also to require that it is possible to classically compute a description of C_n from an input of n in polynomial time. Without this restriction, we would admit as computable many languages that are not generally thought of as computable. This restriction of circuits also works in the classical case, reducing the resulting complexity class from P/poly to P.

Many complexity classes of interest are derived from more traditional complexity classes by replacing the classical method of computation with the quantum one. Doing so to the class P results in the class BQP, doing so to the class MA results in the class QMA, and so on. Using this observation as a basis allows us to imagine a non-rigorous "pseudo-operator" $\mathsf{C} \mapsto \mathcal{Q} \cdot \mathsf{C}$ that transforms a classical complexity class C into its quantum counterpart $\mathcal{Q} \cdot \mathsf{C}$. Since operators must be able to act on complexity classes independently of the underlying means of computation, we cannot make $\mathcal{Q}$ into a rigorously defined operator. Nevertheless, it is intuitively useful to think of $\mathcal{Q}$ as an operator and note one property in particular: $\mathsf{C} \subseteq \mathcal{Q} \cdot \mathsf{C}$ for every C. In other words, any computations that are possible classically must also be possible in the quantum world. This is not immediately obvious from our models of computation, because, for example, classical circuits allow for non-reversible operations (e.g. one cannot determine from the value of an $\wedge$-vertex the value of its predecessors) while quantum operations are unitary and therefore reversible by necessity. However, any classical operations have reversible quantum versions, generally implemented by means of additional "scratch qubits" that are assumed to be zero at the outset. For instance to implement the $\wedge$-operator, we apply the unitary mapping

$$|xy\rangle|z\rangle \mapsto |xy\rangle|z +_2 xy\rangle,$$

where $+_2$ indicates addition modulo 2. As long as the scratch qubit $|z\rangle$ is initialized to 0, the bit $|z +_2 xy\rangle$ will be equal to the classical $\wedge$-operator on x and y. The need for extra scratch bits is the reason we allow trailing zeros in the input to a quantum circuit.

2.1.5. Universal Turing Machines. Turing machines should likewise be encoded as elements of Σ^*. We fix a surjective mapping $\alpha \mapsto M_\alpha$ from Σ^* to the set of Turing machines. For convenience, this mapping should have the property that for every Turing machine M there exist infinitely many α such that $M = M_\alpha$. The encoding is chosen so that the following theorem is true:

THEOREM 2.1.2. *There exists a Turing machine U such that $U(x,\alpha) = M_\alpha(x)$ for all $x,\alpha \in \Sigma^*$ with the property that if $T(x,\alpha)$ is the computation time for $M_\alpha(x)$, then the computation time for $U(x,\alpha)$ is $CT(x,\alpha)\log T(x,\alpha)$, where C depends only on the number of tapes and states of M_α.*

Functions are generally restricted to those that are *time-constructible*, meaning that the function $T : \mathbb{N} \to \mathbb{N}$ can be computed by a Turing machine in time $O(T(n))$ and $T(n) \geq n$ for all $n \in \mathbb{N}$.

2.2. Operators and Relativization

Each class in Complexity Zoology's data set can be *relativized*. Informally, relativization is the process of taking a particular computational problem $f : \Sigma^* \to \Sigma^*$ as a black box that can be given any input, and the answer is instantaneously given. The black box is called the *oracle* and the process of giving an input to the black box is referred to as *querying* the oracle. For a complexity class C, we denote by C^f the complexity class with the same computational model as C, but for which queries to an oracle f are allowed. At a minimum, this means that if $f : \Sigma^* \to \Sigma$ is a decision function, then the associated language $\mathscr{L}$ lies in C^f. Moreover, we expect that $\mathsf{C} \subseteq \mathsf{C}^f$, since the oracle can simply be ignored during any computation. For a pair of complexity classes $\mathsf{C}_1, \mathsf{C}_2$, we also set $\mathsf{C}_1^{\mathsf{C}_2} = \bigcup_{f \in \mathsf{C}_2} \mathsf{C}_1^f$.

Each of the computational models we consider is powerful enough to encode finite changes to an oracle. For this reason, if f and g are oracles that differ only on a finite subset of Σ^*, then $\mathsf{C}^f = \mathsf{C}^g$. This fact is crucial to the concept of a *random oracle*.

LEMMA 2.2.1 (**[For18]**). *Let $r : \Sigma^* \to \Sigma$ be chosen uniformly at random, so that* $\Pr[r(x) = 1] = 1/2$ *for each $x \in \Sigma^*$. If $\mathscr{F}$ is any set of functions $\Sigma^* \to \Sigma$ that is closed under finite differences between functions, then* $\Pr[r \in \mathscr{F}] = 0$ *or* 1.

PROOF. The *Kolmogorov zero-one law* states that if $S = \{X_j : j \in \mathbb{N}\}$ is a countable set of independent random variables and E is an event that is independent of each finite subset of S, then $\Pr[E] = 0$ or 1. For our purposes, $X_j = r(j)$ and E is the event $r \in \mathscr{F}$. This event is independent of finitely many choices of values $r(j)$, because the closure property of $\mathscr{F}$ implies that changing finitely many $r(j)$ does not affect whether $r \in \mathscr{F}$. Thus, the zero-one law proves the lemma. $\square$

By the previous remark, the set

$$\mathscr{F} = \{f : f \text{ is a function } \Sigma^* \to \Sigma \text{ such that } \mathsf{C}_1^f \subseteq \mathsf{C}_1^f\}$$

for a pair of complexity classes C_1 and C_2 is closed under finite differences. We therefore have the following result:

THEOREM 2.2.1. *If $r : \Sigma^* \to \Sigma$ is chosen uniformly at random, then $\mathsf{C}_1^r \subseteq \mathsf{C}_2^r$ with probability* 0 *or* 1.

As a consequence of this theorem, we can consider whether $\mathsf{C}_1^r \subseteq \mathsf{C}_2^r$ with respect to "the" random oracle r.

Contrary to what the notation would suggest, $\mathsf{C} \mapsto \mathsf{C}^f$ is not a map on the set of complexity classes. For example, there is an oracle f such that $\mathsf{P}^f \neq \mathsf{NP}^f$, but if $\mathsf{C} \mapsto \mathsf{C}^f$ were a function on the set of complexity classes, this would imply $\mathsf{P} \neq \mathsf{NP}$, which is an open problem. Instead, the action $\mathsf{C} \mapsto \mathsf{C}^f$ transforms the computational model itself. In the case of complexity classes involving Turing machines, the Turing machines are replaced with *oracle Turing machines* capable of querying an oracle.

Unfortunately, there is no uniform way to define C^f from C that works for every complexity class in the data set. Thus, we formally consider a complexity class to be a family of sets $\mathsf{C} = \{\mathsf{C}^f : f \text{ is a function } \Sigma^* \to \Sigma^*\}$. We often identify C with C^t, where t is the trivial oracle defined by $t(x) = 0$ for all $x \in \Sigma^*$, and the definition of C^f will depend on the computational model for the class in question. For classes defined in terms of Turing machines, we use *oracle* Turing machines, which have a special oracle tape which can be used to query the oracle and receive a response. In circuit models of computation, special gates are used to query the oracle.

Oracle relativization has been useful in assessing the viability of several common proof techniques for answering complexity theoretic questions. For example, the celebrated theorem of Baker, Gill, and Solovay states that there exist oracles f and g such that $\mathsf{P}^f = \mathsf{NP}^f$ and $\mathsf{P}^g \neq \mathsf{NP}^g$ [**BGS75**]. (For definitions of the classes P and NP, see Subsection 4.1.1.) Therefore, if a proof technique relativizes—that is, if the proof is independent of any oracles that are applied to the complexity classes involved—it cannot be used to settle the P vs. NP question. This obstacle is known as the *relativization barrier*.

Later, Aaronson and Wigderson introduced a refinement known as the *algebrization barrier* [**AW09**]. More recently, Aydinlioğlu and Bach reformulated the idea of an algebrizing proof into one that relativizes with respect to a class of oracles they refer to as *affine*. In this thesis, we refer to these oracles as *algebraic*.

2.3. Complexity Class Operators

This section, and the theory of complexity class operators generally, is based on a paper by Zachos and Pagourtzis [**ZP03**].

A *complexity class operator* op is an inclusion-preserving automorphism on the set of all complexity classes, written as $op \cdot \mathsf{C}$ for a complexity class C. Thus, if $\mathsf{C} \subseteq \mathsf{D}$, then $op \cdot \mathsf{C} \subseteq op \cdot \mathsf{D}$. Complexity Zoology's knowledge of operators consists of inequalities of the form $op_1 \leq op_2$ and quadratic relations of the form $op_1 \cdot op_2 = op_3 \cdot op_4$.

DEFINITION 2.3.1. For complexity class operators op_1, op_2, op_3, and op_4, we denote by $op_1 \leq op_2$ the proposition that $(op_1 \cdot \mathsf{C})^f \subseteq (op_2 \cdot \mathsf{C})^f$ for each class C and oracle f, and we denote by $op_1 \cdot op_2 = op_3 \cdot op_4$ the proposition that $(op_1 \cdot op_2 \cdot \mathsf{C})^f = (op_3 \cdot op_4 \cdot \mathsf{C})^f$ for each class C and oracle f.

2.3.1. Definitions. The definitions of the following complexity class operators preserve relativization. In other words, if an operator op is defined by the property that

$$op \cdot \mathsf{C} = \{\mathscr{L} \subseteq \Sigma^* : \varphi(\mathscr{L}, \mathsf{C})\}$$

for any complexity class C, then the relativized version of $op \cdot \mathsf{C}$ is

$$(op \cdot \mathsf{C})^f = \{\mathscr{L} \subseteq \Sigma^* : \varphi(\mathscr{L}, \mathsf{C}^f)\}.$$

The simplest operators are id, the identity operator; co, which swaps "yes" and "no" answers to each decision problem; and $cocap$, which takes the intersection of a class with its complement.

DEFINITION 2.3.2. For each complexity class C, we set

$$id \cdot \mathsf{C} := \{\mathscr{L} \subseteq \Sigma^* : \mathscr{L} \in \mathsf{C}\} = \mathsf{C},$$

$$co \cdot \mathsf{C} := \{\mathscr{L} \subseteq \Sigma^* : \Sigma^* \setminus \mathscr{L} \in \mathsf{C}\},$$

$$cocap \cdot \mathsf{C} := \{\mathscr{L} \subseteq \Sigma^* : \mathscr{L} \in \mathsf{C} \,\&\, \mathscr{L} \in co \cdot \mathsf{C}\} = \mathsf{C} \cap (co \cdot \mathsf{C}).$$

A class is **symmetric** if $\mathsf{C} = co \cdot \mathsf{C}$ with respect to every oracle.

For example, the class P is symmetric, while NP is not, because there is an oracle f relative to which $\mathsf{NP}^f \neq \mathsf{coNP}^f$ (although, of course, this is an open problem in the absence of an oracle).

The *poly* operator adds a polynomial-length *advice string* to each input.

DEFINITION 2.3.3. For a complexity class C, we define

$$poly \cdot \mathsf{C} = \{\mathscr{L} \subseteq \Sigma^* : (\exists \mathscr{L}' \in \mathsf{C}, |p(n)| = O(n^*))(\forall x \in \Sigma^*)[x \in \mathscr{L} \Longleftrightarrow \langle x, p(|x|)\rangle \in \mathscr{L}']\}.$$

We allow advice functions to map to the null string ε of length zero. In the case of tuples, $\langle x, \varepsilon \rangle$ should be understood to be x, so that, as we will see, $poly \cdot \mathsf{C}$ always contains C. For most classes with polynomial advice, we write $poly \cdot \mathsf{C} = \mathsf{C}/\mathsf{poly}$; we have, for instance, P/poly, $\mathsf{NP}/\mathsf{poly}$, and $\mathsf{BQP}/\mathsf{poly}$. We also use the suffixes $/\mathsf{mpoly}$ and $/\mathsf{qpoly}$, meaning *Merlinized polynomial advice* and *quantum polynomial advice*, respectively, but neither of these can be rigorously defined as an operator. Merlinized polynomial advice is used to exempt a probabilistic complexity class from satisfying a normally required probability gap when the advice string is unhelpful. Quantum polynomial advice consists of a string of qubits rather than classical bits, and the qubits can be in any state. See Subsection 4.1.1 for an example of a class with quantum polynomial advice and Subsection 4.2.12 for an explanation of Merlinized polynomial advice.

The operators $\oplus$, $\mathcal{N}$, and $\mathcal{P}$ are all defined in terms of certificates, strings whose lengths are polynomials of the length of the original input. For a quantifier Q, we can define an operator op_Q by

$$op_Q \cdot \mathsf{C} := \{\mathscr{L} \subseteq \Sigma^* : (\exists \mathscr{L}' \in \mathsf{C}, p(n) = O(n^*))(\forall x \in \Sigma^*)[x \in \mathscr{L} \Longleftrightarrow (Qy \in \Sigma^{p(|x|)})[\langle x, y\rangle \in \mathscr{L}']]\}.$$

The aforementioned operators are then equal to op_Q for different choices of Q.

DEFINITION 2.3.4. The operators $\oplus$, $\mathcal{N}$, and $\mathcal{P}$ are defined as follows for a complexity class C:

- $\oplus \cdot \mathsf{C} := op_Q \cdot \mathsf{C}$, where $(Qy \in S)$ means "for an odd number of $y \in S$."
- $\mathcal{N} \cdot \mathsf{C} := op_Q \cdot \mathsf{C}$, where $(Qy \in S)$ means $(\exists y \in S)$.
- $\mathcal{P} \cdot \mathsf{C} := op_Q \cdot \mathsf{C}$, where $(Qy \in S)$ means "for at least 1/2 of all $y \in S$."

$\oplus$ is read as "parity."

The bounded probabilistic operator $\mathcal{BP}$ is defined similarly.

DEFINITION 2.3.5. For each complexity class C,

$$\mathcal{BP}\cdot\mathsf{C} := \{\mathscr{L}\subseteq\Sigma^* : (\exists\mathscr{L}', p(n)=O(n^*))(\forall x\in\Sigma^*)[[x\in\mathscr{L}\Longrightarrow(\exists_{>2/3}\, y\in\Sigma^{p(|x|)})[\langle x,y\rangle\in\mathscr{L}']]$$
$$\&\,[x\notin\mathscr{L}\Longrightarrow(\exists_{>2/3}\, y\in\Sigma^{p(|x|)})[\langle x,y\rangle\notin\mathscr{L}']]]\},$$

where $(\exists_{>2/3}\, y\in\Sigma^{p(|x|)})$ is understood to mean "for more than 2/3 of all $y\in\Sigma^{p(|x|)}$."

All of these operators are named in such a way that they suggest the definitions of common complexity classes; for example, $\mathsf{NP}=\mathcal{N}\cdot\mathsf{P}$, $\mathsf{PP}=\mathcal{P}\cdot\mathsf{P}$, $\mathsf{BPP}=\mathcal{BP}\cdot\mathsf{P}$, and $\oplus\mathsf{P}=\oplus\cdot\mathsf{P}$.

Also, we have exppad, which adds an exponential length of zeros to input, generally for the purpose of buying additional computational time.

DEFINITION 2.3.6. Write $f=O(2^{poly})$ if $f(n)=O(2^{p(n)})$ for some $p(n)=O(n^*)$. Then, for a complexity class C,

$$\mathit{exppad}\cdot\mathsf{C} := \{\mathscr{L}\subseteq\Sigma^* : (\exists\mathscr{L}'\in\mathsf{C}, f=O(2^{poly}))[x\in\mathscr{L}\Longleftrightarrow x0^{f(|x|)}\in\mathscr{L}']\}.$$

NEXP, for example, is not defined to be $\mathcal{N}\cdot\mathsf{EXP}$, but rather $\mathit{exppad}\cdot\mathsf{NP}$.

Finally, note that for any fixed complexity class C, the map $\mathsf{C}'\mapsto\mathsf{C}^{\mathsf{C}'}$ defines an operator. In Complexity Zoology, $\mathsf{C}\mapsto\mathsf{P}^{\mathsf{C}}$, where P is the class of polynomial-time computable languages, is a declared operator.

2.3.2. Properties of Complexity Classes. Proving the properties of complexity operators often requires that the underlying complexity classes themselves have certain regularity properties. First, every complexity class of interest should be *nontrivial* in the sense that it contains a nonempty language not equal to Σ^*. We also expect that if $\mathscr{L}\in\mathsf{C}$, then any languages that are reducible to $\mathscr{L}$ in polynomial time are also in C.

DEFINITION 2.3.7. A complexity class C is **closed under polynomial-time reductions** if for every $\mathscr{L}\in\mathsf{C}$ and every function $f\in\mathsf{FP}$,

$$f^{-1}[\mathscr{L}]=\{x\in\Sigma^* : f(x)\in\mathscr{L}\}\in\mathsf{C}.$$

Every class in this project is relativizingly nontrivial and polynomial-time self-reducible, so that for each oracle g, if $f \in \mathsf{FP}^g$ and $\mathscr{L} \in \mathsf{C}^g$ then $f^{-1}[\mathscr{L}] \in \mathsf{C}^g$. As a result, P lies at the bottom of Complexity Zoology's inclusion hierarchy.

PROPOSITION 2.3.1. *If* C *is a nontrivial complexity class that is closed under polynomial-time reductions, then* $\mathsf{P} \subseteq \mathsf{C}$.

PROOF. Fix a nontrivial language $\mathscr{L} \in \mathsf{C}$, so that $\mathscr{L} \neq \emptyset$ and $\mathscr{L} \in \Sigma^*$. Then there exists $x_1 \in \mathscr{L}$ and $x_0 \notin \mathscr{L}$.

Now suppose $\mathscr{L}' \in \mathsf{P}$. Then define $f : \Sigma^* \to \Sigma^*$ to be

$$f(x) = \begin{cases} x_1 & \text{if } x \in \mathscr{L}', \\ x_0 & \text{if } x \notin \mathscr{L}'. \end{cases}$$

We have $f \in \mathsf{FP}$, since it can be determined whether or not $x \in \mathscr{L}'$ in polynomial time, and then writing x_1 or x_0 can be accomplished in constant time. Therefore $\mathscr{L}' = f^{-1}[\mathscr{L}] \in \mathsf{C}$ by polynomial-time self-reducibility, so we can conclude that $\mathsf{P} \subseteq \mathsf{C}$. $\square$

Additionally, complexity classes should be closed under joins, projections, and polynomial majorities. The *join* of a pair of languages $\mathscr{L}, \mathscr{L}' \subseteq \Sigma^*$ is

$$\mathscr{L} \oplus \mathscr{L}' = \{x \in \Sigma^* : (x = 0y \,\&\, y \in \mathscr{L}) \text{ or } (x = 1y \,\&\, y \in \mathscr{L}')\}.$$

The *0-projection* of a language $\mathscr{L}$ is

$$\{x \in \Sigma^* : 0x \in \mathscr{L}\},$$

and the *1-projection* is defined similarly. Given a complexity class C, a language $\mathscr{L}$ is a *polynomial majority* of C if there exist $\mathscr{L}' \in \mathsf{C}$, $p(n) = O(n^*)$ such that for each $x \in \Sigma^*$, $x \in \mathscr{L}$ if and only if $\langle x, m \rangle \in \mathscr{L}'$ for a majority of $m \in \{0, \dots, p(|x|)\}$.

2.3.3. Relations and Inclusions.

PROPOSITION 2.3.2. *The* $\mathcal{id}$*,* $\mathcal{co}$*, and* $\mathcal{cocap}$ *operators satisfy the following properties:*

(1) $\mathcal{cocap} \leq \mathcal{co}$ *and* $\mathcal{cocap} \leq \mathcal{id}$*;*

(2) $\mathcal{co}$ *is involutive, so that* $\mathcal{co} \cdot \mathcal{co} = \mathcal{id}$*;*

(3) $co \cdot cocap = cocap \cdot co = cocap$.

PROOF. (1) and (2) are immediate from the definitions of the operators. For (3), we have

$$\begin{aligned} cocap \cdot co \cdot \mathsf{C} &= (co \cdot \mathsf{C}) \cap (co \cdot co \cdot \mathsf{C}) \\ &= (co \cdot \mathsf{C}) \cap \mathsf{C} \\ &= cocap \cdot \mathsf{C}, \end{aligned}$$

and

$$\begin{aligned} \mathscr{L} \in co \cdot cocap \cdot \mathsf{C} &\iff \Sigma^* \setminus \mathscr{L} \in cocap \cdot \mathsf{C} \\ &\iff \Sigma^* \setminus \mathscr{L} \in \mathsf{C} \,\&\, \Sigma^* \setminus \mathscr{L} \in co \cdot \mathsf{C} \\ &\iff \mathscr{L} \in co \cdot \mathsf{C} \,\&\, \mathscr{L} \in co \cdot co \cdot \mathsf{C} \\ &\iff \mathscr{L} \in co \cdot \mathsf{C} \,\&\, \mathscr{L} \in \mathsf{C} \\ &\iff \mathscr{L} \in cocap \cdot \mathsf{C}. \end{aligned}$$

□

For many operators, it is the case that $\mathsf{C} \subseteq op \cdot \mathsf{C}$ for every C, because the definitions of these classes include an additional certificate or advice string that can be ignored.

PROPOSITION 2.3.3. *$id \subseteq op$, where $op = poly, \oplus, \mathcal{BP}, \mathcal{P}, \mathcal{N}$ or $exppad$.*

PROOF. Fix $\mathscr{L} \in \mathsf{C}$. Then $\mathscr{L} \in op \cdot \mathsf{C}$ for each possible choice of op:

- If $op = poly$, take $\mathscr{L}' = \mathscr{L}$ and $p(n) = \varepsilon$ for all $n \in \mathbb{N}$ in the definition of $poly \cdot \mathsf{C}$.
- If $op = \oplus, \mathcal{BP}, \mathcal{P}$, or $\mathcal{N}$, take $\mathscr{L}' = \mathscr{L}$ and $p(n) = 0$ for all $n \in \mathbb{N}$ in the definition of $op \cdot \mathsf{C}$.
- If $op = exppad$, take $\mathscr{L}' = \mathscr{L}$ and $f(n) = \varepsilon$ for all $n \in \mathbb{N}$ in the definition of $exppad \cdot \mathsf{C}$.

□

Since the condition $\mathscr{L} \in \mathcal{BP} \cdot \mathsf{C}$ is a strengthening of the condition that $\mathscr{L} \in \mathcal{P} \cdot \mathsf{C}$, the following is immediate.

PROPOSITION 2.3.4. *$\mathcal{BP} \leq \mathcal{P}$.*

We next consider some commutativity properties.

PROPOSITION 2.3.5. *$co \cdot op = op \cdot co$, where $op = \mathcal{BP}, \mathcal{P}$, or poly.*

PROOF. For each of the possible choices of op, the definition of $op \cdot \mathsf{C}$ has the following form:

$$op \cdot \mathsf{C} := \{\mathscr{L} \subseteq \Sigma^* : (\exists \mathscr{L}' \in \mathsf{C})\, \psi(\mathscr{L}, \mathscr{L}')\},$$

where $\psi(\mathscr{L}, \mathscr{L}')$ is a proposition having the property that

$$\psi(\mathscr{L}, \Sigma^* \setminus \mathscr{L}') \Longleftrightarrow \psi(\Sigma^* \setminus \mathscr{L}, \mathscr{L}').$$

Thus,

$$\begin{aligned} co \cdot op \cdot \mathsf{C} &= \{\mathscr{L} \subseteq \Sigma^* : (\exists \mathscr{L}' \in \mathsf{C})\, \psi(\Sigma^* \setminus \mathscr{L}, \mathscr{L}')\} \\ &= \{\mathscr{L} \subseteq \Sigma^* : (\exists \mathscr{L}' \in \mathsf{C})\, \psi(\mathscr{L}, \Sigma^* \setminus \mathscr{L}')\} \\ &= \{\mathscr{L} \subseteq \Sigma^* : (\exists \mathscr{L}' \in co \cdot \mathsf{C})\, \psi(\mathscr{L}, \mathscr{L}')\} \\ &= op \cdot co \cdot \mathsf{C} \end{aligned}$$

for each possible choice of op. □

A similar argument, based on the structure of the definitions of the relevant operators, can be used to show that $poly$ commutes with $\oplus$, $\mathcal{N}$, and $\mathcal{P}$.

PROPOSITION 2.3.6. *If complexity classes are assumed to be closed under polynomial-time reductions, then $poly \cdot op = op \cdot poly$, where $op = \oplus, \mathcal{N}$, or $\mathcal{P}$.*

PROOF. We say that $\mathscr{L} \in poly \cdot op \cdot \mathsf{C}$ if there exist $\mathscr{L}' \in \mathsf{C}$, $p(n) = O(n^*)$, and $|q(n)| = O(n^*)$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \Longleftrightarrow (Qy \in \Sigma^{p(|\langle x, q(|x|)\rangle|)})[\langle\langle x, q(|x|)\rangle, y\rangle \in \mathscr{L}'],$$

where Q is the quantifier in the definition of the operator that is being considered. Similarly, we say that $\mathscr{L} \in op \cdot poly \cdot \mathsf{C}$ if there exist $\mathscr{L}' \in \mathsf{C}$, $p(n) = O(n^*)$, and $|q(n)| = O(n^*)$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \Longleftrightarrow (Qy \in \Sigma^{p(|x|)})[\langle\langle x, y\rangle, q(|\langle x, y\rangle|)\rangle \in \mathscr{L}'].$$

The condition $\mathscr{L} \in \mathit{poly} \cdot \mathit{op} \cdot \mathsf{C}$ is equivalent to the condition that there are $\mathscr{L}' \in \mathsf{C}$, $\bar{p} \in O(n^*)$, and $q \in |O(n^*)|$ such that for all $x \in \Sigma^*$,

$$x \in \mathscr{L} \iff (Qy \in \Sigma^{\bar{p}(|x|)})[\langle\langle x, q(|x|)\rangle, y\rangle \in \mathscr{L}'].$$

For instance, if $\mathscr{L} \in \mathit{poly} \cdot \mathit{op} \cdot \mathsf{C}$, then we can set $\bar{p}(n) = p(N)$, where $N = |\langle x, q(|x|)\rangle|$ for $|x| = n$. Likewise, in the conditions for $\mathscr{L} \in \mathit{op} \cdot \mathit{poly} \cdot \mathsf{C}$ we can replace q with a $\bar{q}$ so that

$$x \in \mathscr{L} \iff (Qy \in \Sigma^{p(|x|)})[\langle\langle x, y\rangle, \bar{q}(|x|)\rangle \in \mathscr{L}'].$$

The rewritten conditions for $\mathscr{L} \in \mathit{poly} \cdot \mathit{op} \cdot \mathsf{C}$ and $\mathscr{L} \in \mathit{op} \cdot \mathit{poly} \cdot \mathsf{C}$ are then equivalent to each other because a mapping between $\langle\langle x, z\rangle, y\rangle$ and $\langle\langle x, y\rangle, z\rangle$ is polynomial-time computable. $\square$

PROPOSITION 2.3.7. *If complexity classes are assumed to be closed under polynomial-time reductions, then $\mathit{co} \cdot \oplus = \oplus \cdot \mathit{co}$.*

Finally, we consider the properties of the operator $\mathsf{C} \mapsto \mathsf{P}^{\mathsf{C}}$.

PROPOSITION 2.3.8. *For any complexity class C,*

(1) $\mathsf{C} \subseteq \mathsf{P}^{\mathsf{C}}$*;*
(2) $\mathit{co} \cdot \mathsf{C} \subseteq \mathsf{P}^{\mathsf{C}}$*;*
(3) $\mathit{co} \cdot \mathsf{P}^{\mathsf{C}} = \mathsf{P}^{\mathit{co} \cdot \mathsf{C}} = \mathsf{P}^{\mathsf{C}}$.

PROOF. If $\mathscr{L} \in \mathsf{C}$ and f is the decision function for $\mathscr{L}$, then $\mathscr{L} \in \mathsf{P}^f \subseteq \mathsf{P}^{\mathsf{C}}$. Hence $\mathsf{C} \subseteq \mathsf{P}^{\mathsf{C}}$. Moreover, P^f is a symmetric class for every f, and so

$$\begin{aligned}
\mathscr{L} \in \mathit{co} \cdot \mathsf{P}^{\mathsf{C}} &\iff \Sigma^* \setminus \mathscr{L} \in \mathsf{P}^{\mathsf{C}} \\
&\iff (\exists f \in \mathsf{C})[\Sigma^* \setminus \mathscr{L} \in \mathsf{P}^f] \\
&\iff (\exists f \in \mathsf{C})[\mathscr{L} \in \mathsf{P}^f] \\
&\iff \mathscr{L} \in \mathsf{P}^{\mathsf{C}},
\end{aligned}$$

and

$$\begin{aligned}\mathscr{L} \in \mathsf{P}^{co\cdot\mathsf{C}} &\iff (\exists f \in \mathsf{C})[\mathscr{L} \in \mathsf{P}^{1-f}] \\ &\iff (\exists f \in \mathsf{C})[\mathscr{L} \in \mathsf{P}^{f}] \\ &\iff \mathscr{L} \in \mathsf{P}^{\mathsf{C}},\end{aligned}$$

because $\mathsf{P}^f = \mathsf{P}^{1-f}$ for every oracle f. Thus, (1) and (3) are true. (2) then follows immediately, because $\mathsf{C} \subseteq \mathsf{P}^{\mathsf{C}} \Longrightarrow co\cdot\mathsf{C} \subseteq co\cdot\mathsf{P}^{\mathsf{C}} \Longrightarrow co\cdot\mathsf{C} \subseteq \mathsf{P}^{\mathsf{C}}$. □

PROPOSITION 2.3.9. *For any non-trivial class* C *that is closed under joins and polynomial-time reductions,* $poly\cdot\mathsf{P}^{\mathsf{C}} = \mathsf{P}^{poly\cdot\mathsf{C}}$. *If* C *is also closed under polynomial majorities, then* $\mathcal{BP}\cdot\mathsf{P}^{\mathsf{C}} = \mathsf{P}^{\mathcal{BP}\cdot\mathsf{C}}$.

PROOF OF FIRST EQUATION. First, we show that it is unconditionally the case that $\mathsf{P}^{poly\cdot\mathsf{C}} \subseteq poly\cdot\mathsf{P}^{\mathsf{C}}$. Suppose that $\mathscr{L} \in \mathsf{P}^{poly\cdot\mathsf{C}}$. Then there is a polynomial-time algorithm with f-oracle that computes $\mathscr{L}$, where f is a decision function for a language $\mathscr{L}' \in poly\cdot\mathsf{C}$. By the definition of the $poly$ operator, there exists a language $\mathscr{L}'' \in \mathsf{C}$ and an advice function $|p(n)| = O(n^*)$ such that $x \in \mathscr{L}'$ if and only if $\langle x, p(|x|)\rangle \in \mathscr{L}''$.

Let g indicate the decision function for $\mathscr{L}''$. Also, let $q(n) = O(n^*)$ denote the time bound for the P^f algorithm for $\mathscr{L}$, so that the question of whether $x \in \mathscr{L}$ is decided in at most $q(|x|)$ computational steps. Define $P : \mathbb{N} \to \Sigma^*$ so that, for each $n \in \mathbb{N}$, $P(n)$ is the concatenation $p(0)p(1)\dots p(q(n))$.

The following algorithm in $poly\cdot\mathsf{P}^g$ decides whether $x \in \mathscr{L}$:

(1) The advice function is P. Note that $|P(n)| = O(n^*)$, because for sufficiently large n $|P(n)|$ is at most $q(n)|p(q(n))|$.
(2) Follow the P^f algorithm for $\mathscr{L}$ exactly, except when there is an oracle call.
(3) When an oracle call to f occurs with the string $y \in \Sigma^*$, replace it with an oracle call to f' with the string $\langle y, p(|y|)\rangle$. This oracle call is possible because $P(|x|)$ contains the advice strings for all y that are short enough for the P^f algorithm to be able to query the oracle.

Thus, we have $\mathscr{L} \in poly\cdot\mathsf{P}^g \subseteq poly\cdot\mathsf{P}^{\mathsf{C}}$, and we can conclude that $\mathsf{P}^{poly\cdot\mathsf{C}} \subseteq poly\cdot\mathsf{P}^{\mathsf{C}}$ in all cases.

For the inclusion $poly\cdot\mathsf{P}^{\mathsf{C}} \subseteq \mathsf{P}^{poly\cdot\mathsf{C}}$, suppose that $\mathscr{L} \in poly\cdot\mathsf{P}^{\mathsf{C}}$. This means that there exists a P^f algorithm that decides $\mathscr{L}$ when provided with some advice function $|p(n)| = O(n^*)$, where f is the decision function of some $\mathscr{L}' \in \mathsf{C}$. Define $g : \Sigma^* \to \Sigma$ according to the following rules:

- If $x = 0\langle y,z\rangle$, then $g(x)$ is equal to the zth bit of $p(|y|)$.
- If $x = 1y$, then $g(x) = A(y)$.

The language $\mathscr{L}''$ determined by g is the join of two languages. One, which we will call $\mathscr{L}_1''$, is the set of all $\langle y,z\rangle$ such that the zth bit of $p(|y|)$ is 1; the second language is $\mathscr{L}'$.

We claim that $\mathscr{L}''$ lies in $\mathit{poly}\cdot\mathsf{C}$. To prove this, it is enough to show that $\mathscr{L}_1'' \in \mathsf{P/poly}$. Then $\mathsf{P/poly} \subseteq \mathit{poly}\cdot\mathsf{C}$ (we know that $\mathsf{P} \subseteq \mathsf{C}$ because C is assumed to be nontrivial and closed under polynomial-time reductions), and $\mathscr{L}' \in \mathsf{C} \subseteq \mathit{poly}\cdot\mathsf{C}$, so $\mathscr{L}'' \in \mathit{poly}\cdot\mathsf{C}$ by the hypothesis that C, and therefore $\mathit{poly}\cdot\mathsf{C}$, is closed under joins.

To see that $\mathscr{L}_1'' \in \mathsf{P/poly}$, let $|P(n)| = O(n^*)$ be the function defined by the concatenation

$$P(n) = p(0)p(1)\ldots p(n).$$

Then, given $\langle y,z\rangle$, P can be used as an advice function to check whether the zth bit of $p(|y|)$ is 1.

Hence $\mathscr{L}'' \in \mathit{poly}\cdot\mathsf{C}$. To show that $\mathscr{L} \in \mathsf{P}^{\mathit{poly}\cdot\mathsf{C}}$, we show that $\mathscr{L} \in P^g$. The following is a P^g algorithm for deciding whether $x \in \mathscr{L}$:

(1) First, extract the advice string $p(|x|)$ from the g-oracle. Make the oracle queries $0\langle x,j\rangle$, $j \leq |p(|x|)|$ until the entirety of $p(|x|)$ has been recorded.

(2) Carry out the rest of the computation according to the $\mathit{poly}\cdot\mathsf{P}^f$ algorithm. Replace oracle queries to f about the string y with oracle queries to g about the string y.

Therefore $\mathscr{L} \in \mathsf{P}^{\mathit{poly}\cdot\mathsf{C}}$, concluding the proof that $\mathsf{P}^{\mathit{poly}\cdot\mathsf{C}} = \mathit{poly}\cdot\mathsf{P}^{\mathsf{C}}$. □

CHAPTER 3

The Complexity Zoology Program

In this chapter, we describe the algorithm that Complexity Zoology follows. Here is a high-level description of the procedure:

(1) **Read input:** The program parses the plain-text input files (one for classes and one for operators).

(2) **Process equalities:** The input file contains statements of the form `C1` = `C2`, where `C1` and `C2` are names for complexity classes. Statements of this form are understood to indicate that the two classes are equal with respect to every oracle. Complexity Zoology uses the transitivity of equality to learn which classes are equal and then chooses an official name for each class according to preferences specified in the input file.

(3) **Expand operators:** Zoology understands each operator as a partial function on the set of unique classes in the data set. Using the rules specified in the input file for operators, the program expands each partial function to be as large as possible.

(4) **Deduce:** The system applies its list of inference rules to deduce inclusions, oracle separations, and open problems.

(5) **Postprocess:** The program prepares the expanded knowledge database for output. In particular, Complexity Zoology computes the relations that must be shown on the final diagram.

(6) **Output:** Zoology produces an HTML file with clickable diagrams showing complexity class relationships in each modal world.

3.1. Input Files and Syntax

Complexity Zoology reads its initial data from two plain text files: one consisting of complexity classes and their inclusions and oracle separations (`classes.txt`), and another consisting of complexity class operators and their relations (`operators.txt`). Both classes and operators must be declared in their respective files before they can be used. If an undeclared class or operator is used in some inclusion, separation, or relation, Complexity Zoology will halt and print an error. A declaration consists of a line of text having the following form:

```
NAME : description :  keyword1, keyword2, keyword3
```

Here, `NAME` is the name by which the class or operator is referenced both internally and in the output. Any alphanumeric characters, as well as hyphens, can be used for names. The `description` is a short phrase used to indicate the nature of the class or operator to a human reader; the program itself does not use the description. Like names, descriptions should consist of alphanumeric characters and hyphens, although whitespace is also allowed. Finally, keywords can optionally be included in a declaration. Keywords follow the same naming rules as class and operator names, and multiple keywords must be separated by commas, which can be surrounded by any amount of whitespace. If there are no keywords, the second colon must be omitted.

Keywords are used to provide additional information about the class or operator being declared. Most often, a keyword is shorthand for commonly arising relations. For example, the class keyword `symmetric` is equivalent to including the line `C = co.C`, where `C` is the name of the declared class. The following keywords are defined for complexity classes:

- `hidden` – The class is suppressed in the final output, but it is still included for the purposes of calculation and deduction.
- `ignore` – The class is not included in calculation or output; any relations involving the class are effectively commented out.
- `preferred` – If this class is equal to another, this class should be the preferred name.
- `preferred[#]` – Here, the symbol `#` should be replaced with a positive integer and indicates the *preference rank* of the declared class. When Complexity Zoology chooses a name for equal complexity classes, it favors those with the smaller preference rank. The `preferred` keyword is equivalent to `preferred[1]`.
- `symmetric` – The class is *symmetric* in the sense of being equivalent to its complement: for a complexity class C, this means that $\mathsf{C} = \mathsf{co} \cdot \mathsf{C}$ relative to every oracle.

For operators, there is currently only one keyword:

- `idempotent` – Applying the operator to a class a second time has the same effect as applying it once. This keyword is equivalent to the relation `op.op = op`.

Aside from class and operator declarations, the input files also include *relations* describing what is (initially) known about the classes and operators. For complexity classes, relations are either statements of equality, statements of inclusion, or statements of oracle separation. Suppose that C_1 and C_2 are classes declared with the names `C1` and `C2`, respectively. Then we have these valid relations:

- `C1 = C2` $\leftrightarrow \mathsf{C}_1^A = \mathsf{C}_2^A$ for every oracle A.
- `C1 < C2` $\leftrightarrow \mathsf{C}_1^A \subseteq \mathsf{C}_2^A$ for every oracle A.
- `C1 r< C2` $\leftrightarrow \mathsf{C}_1^A \subseteq \mathsf{C}_2^A$ with probability 1 for a random oracle A.
- `C1 a< C2` $\leftrightarrow \mathsf{C}_1^A \subseteq \mathsf{C}_2^A$ for every algebraic oracle A.
- `C1 t< C2` $\leftrightarrow \mathsf{C}_1 \subseteq \mathsf{C}_2$ relative to the trivial oracle.
- `C1 x< C2` $\leftrightarrow \mathsf{C}_1^A \subseteq \mathsf{C}_2^A$ for some algebraic oracle A.
- `C1 o< C2` $\leftrightarrow \mathsf{C}_1^A \subseteq \mathsf{C}_2^A$ for some oracle A.
- `C1 osep C2` $\leftrightarrow \mathsf{C}_1^A \nsubseteq \mathsf{C}_2^A$ for some oracle A.
- `C1 rsep C2` $\leftrightarrow \mathsf{C}_1^A \nsubseteq \mathsf{C}_2^A$ with probability 1 for a random oracle A.
- `C1 xsep C2` $\leftrightarrow \mathsf{C}_1^A \nsubseteq \mathsf{C}_2^A$ for some algebraic oracle A.
- `C1 tsep C2` $\leftrightarrow \mathsf{C}_1 \nsubseteq \mathsf{C}_2$ relative to the trivial oracle.
- `C1 asep C2` $\leftrightarrow \mathsf{C}_1^A \nsubseteq \mathsf{C}_2^A$ for every algebraic oracle A.
- `C1 sep C2` $\leftrightarrow \mathsf{C}_1^A \nsubseteq \mathsf{C}_2^A$ for every oracle A.

For operators op_1, op_2, op_3 and op_4 with declared names `op1`, `op1`,`op1`, and `op1`, respectively, we have these relations:

- `op1.op2 = op3.op4` $\leftrightarrow op_1 \cdot op_2 = op_3 \cdot op_4$. Omitting one of the operators on either side of the equation is allowed; e.g., `op1.op2 = op3`, which is interpreted as $op_1 \cdot op_2 = op_3$. Complexity Zoology implements this by replacing the missing operator with the identity operator id.
- `op1 z= op2` $\leftrightarrow op_1$ and op_2 commute. This is equivalent to including the line `op1.op2 = op2.op1`.
- `op1 p= op2` $\leftrightarrow op_2$ absorbs op_1 on the left and right. This is equivalent to including the lines `op1.op2 = op2` and `op2.op1 = op2`.
- `op1 < op2` $\leftrightarrow op_1 \leq op_2$; i.e., $op_1 \cdot \mathsf{C} \subseteq op_2 \cdot \mathsf{C}$ for every complexity class C.

For all relations, with the exception of the quadratic operator relations of the form `op1.op2 = op3.op4`, it is possible to include multiple classes or operators separated by commas:

```
C1,C2 = C3,C4,C5
```

This example is equivalent to including six lines: `C1 = C3`, `C1 = C4`, `C1 = C5`, `C2 = C3`, `C2 = C4`, and `C2 = C5`.

Lastly, there are two ways to include text that Complexity Zoology will ignore: *comments* and *citations*. Comments consist of the character `#` followed by all text up to the end of the current line. Citations consist of text surrounded by square brackets. In the current version of the input files, citations are used both to refer to this documentation's bibliography and to annotate common arguments.

3.2. Operator Propagation and Inference

Operators in Complexity Zoology are implemented as partial functions on the set of all distinct complexity classes. Processing of operators occurs after the processing of equality statements, so we can assume that we have a quotient map $q : \mathscr{N} \to \mathscr{C}$, where $\mathscr{N}$ is the set of all names for complexity classes, $\mathscr{C}$ is the set of all distinct complexity classes, and $q(x) = q(y)$ if and only if x and y are names for the same class. The system's understanding of a complexity class operator op is a partial function $op : \mathscr{C} \rightharpoonup \mathscr{C}$; i.e., a function whose domain is a subset of $\mathscr{C}$ that takes values in $\mathscr{C}$. The function that defines op is necessarily a partial one, because for a given complexity class C, the class $\mathit{op} \cdot \mathsf{C}$ might not be declared in Complexity Zoology's data set.

Initially, we assume that the following is true of our operator partial functions:

(1) $\mathit{id}(x) = x$ for all $x \in \mathscr{C}$.
(2) If a class x has a name of the form `op.y`, where $y \in \mathscr{N}$ and `op` is the name of an operator, then we set $op(q(y)) = x$.

Then, the partial functions are expanded according to the quadratic relations specified in the input file. More specifically, suppose that one such relation is

$$\mathit{op}_1 \cdot \mathit{op}_2 = \mathit{op}_3 \cdot \mathit{op}_4.$$

Then, the operator partial functions can be expanded according to the following rules:

(1) If $op_1(op_2(x))$ and $y = op_4(x)$ are defined, then define $op_3(y) := op_1(op_2(x))$.

(2) If $op_3(op_4(x))$ and $y = op_2(x)$ are defined, then define $op_1(y) := op_3(op_4(x))$.

These rules are applied iteratively until the partial functions can be expanded no further. This process uses the same task list-based system that is used in the primary inference engine (see Section 3.3).

While the partial functions are expanding, it is possible that Complexity Zoology learns that $op(x) = y$ and $op(x) = z$, where y and z are different names for complexity classes. If this occurs, then Complexity Zoology stops and produces an error, because it is expected that all equalities between complexity classes are learned through explicit statements in the data file and the transitivity of the equality relation.

3.3. To-do List Inference Algorithm

Both the primary inference engine and the propagation of operator partial functions follow the same basic procedure. We begin with some database D_0, which can be taken to be a set of propositions that are regarded as true. We also assume that there is a set R of inference rules, which are tuples of the form

$$\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$$

for a positive integer n (although for our purposes $n \leq 2$). The inference algorithm that Complexity Zoology employs is as follows:

(1) Populate a list L with the propositions in D_0, and set $D = \emptyset$.

(2) While L is nonempty, carry out the steps (3) through (6).

(3) Remove the top proposition φ from L.

(4) If $\varphi \in D$, return to step (3).

(5) Add φ to the set D.

(6) For each inference rule $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$ and all $\varphi'_1, \varphi'_2, \ldots, \varphi'_{n-1} \in D$, check whether some permutation of $\varphi, \varphi'_1, \ldots, \varphi'_{n-1}$ matches $\varphi_1, \varphi_2, \ldots, \varphi_n$. If it does, append ψ to L.

The resulting database D has D_0 as a subset and is closed under inference rules. Moreover, this algorithm eventually terminates, because when a proposition has been removed from L once, it cannot again result in any additional proposition being appended to L. Thus, the algorithm deduces all logical consequences of the initial database D_0, and it does so faster than the naive approach of repeatedly applying all inference rules to all the propositions in D until D grows no further.

3.4. The Logic of Complexity Zoology

Propositions in Complexity Zoology are inclusions of the form $\mathsf{C}_1 \subseteq \mathsf{C}_2$, where C_1 and C_2 are complexity classes. Each such inclusion is true or false in a particular *world* in the sense of modal logic: for a world W, we write $\mathsf{C}_1 \subseteq_W \mathsf{C}_2$ to indicate that the inclusion is true in W. Each world W has a transitive dual W^* with respect to which the following inference rules are true:

$$\mathsf{C}_1 \subseteq_W \mathsf{C}_2 \,\&\, \mathsf{C}_2 \subseteq_{W^*} \mathsf{C}_3 \Longrightarrow \mathsf{C}_1 \subseteq_W \mathsf{C}_3,$$

$$\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2 \,\&\, \mathsf{C}_2 \subseteq_W \mathsf{C}_3 \Longrightarrow \mathsf{C}_1 \subseteq_W \mathsf{C}_3.$$

Additionally, there is a partial ordering $\to$ on the set of worlds such that if $W_1 \to W_2$, then the following inference rule is true:

$$\mathsf{C}_1 \subseteq_{W_1} \mathsf{C}_2 \Longrightarrow \mathsf{C}_1 \subseteq_{W_2} \mathsf{C}_2.$$

In the current version of Complexity Zoology, there are six worlds:

$$E \longleftrightarrow \text{every oracle},$$

$$A \longleftrightarrow \text{every algebraic oracle},$$

$$X \longleftrightarrow \text{some algebraic oracle},$$

$$R \longleftrightarrow \text{the random oracle},$$

$$T \longleftrightarrow \text{the trivial oracle},$$

$$O \longleftrightarrow \text{some oracle}.$$

For example, we write $\mathsf{C}_1 \subseteq_X \mathsf{C}_2$ if $\mathsf{C}_1^f \subseteq \mathsf{C}_2^f$ for some algebraic oracle f. The worlds E, A, R, and T are all *transitive* worlds in the sense that they are their own transitive duals: $E^* = E$, $A^* = A$, $R^* = R$, and $T^* = T$. On the other hand $X^* = A$ and $O^* = E$.

The remaining inference rules pertain to complexity class operators. For each operator *op* and world W,

$$\mathsf{C}_1 \subseteq_W \mathsf{C}_2 \Longrightarrow op \cdot \mathsf{C}_1 \subseteq_W op \cdot \mathsf{C}_2$$

is an inference rule. There is also a special pair of inference rules involving the co and *cocap* operators: for a complexity class C, $cocap \cdot \mathsf{C}$ is the *meet* of C and $\mathsf{co} \cdot \mathsf{C}$. For each world W, we have

$$\mathsf{C}_1 \subseteq_W \mathsf{C}_2 \,\&\, \mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2 \Longleftrightarrow \mathsf{C}_1 \subseteq_W cocap \cdot \mathsf{C}_2,$$

$$\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2 \,\&\, \mathsf{C}_1 \subseteq_W \mathsf{C}_2 \Longleftrightarrow \mathsf{C}_1 \subseteq_W cocap \cdot \mathsf{C}_2.$$

This logical framework – the propositions that specify inclusions, the worlds, and the inference rules – represents the basic knowledge and reasoning of which Complexity Zoology is capable. However, since the purpose of Complexity Zoology is to partially automate the process of surveying complexity theory, it requires a formal system that can describe not merely whether a statement is true or false, but whether it is regarded as proven, disproven, or open. To create such a logic, we introduce a four-valued system consisting of these values: proven, disproven, not proven, and not disproven. The term "open" will then be applied to propositions that are both not proven and not disproven. The advantage to this approach is that the rules of inference are more straightforward to describe than one in which the only logical values are proven, disproven, and open.

Each of the statements that we have described so far, along with the inference rules, can be formally expressed in the language of set theory. We assume that there is a set P of inclusion statements and negations thereof that are regarded as *proven*. We assume that P is closed under the logical system of Complexity Zoology. In other words, if CZ is the set of formalized inference rules that Complexity Zoology uses, and if $CZ \cup P \vdash \varphi$ (φ is a logical consequence of $CZ \cup P$), where φ is a formalized inclusion of complexity classes, then $\varphi \in P$. Thus, we regard CZ as a system that is simple enough so that any of its implications from proven facts should also be regarded as proven.

A formal inclusion φ is regarded as *proven* if it lies in P, *disproven* if its negation $\neg\varphi$ lies in P, and *open* if it is neither proven nor disproven. Also, a statement is also sometimes called *provable* if it is not disproven and *disprovable* if it is not proven. We use the following notation:

$$\begin{aligned}
P(\varphi) &\longleftrightarrow \varphi \text{ is proven,}\\
D(\varphi) &\longleftrightarrow \varphi \text{ is disproven,}\\
p(\varphi) &\longleftrightarrow \varphi \text{ is provable,}\\
d(\varphi) &\longleftrightarrow \varphi \text{ is disprovable,}\\
O(\varphi) &\longleftrightarrow \varphi \text{ is open.}
\end{aligned}$$

Complexity Zoology's data set is a partial description of the set P in the form of a list of statements that are proven, disproven, or open. Complexity Zoology then attempts to deduce as much as it can about which inclusions are proven, disproven, or open. Complexity Zoology has no understanding of complexity theory as such; its strength is in organizing the state of knowledge in the field – that is, Complexity Zoology is an expert at surveying complexity theory, not in complexity theory itself.

Internally, Complexity Zoology represents propositions as a quadruple

$$(\text{status}, \text{world}, \mathsf{C}_1, \mathsf{C}_2),$$

where the status is either proven, disproven, provable, or disprovable; the quadruple is interpreted as "the inclusion $\mathsf{C}_1 \subseteq \mathsf{C}_2$ has the specified status in the specified world." Then, from the base inference rules of the system *CZ*, we generate the full set of inference rules as follows:

(1) For any formalized inclusion φ, $P(\varphi) \Longrightarrow p(\varphi)$ and $D(\varphi) \Longrightarrow d(\varphi)$.

(2) If $\varphi_1 \,\&\cdots\&\, \varphi_n \Longrightarrow \psi$ is an inference rule ($n = 1$ or 2 in Complexity Zoology), then

$$P(\varphi_1) \,\&\cdots\&\, P(\varphi_n) \Longrightarrow P(\psi).$$

(3) The partial involutions $(P \mapsto D \mapsto P)$ and $(P \mapsto d \mapsto P, D \mapsto p \mapsto D)$ are implication-reversing. For example, if $P(\varphi) \Longrightarrow P(\psi)$, then $D(\psi) \Longrightarrow D(\varphi)$, and if

$$P(\varphi_1) \,\&\cdots\&\, P(\varphi_n) \Longrightarrow P(\psi),$$

then

$$P(\varphi_1)\,\&\cdots\&\,P(\varphi_{n-1})\,\&\,d(\psi)\Longrightarrow d(\varphi_n).$$

To see one possible rule, if $\mathsf{C}_1^A \subseteq \mathsf{C}_2^A$ for every oracle A and $\mathsf{C}_1^A \not\subseteq \mathsf{C}_3^A$ for some oracle A, then $\mathsf{C}_2^A \not\subseteq \mathsf{C}_3^A$ for some oracle A. In our notation,

$$P(\mathsf{C}_1 \subseteq_E \mathsf{C}_2)\,\&\,D(\mathsf{C}_1 \subseteq_E \mathsf{C}_3)\Longrightarrow D(\mathsf{C}_2 \subseteq_E \mathsf{C}_3).$$

3.5. Extremal Unknowns

Identifying key missing information is an important part of the survey process. Complexity Zoology has the capacity to identify a subset of unknown inclusions that are sufficient to decide all other unknowns. To be precise, recall that an inclusion $\mathsf{C}_1 \subseteq_W \mathsf{C}_2$ can exist in one of three possible states: proven, disproven, or open. However, since Complexity Zoology's data set is incomplete, it is possible that the system cannot infer which of the three possibilities applies to a given inclusion. When this occurs, we say that the inclusion is *unknown*, not to be confused with the inclusion being open. In short, to say that an inclusion is open is to say that it does not have a known proof or disproof, while to say that an inclusion is unknown means that Complexity Zoology itself does not know the status. We denote an unknown inclusion by $\mathsf{C}_1 \overset{?}{\subseteq}_W \mathsf{C}_2$.

Ideally, we would like to complete the data set so that there are no unknown inclusions. However, this is a process that potentially involves a great deal of redundancy, since listing a formerly unknown inclusion as proven (for example) may decide several other unknown inclusions. Therefore, it is helpful to identify a subset of unknown inclusions that are sufficient to decide all others. We refer to these as *extremal unknowns*.

A *most likely extremal unknown* is, roughly, an unknown inclusion that, if it were listed as proven, would not result in any other inclusions being listed as proven. Complexity Zoology considers extremal unknowns separately for each of the worlds, and for simplicity inference rules involving multiple worlds or involving operators are not considered. Thus, the only inference rules that are relevant to determining whether Complexity Zoology considers an unknown inclusion to be a most likely extremal are

$$p(\mathsf{C}_1 \subseteq_W \mathsf{C}_2)\,\&\,p(\mathsf{C}_2 \subseteq_{W^*} \mathsf{C}_3)\Longrightarrow p(\mathsf{C}_1 \subseteq_W \mathsf{C}_3),$$

$$p(\mathsf{C}_1 \subseteq_W \mathsf{C}_2)\,\&\,p(\mathsf{C}_2 \subseteq_{W^*} \mathsf{C}_3)\Longrightarrow p(\mathsf{C}_1 \subseteq_W \mathsf{C}_3),$$

where W is the world under consideration and W^* is the transitive dual. In testing whether an unknown inclusion $\mathsf{C}_1 \overset{?}{\subseteq}_W \mathsf{C}_2$ is a most likely extremal, Complexity Zoology tests whether there exists a third class C_3 that is distinct from C_1 and C_2 in W satisfying one of the following two conditions:

$$\mathsf{C}_1 \overset{?}{\subseteq}_W \mathsf{C}_3 \,\&\, p(\mathsf{C}_2 \subseteq_{W^*} \mathsf{C}_3),$$
$$\mathsf{C}_3 \overset{?}{\subseteq}_W \mathsf{C}_2 \,\&\, p(\mathsf{C}_3 \subseteq_{W^*} \mathsf{C}_1).$$

If such a C_3 exists, then the system concludes that the unknown is *not* most likely extremal.

Similarly, a *least likely extremal unknown* is intended to be an unknown inclusion that, if listed as disproven, would not result in any other unknown inclusions being decided as disproven. In this case, the relevant inference rules are

$$d(\mathsf{C}_1 \subseteq_W \mathsf{C}_2) \,\&\, p(\mathsf{C}_3 \subseteq_{W^*} \mathsf{C}_2) \Longrightarrow p(\mathsf{C}_1 \subseteq_W \mathsf{C}_3),$$
$$d(\mathsf{C}_1 \subseteq_W \mathsf{C}_2) \,\&\, p(\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_3) \Longrightarrow p(\mathsf{C}_3 \subseteq_W \mathsf{C}_2).$$

Complexity Zoology checks whether a given unknown inclusion $\mathsf{C}_1 \overset{?}{\subseteq}_W \mathsf{C}_2$ is a least likely extremal by checking whether there exists a class C_3 that is distinct from C_1 and C_2 in W and satisfies at least one of the following:

$$\mathsf{C}_1 \overset{?}{\subseteq}_W \mathsf{C}_3 \,\&\, p(\mathsf{C}_3 \subseteq_{W^*} \mathsf{C}_2),$$
$$\mathsf{C}_3 \overset{?}{\subseteq}_W \mathsf{C}_2 \,\&\, p(\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_3).$$

Complexity Zoology lists extremal unknowns of both types for each world. It also specifies whether each extremal is provable, disprovable, or completely unknown (neither provable nor disprovable).

PROPOSITION 3.5.1. *Deciding the extremal unknowns is sufficient to decide all the unknowns for a world.*

The unknowns that Complexity Zoology has identified as extremal have resulted in several interesting questions. (Examples are given in the introduction.) They have also been extremely useful in filling in large portions of the data very quickly. In particular, a disproof for a most likely inclusion or a proof for a least likely inclusion tend to settle many other unknowns.

3.6. Output

The output of Complexity Zoology is an HTML file that displays the logical consequences of the information in the input file. The ouput shows a diagram for each world W that indicates the distinct classes in W and the relationships between them. The current version of the Complexity Zoology output file can be found at `https://www.math.ucdavis.edu/~rjs/zoology.html`.

Each of the six worlds that Complexity Zoology understands can be viewed in the output file. For each world W, there is a diagram for the transitive dual W^* indicating the inclusions that are true in W^*. A black arrow from C_1 to C_2 indicates that $\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$ and $\mathsf{co}\cdot\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$, a blue arrow indicates that $\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$, a red arrow indicates that $\mathsf{co}\cdot\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$, and a green arrow indicates that $\mathit{cocap}\cdot\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$. Complexity Zoology draws the fewest arrows necessary to describe the structure of W^*. Specifically, the system describes whether an arrow should be drawn from C_1 to C_2 according to the following rules:

(1) Let $\mathsf{C}_1 \leq_1 \mathsf{C}_2$ denote the relation "$\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$ or $\mathsf{co}\cdot\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$." Let $\mathsf{C}_1 \leq_2 \mathsf{C}_2$ denote the relation $\mathit{cocap}\cdot\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$. Denote the Hasse relation of $\leq_j$ by $\leq'_j$ so that, for example, $\mathsf{C}_1 \leq'_1$ C2 indicates that $\mathsf{C}_1 \leq \mathsf{C}_2$ and there is no C_3 distinct from C_1 and C_2 such that $\mathsf{C}_1 \leq_1 \mathsf{C}_3 \leq \mathsf{C}_2$.

(2) An arrow is drawn from C_1 to C_2 if and only if $\mathsf{C}_1 \leq'_1 \mathsf{C}_2$ or $\mathsf{C}_1 \leq'_2 \mathsf{C}_2$.

(3) Suppose that, by (2), an arrow is to be drawn from C_1 to C_2. If $\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$ and $\mathsf{co}\cdot\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$, color the arrow black. If $\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$ but not $\mathsf{co}\cdot\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$, color the arrow blue. If $\mathsf{co}\cdot\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$ but not $\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$, color the arrow red. If the arrow is not to be colored black, blue, or red, then color the arrow green.

So far, the diagrams for W and its transitive dual W^* are identical. They are distinguished by the fact that they are active diagrams: by clicking on the node in the diagram corresponding to the class C_1, all nodes in the diagram are colored. Assuming the node for a first class C_1 has been clicked, the node for a second class is colored in as follows:

- The color of the left side of the node indicates the status of $\mathsf{C}_1 \subseteq_W \mathsf{C}_2$.
- The color of the right side of the node indicates the status of $\mathsf{co}\cdot\mathsf{C}_1 \subseteq_W \mathsf{C}_2$.
- The color of the middle of the node indicates the status of $\mathit{cocap}\cdot\mathsf{C}_1 \subseteq_W \mathsf{C}_2$. (If the status of $\mathsf{C}_1 \subseteq \mathsf{C}_2$ or $\mathsf{co}\cdot\mathsf{C}_1 \subseteq \mathsf{C}_2$ implies the status of $\mathit{cocap}\cdot\mathsf{C}_1 \subseteq \mathsf{C}_2$, then there is no middle color, and only the left and right halves of the node are colored.)

Green indicates that the inclusion has been proven, red indicates that the inclusion has been disproven, yellow indicates that the inclusion is open, and gray indicates that the status of the inclusion is unknown to Complexity Zoology. Different shades of gray are used to indicate the type of unknown: green-tinted for provable, red-tinted for disprovable, and untinted for completely unknown.

Finally, a table below the diagram lists alternate names for each complexity class in W^*. In other words, for each class C_1 that appears in the diagram, the table lists classes C_1 for which $\mathsf{C}_1 \subseteq_{W^*} \mathsf{C}_2$ and $\mathsf{C}_2 \subseteq_{W^*} \mathsf{C}_1$. Complexity Zoology uses its internal hierarchy of preferred names to determine which name for a complexity class should appear in the diagram, choosing arbitrarily between the possibilities having equal preference.

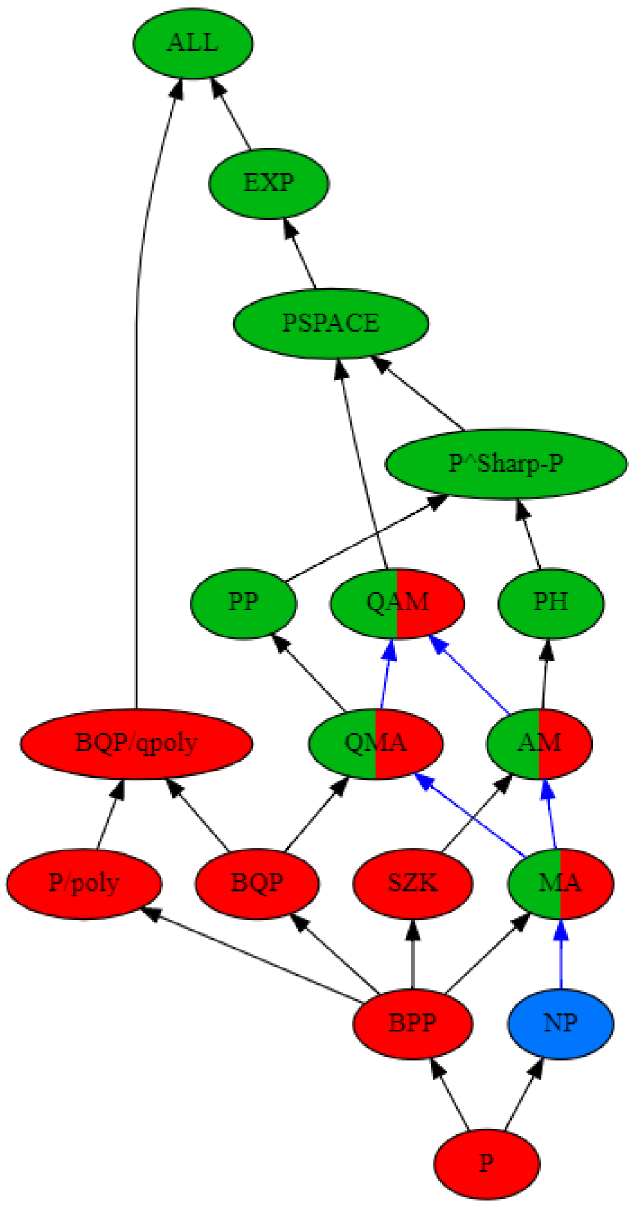

FIGURE 3.1. A sample diagram in the “all oracles” world in which NP is selected. The blue node is selected, a green node indicates that the selected class lies inside the node’s class, and a red node indicates that the selected class does not lie inside the node’s class. The color on the left side of a node corresponds to the class written in the bubble, while the color on the right corresponds to the complement of the written class. Blue arrows denote containment, while black arrows denote symmetric containment.

CHAPTER 4

Complexity Theory Results

What follows is a survey of complexity theory that has been aided by the Complexity Zoology software. It is similar in structure to Scott Aaronson's Complexity Zoo, although with a much smaller collection of complexity classes. The role of the software in producing this software has been threefold: it has identified unanswered questions most useful to creating a complete picture of the field, it has highlighted redundant information that is the logical consequence of other data, and it has provided a high-level view of the current state of knowledge.

Where possible, redundant data has been removed from the survey, although it is occasionally left in when it represents a particularly notable or foundational result. Results are explained in varying levels of detail with the intent of both providing a roadmap of complexity theory and illustrating some key arguments and frequently used techniques. It is hoped that this overview demonstrates the usefulness of a computer-assisted approach to compiling knowledge about the relationships between complexity classes.

4.1. Simplified Version

We begin by considering a simplified version of our survey. In this version, there are just 17 complexity classes, and we will mostly examine their relationships in the world of all oracles.

4.1.1. Complexity Classes.

The classes in this mini-survey are listed here in alphabetical order, with definitions for each.

- ALL: The class of all languages. Naturally, $\mathsf{ALL}^f = \{\mathscr{L} : \mathscr{L} \subseteq \Sigma^*\}$ for every oracle f.
- AM: The class of languages computable by the *Arthur-Merlin protocol*. Merlin is a *prover* who wants to convince the *verifier*, Arthur, that an input x lies in the language $\mathscr{L}$. Merlin knows at the outset whether $x \in \mathscr{L}$ and can make any argument, but he is also biased: he wishes to convince

Arthur that $x \in \mathscr{L}$ regardless of whether this is actually true. Arthur, meanwhile, is a polynomial-time classical computer. Arthur may use randomness in his calculations, but the results of his coin tosses are known to Merlin in advance.

A language $\mathscr{L}$ is said to be in AM if, when $x \in \mathscr{L}$, Merlin can convince Arthur of this fact with probability $\geq 2/3$, while if $x \notin \mathscr{L}$, then Merlin cannot convince Arthur that $x \in \mathscr{L}$ with a probability of more than $1/3$. Formally, $\mathscr{L} \in \mathsf{AM}^f$ if and only if there exists a polynomial-time Turing machine M with f-oracle and functions $p(n), q(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \Longrightarrow \Pr_{y \in \Sigma^{p(|x|)}} [(\exists z \in \Sigma^{q(|x|)})[M(x,y,z) = 1]] \geq 2/3,$$

$$x \notin \mathscr{L} \Longrightarrow \Pr_{y \in \Sigma^{p(|x|)}} [(\exists z \in \Sigma^{q(|x|)})[M(x,y,z) = 1]] \leq 1/3.$$

- BPP: The class of languages computable in polynomial time, with randomness. We can model BPP with a special probabilistic Turing machine capable of making coin tosses as part of its computation. Alternatively, we can make coin tosses in advance and supply the result to a deterministic Turing machine as an ancillary string along with the input. Formally, $\mathscr{L} \in \mathsf{BPP}^f$ if and only if there exists a polynomial-time Turing machine M with f-oracle and a function $p(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$\Pr_{y \in \Sigma^{p(|x|)}} [M(x,y) = \mathscr{L}(x)] \geq 2/3.$$

- BQP: The class of languages computable in polynomial time by a quantum computer. Formally, this means that languages in BQP are computable by a polynomially sized family of quantum circuits. As discussed in Subsection 2.1.4, this family must be uniform.
- $\mathsf{BQP/qpoly}$: BQP with quantum polynomial advice. This class consists of languages that can be computed by a polynomial-time quantum computer with polynomial-length *quantum advice*. Quantum advice is a string of qubits that depends only on the length of the input; the qubits are allowed to be in a state of superposition.
- EXP: The class of languages computable in exponential-time. Formally, $\mathscr{L} \in \mathsf{EXP}^f$ if there exists an oracle Turing machine M with f-oracle that computes $\mathscr{L}$ in $T(n)$-time with $T(n) = O(2^{p(n)})$, where $p(n) = O(n^*)$.

- MA: The class of languages computable using the *Merlin-Arthur protocol*. This is identical to the Arthur-Merlin protocol, except Arthur's coin tosses are unknown to Merlin. Formally, $\mathscr{L} \in \mathsf{MA}^f$ if there exists a polynomial-time Turing machine M with f-oracle and functions $p(n), q(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \implies (\exists z \in \Sigma^{q(|x|)}) \left[\Pr_{y \in \Sigma^{p(|x|)}} [M(x,y,z) = 1] \geq 2/3 \right],$$
$$x \notin \mathscr{L} \implies (\forall z \in \Sigma^{q(|x|)}) \left[\Pr_{y \in \Sigma^{p(|x|)}} [M(x,y,z) = 1] \leq 1/3 \right].$$

- NP: The class of languages that can be computed by a non-deterministic algorithm in polynomial-time. We replace the usual notion of a Turing machine with that of a non-deterministic Turing machine with two transition functions. Thus, instead of the computational process consisting of a sequence of configurations, it consists of a tree of configurations. Then $\mathscr{L} \in \mathsf{NP}$ precisely when if $x \in \mathscr{L}$, there is a path in the tree that accepts the input, while if $x \notin \mathscr{L}$, all paths reject. NP can also be defined using deterministic Turing machines with *certificates*: formally, $\mathscr{L} \in \mathsf{NP}^f$ if and only if there exist a polynomial-time oracle Turing machine M with f-oracle and a function $p(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \iff (\exists y \in \Sigma^{p(|x|)})[M(x,y) = 1].$$

- P: The class of languages that can be computed by a polynomial-time Turing machine (or an oracle Turing machine with f-oracle in the case of P^f). A polynomial-time Turing machine is one that, on an input of length n, terminates within $T(n)$ steps, where $T(n) = O(n^*)$.
- $\mathsf{P}^{\#\mathsf{P}}$: P with $\#\mathsf{P}$-oracle. This class consists of languages computable in polynomial time with an oracle f that lies in the function class $\#\mathsf{P}$. For a pair (M, p) consisting of a polynomial-time Turing machine M and a function $p(n) = O(n^*)$, set

$$Y_{(M,p)} = \{y \in \Sigma^{p(|x|)} : M(x,y) = 1\}.$$

Then, $g : \Sigma^* \to \Sigma^*$ lies in $\#\mathsf{P}^f$ if and only if there exists a pair (M, p) such that $g(x) = |Y_{(M,p)}|$ for every $x \in \Sigma^*$. Now define $(\mathsf{P}^{\#\mathsf{P}})^f = \bigcup_{g \in \#\mathsf{P}^f} \mathsf{P}^g$.

- PH: The polynomial hierarchy. For an oracle f, define $\Sigma_0\mathsf{P}^f = \mathsf{P}^f$. Then, for each $j \in \mathbb{N}$, set $\Sigma_{j+1}\mathsf{P}^f = \mathsf{NP}^{\Sigma_j\mathsf{P}^f}$. Finally, define $\mathsf{PH} = \bigcup_{j=0}^{\infty}\Sigma_j\mathsf{P}^f$. We also define $\Pi_j\mathsf{P}^f$ and $\Delta_j\mathsf{P}^f$ by $\Sigma_0\mathsf{P}^f = \Delta_0\mathsf{P}^f = \mathsf{P}^f$ and $\Pi_j\mathsf{P}^f = \mathsf{coNP}^{\Sigma_{j-1}\mathsf{P}^f}$, $\Delta_j\mathsf{P}^f = \mathsf{P}^{\Sigma_{j-1}\mathsf{P}^f}$ for $j \geq 1$.
- PP: Like BPP, a class of polynomial-time computable languages, with randomness. The defining condition of PP is weaker than that of BPP, requiring only that if an input x lies in the language, a probabilistic Turing machine obtains the correct answer with a probability of at least $1/2$. Formally, $\mathscr{L} \in \mathsf{PP}^f$ if and only if there exists a polynomial-time Turing machine M with f-oracle and a function $p(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \Longleftrightarrow \Pr_{y\in\Sigma^{p(|x|)}}[M(x,y) = 1] \geq 1/2.$$

- $\mathsf{P/poly}$: P with polynomial advice. An *advice string* is a fixed string y_n that accompanies an input of length n. Being *polynomial* advice means that the function $n \mapsto |y_n|$ is $O(n^*)$. Formally, $\mathscr{L} \in (\mathsf{P/poly})^f$ if and only if there exist a polynomial-time oracle Turing machine M with f-oracle and a function $g : \mathbb{N} \mapsto \Sigma^*$, where $|g(n)| = O(n^*)$, such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \Longleftrightarrow M(x, g(|x|)) = 1.$$

  Equivalently, $\mathsf{P/poly}$ is the class of languages that can be computed by a non-uniform family of polynomial-size classical circuits (see Subsection 2.1.3).
- PSPACE: The class of languages that are polynomial-space computable. For a function $T : \mathbb{N} \to \mathbb{N}$, $\mathscr{L} \in \mathsf{SPACE}(T(n))^f$ if and only if there exist an oracle Turing machine M with f-oracle that computes $\mathscr{L}$ and a constant C such that, when $x \in \Sigma^*$ is given as an input to the machine, there are at most $CT(|x|)$ cells of each tape that are ever written onto during the computation. Then, define $\mathsf{PSPACE}^f = \bigcup_{k=0}^{\infty}\mathsf{SPACE}(n^k)^f$. We adopt the convention that the space limitation of PSPACE^f applies to the oracle tape as well, so that oracle calls are limited to a polynomial of the length of the input.
- QAM: The class of languages using the quantum Arthur-Merlin protocol. The definition is similar to AM^f, except M is a polynomial-time quantum computer with f-oracle rather than a classical computer, and Merlin is an all-powerful quantum-computer capable of computing any system of

qubits. Arthur sends Merlin a random string of classical bits, Merlin responds with a polynomial-length quantum message, and then Arthur uses the random bits along with Merlin's message to decide whether to accept or reject the input.

- QMA: The class of languages using the quantum Merlin-Arthur protocol. The definition is the same as MA^f, except M is a polynomial-time quantum computer with f-oracle rather than a classical computer, and Merlin is an all-powerful quantum-computer. Arthur and Merlin exchange messages in the form of systems of qubits rather than classical bit strings.
- SZK: Traditionally defined as the class of languages that can be computed using a statistical zero-knowledge proof protocol, SZK can be defined in a simpler way using a special case of this protocol. We call the special case the *rhetorical question protocol*. This is similar to the Arthur-Merlin protocol, except that Arthur's message to Merlin consists of a question for which Arthur has privately computed a correct answer. Arthur accepts the input if and only if Merlin's response matches Arthur's correct answer. Formally, $\mathscr{L} \in \mathsf{SZK}^f$ if and only if there exist a polynomial-time oracle Turing machine M with f-oracle and a function $p(n) = O(n^*)$ such that for all $x \in \Sigma^*$,

$$x \in \mathscr{L} \Longrightarrow (\exists P \in \mathscr{M}) \left[\Pr_{y \in \Sigma^{p(|x|)}} [P(x, M(x,y,0)) = M(x,y,1)] \geq 2/3 \right],$$
$$x \notin \mathscr{L} \Longrightarrow (\forall P \in \mathscr{M}) \left[\Pr_{y \in \Sigma^{p(|x|)}} [P(x, M(x,y,0)) = M(x,y,1)] \leq 1/3 \right],$$

where

$$\mathscr{M} = \{P : \Sigma^* \to \Sigma^* : (\exists q(n) = O(n^*))[P(x) \in \Sigma^{q(|x|)}]\}.$$

See Subsection 4.2.79 for a discussion of the relationship between this definition of SZK and a more conventional definition.

4.1.2. Inclusions. To establish the overall structure of the relationships between these 17 complexity classes, we first establish which inclusions are relativizing – i.e., which inclusions hold for every oracle. To that end, we identify the symmetric classes, by which we mean the classes C such that $\mathsf{C} = co \cdot \mathsf{C}$ with respect to every oracle. This will allow us to easily strengthen many of the inclusions described here. For example, knowing that P is symmetric and that $\mathsf{P} \subseteq \mathsf{NP}$ for every oracle allows us to conclude that $\mathsf{P} \subseteq cocap \cdot \mathsf{NP} = \mathsf{NP} \cap \mathsf{coNP}$ with respect to every oracle.

It is immediate that P is symmetric, because a polynomial-time computer can simply swap its output α with $1-\alpha$. The computer can still perform this operation if it is quantum, exponential-time, polynomial-space, or has access to a #P-oracle, so BQP, EXP, PSPACE, and $\mathsf{P}^{\#\mathsf{P}}$ are likewise symmetric. A computer receiving advice strings does not affect this capacity, so $\mathsf{P/poly}$ and $\mathsf{BQP/poly}$ are symmetric as well. As was shown in Section 2.3, the operators $\mathcal{BP}$ and $\mathcal{P}$ commute with co. Since $\mathsf{BPP} = \mathcal{BP}\cdot\mathsf{P}$ and $\mathsf{PP} = \mathcal{P}\cdot\mathsf{P}$, it follows that BPP and PP are symmetric. Of course, ALL is clearly symmetric.

PH is symmetric, because $\mathsf{PH} = \bigcup_{k=0}^{\infty}\Sigma_k\mathsf{P} = \bigcup_{k=0}^{\infty}\Pi_k\mathsf{P}$. More explicitly:

LEMMA 4.1.1. *For every $k\in\mathbb{N}$, $\Delta_k\mathsf{P}\subseteq\Sigma_k\mathsf{P}$, $\Delta_k\mathsf{P}\subseteq\Pi_k\mathsf{P}$, $\Sigma_k\mathsf{P}\subseteq\Delta_{k+1}\mathsf{P}$, and $\Pi_k\mathsf{P}\subseteq\Delta_{k+1}\mathsf{P}$.*

PROOF. In the case that $k=0$, $\Delta_k\mathsf{P}\subseteq\Sigma_k\mathsf{P}$ because both classes are equal to P. If $k>0$, then $\Delta_k\mathsf{P}\subseteq\mathsf{P}^{\Sigma_{k-1}\mathsf{P}}\subseteq\mathsf{NP}^{\Sigma_{k-1}\mathsf{P}}=\Sigma_k\mathsf{P}$. Since $\Delta_k\mathsf{P}$ is a symmetric class (being P with an oracle), it follows that $\Delta_k\mathsf{P}\subseteq\Pi_k\mathsf{P}$ for every k.

For the second pair of inclusions, $\Delta_k\mathsf{P}\subseteq\mathsf{P}^{\Sigma_k\mathsf{P}}=\Delta_{k+1}\mathsf{P}$, and then $\Pi_k\mathsf{P}\subseteq\Delta_{k+1}\mathsf{P}$ likewise follows from the symmetry of $\Delta_{k+1}\mathsf{P}$. $\square$

SZK is a symmetric class:

THEOREM 4.1.1 (Okamoto, [**Oka00**]). $\mathsf{SZK} = co\cdot\mathsf{SZK}$ *with respect to any oracle.*

Next we discuss which complexity classes are subsets of each other relative to every oracle. We consider a minimal generating set of inclusions, which in this context means that all the inclusions will be Hasse relative to the 17 classes we are considering. In other words, we do not need to prove $\mathsf{P}\subseteq\mathsf{MA}$ because this follows from $\mathsf{P}\subseteq\mathsf{NP}$ and $\mathsf{NP}\subseteq\mathsf{MA}$.

$\mathsf{P}\subseteq\mathsf{BPP}$ and $\mathsf{P}\subseteq\mathsf{NP}$ are true because both BPP and NP are polynomial-time classes, but with additional powers that can be ignored: to simulate P with BPP, make no coin tosses; to simulate P with NP, discard the certificate. Alternatively both inclusions can be said to follow from the properties in Section 2.3, since $\mathsf{BPP}=\mathcal{BP}\cdot\mathsf{P}$ and $\mathsf{NP}=\mathcal{N}\cdot\mathsf{P}$. Similarly, $\mathsf{BQP}\subseteq\mathsf{BQP/qpoly}$.

As was mentioned during the discussion on the pseudo-operator $\mathcal{Q}$ in Section 2.3, a quantum computer can perform the same operations as a classical computer (possibly with the aid of ancillary qubits). This applies to probabilistic classic computers as well as deterministic ones, because a coin flip can be simulated by measuring the qubit $\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)$. Hence $\mathsf{BPP}\subseteq\mathsf{BQP}$. By the same principle, $\mathsf{P/poly}\subseteq\mathsf{BQP/qpoly}$, $\mathsf{MA}\subseteq\mathsf{QMA}$, and $\mathsf{AM}\subseteq\mathsf{QAM}$.

The inclusion $\mathsf{BPP} \subseteq \mathsf{MA}$ is straightforward: in the definition of MA, the verifier is a probabilistic polynomial-time Turing machine that must reach the correct answer with a probability of at least $2/3$. Thus, a protocol in which Arthur ignores Merlin and performs his own computations is an an MA-protocol that computes a problem in BPP. The same argument also shows that $\mathsf{BQP} \subseteq \mathsf{QMA}$. Similarly, $\mathsf{BPP} \subseteq \mathsf{SZK}$. While our definition of SZK requires that Arthur bases his conclusion on the response given by Merlin, Arthur can effectively "ignore" Merlin by either telling Merlin the correct answer (if Arthur wants to accept the input) or generating a random string as the correct answer and sending a blank message to Merlin (if Arthur wants to reject the input).

BPP can be *derandomized*, allowing it to be placed inside P/poly.

THEOREM 4.1.2 (Adleman, [**Adl78**]). $\mathsf{BPP} \subseteq \mathsf{P}/\mathsf{poly}$ *relative to every oracle.*

PROOF. Let $\mathscr{L} \in \mathsf{BPP}$. The class BPP is subject to an error-reducing procedure: by running several copies of a BPP-machine in parallel and taking the "majority vote" of the machines, we can obtain the correct answer with a higher probability. Using this technique, we can suppose that there exists a polynomial-time Turing machine M and a function $p(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$\Pr_{y \in p(|x|)}[M(x,y) \neq \mathscr{L}(x)] \leq 1/2^{|x|+1}.$$

Equivalently, for every $x \in \Sigma^*$ there are at most $2^{p(|x|)-|x|-1}$ strings $y \in \Sigma^{p(|x|)}$ such that $M(x,y) \neq \mathscr{L}(x)$. For $x \in \Sigma^*$, set

$$B_x = \{y \in \Sigma^{p(|x|)} : M(x,y) \neq \mathscr{L}(x)\},$$

and for $n \in \mathbb{N}$, set

$$B_n = \bigcup_{x \in \Sigma^n} B_x.$$

B_n is the set of "bad advice" for an input of length n. Now

$$|B_n| \leq \Sigma_{|x|=n} |B_x| \leq \Sigma_{|x|=n} 2^{p(n)-n-1} = 2^n 2^{p(n)-n-1} = 2^{p(n)-1} < 2^{p(n)} = |\Sigma^{p(n)}|,$$

so the set $\Sigma^{p(n)} \setminus B_n$ is nonempty for every n. Thus, there exists a function $f : \mathbb{N} \to \Sigma^*$ such that $f(n) \in \Sigma^{p(n)} \setminus B_n$ for every n. We therefore have that for each $x \in \Sigma^*$,

$$M(x, f(|x|)) = \mathscr{L}(x),$$

so $\mathscr{L} \in \mathsf{P/poly}$ as desired. □

The fact $\mathsf{NP} \subseteq \mathsf{MA}$ follows from the observation that replacing the definition of *MA* with a deterministic Turing machine is precisely the definition of NP.

We now consider the relationship between AM and MA. To do this, we introduce related (and seemingly different) classes $\mathsf{AM}[k]$ for integers $k \geq 2$. This represents the Arthur-Merlin protocol with k rounds.

DEFINITION 4.1.1. A k-round Arthur-Merlin protocol is an interaction $(V,P)_k$ between a probabilistic Turing machine V, representing Arthur, and a function $P : \Sigma^* \to \Sigma^*$, representing Merlin. Given an input $x \in \Sigma^*$, Arthur computes a message m_1 and sends it, along with the random bits y_1 used in the computation, as a string $\alpha_1 = \langle m_1, y_1 \rangle$ to Merlin. Merlin responds with a string α_2 that is limited in length by a polynomial in $|x|$. Arthur sends a new message $\alpha_3 = \langle m_3, y_3 \rangle$, Merlin responds with α_4, and so on, until the sequence $(\alpha_1, \ldots, \alpha_k)$ has been generated. Then Arthur deterministically computes $V(\alpha_1, \ldots, \alpha_k)$ to decide whether to accept the input. We denote the result by $\mathrm{out}_{(V,P)_k}(x)$.

A language $\mathscr{L}$ lies in $\mathsf{AM}[k]$ if and only if there exists a polynomial-time Turing machine V such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \Longrightarrow (\exists P \in \mathscr{M}) \Pr[\mathrm{out}_{(V,P)_k}(x) = 1] \geq 2/3,$$

$$x \notin \mathscr{L} \Longrightarrow (\forall P \in \mathscr{M}) \Pr[\mathrm{out}_{(V,P)_k}(x) = 1] \leq 1/3,$$

where

$$\mathscr{M} = \{P : \Sigma^* \to \Sigma^* : (\exists q(n) = O(n^*))[P(x) \in \Sigma^{q(|x|)}]\}.$$

With this definition, we obtain $\mathsf{AM} = \mathsf{AM}[2]$. Moreover, Arthur can imitate the Merlin-Arthur protocol within a 3-round Arthur-Merlin protocol: after sending his message to Merlin and receiving a response, Arthur makes his final computation with a fresh set of random bits. These bits are sent to Merlin, but since Merlin cannot send a second response, these random bits are effectively private. Hence $\mathsf{MA} \subseteq \mathsf{AM}[3]$.

As it turns out, however, adding extra rounds to an Arthur-Merlin protocol does not change the computational power of AM:

THEOREM 4.1.3 (Babai, [**Bab85**]). $\mathsf{AM}[k+1] \subseteq \mathsf{AM}[k]$ *with respect to any oracle for all integers* $k \geq 2$.

$\mathsf{AM}[2] \subseteq \mathsf{AM}[3] \subseteq \mathsf{AM}[4] \subseteq \ldots$, since Arthur can choose to ignore the later rounds of his interaction with Merlin. Hence $\mathsf{AM} = \mathsf{AM}[k]$ for every k.

As an immediate corollary of $\mathsf{MA} \subseteq \mathsf{AM}[3]$ and the above theorem, we deduce $\mathsf{MA} \subseteq \mathsf{AM}$. These arguments are not affected if Arthur is quantum rather than classical, so we also have $\mathsf{QMA} \subseteq \mathsf{QAM}$. Okamoto [**Oka00**] showed that SZK could be characterized by a constant-round protocol with a public-coin Arthur; it follows that $\mathsf{SZK} \subseteq \mathsf{AM}$.

To see that $\mathsf{AM} \subseteq \mathsf{PH}$, we need an alternate characterization of PH:

LEMMA 4.1.2. *For every language $\mathscr{L} \subseteq \Sigma^*$, $\mathscr{L} \in \Sigma_k\mathsf{P}$ if and only if there exists a polynomial-time Turing machine M and $p_1(n), \ldots, p_k(n) = O(n^*)$ such that for all $x \in \Sigma^*$,*

$$x \in \mathscr{L} \iff (Q_1 y_1 \in p_1(|x|)) \ldots (Q_k y_k \in p_k(|x|))[M(x, y_1, \ldots, y_k) = 1],$$

where Q_j is $\exists$ when j is odd and $\forall$ when j is even.

PROOF. Denote this alternative version of $\Sigma_k\mathsf{P}$ by $\bar{\Sigma}_k\mathsf{P}$, and denote $\bar{\Pi}_n\mathsf{P} = co \cdot \bar{\Sigma}_k\mathsf{P}$. If $k = 0$, then $\Sigma_0\mathsf{P} = \bar{\Sigma}_0\mathsf{P} = \mathsf{P}$. Thus, we can suppose for induction that $\Sigma_{k+1}\mathsf{P} = \bar{\Sigma}_{k+1}\mathsf{P}$. We have

$$\bar{\Sigma}_{k+1}\mathsf{P} = \mathcal{N} \cdot \bar{\Pi}_k\mathsf{P} = \mathcal{N} \cdot \Pi_k\mathsf{P} \subseteq \mathcal{N} \cdot \mathsf{P}^{\Pi_k\mathsf{P}} = \mathcal{N} \cdot \mathsf{P}^{\Sigma_k\mathsf{P}} = \mathsf{NP}^{\Sigma_k\mathsf{P}} = \Sigma_{k+1}\mathsf{P},$$

so it suffices to show that $\Sigma_{k+1}\mathsf{P} \subseteq \bar{\Sigma}_{k+1}\mathsf{P}$.

Suppose $\mathscr{L} \in \Sigma_{k+1}\mathsf{P}$. Then there exists a polynomial-time Turing machine M with f-oracle, where $f \in \Sigma_k\mathsf{P}$, and a function $p(n) = O(n^*)$ such that

$$x \in \mathscr{L} \iff (\exists y \in \Sigma^{p(|x|)})[M(x,y) = 1] \tag{4.1}$$

for every $x \in \Sigma^*$. On an input $\langle x, y \rangle$, the machine makes $q(|x|)$ oracle calls, where $q(n) = O(n^*)$. Let M' be the polynomial-time Turing machine such that $M'(x, y, a_1, \ldots, a_{q(|x|)})$ computes $M(x,y)$ when $a_1, \ldots, a_{q(|x|)}$ are given as oracle replies. Then (4.1) can be written as

$$x \in \mathscr{L} \iff (\exists y \in \Sigma^{p(|x|)}, a_1, \ldots, a_{q(|x|)} \in \Sigma^{q(|x|)})[M'(x, y, a_1, \ldots, a_{q(|x|)})\ \&\ a_1, \ldots, a_{q(|x|)} \text{ are correct}]. \tag{4.2}$$

"Correct" means that, if M queries the oracle with z, the oracle responds with $f(z)$. It is enough to show that "$a_1, \ldots, a_{q(|x|)}$ are correct" is a $\bar{\Sigma}_{k+1}\mathsf{P}$ predicate, because then the entire right side of (4.1.2) is a $\bar{\Sigma}_{k+1}\mathsf{P}$

predicate. To see that this is the case, note that given x, y, and $a_1, \ldots, a_{q(|x|)}$, we can complete the queries $z_1, \ldots, z_{q(|x|)}$ given to the oracle in polynomial time (let M' run, and then record the oracle queries as they occur). Then "$a_1, \ldots, a_{q(|x|)}$ are correct" is equivalent to

$$\bigwedge_{j=1}^{q(|x|)} [A(z_j) = a_j],$$

which is a conjunction of $O(|x|^*)$-many $\bar{\Sigma}_{k+1}\mathsf{P}$ predicates (specifically, $(z_j) = 1$ is a $\bar{\Sigma}_k$ predicate, and $f(z_j) = 0$ is a $\bar{\Pi}_k$ predicate) and therefore a $\bar{\Sigma}_{k+1}\mathsf{P}$ predicate itself. Hence $\mathscr{L} \in \bar{\Sigma}_{k+1}\mathsf{P}$ as desired. $\square$

We also need the following result:

THEOREM 4.1.4 (Sipser-Gàcs-Lautemann). *If $\mathscr{L} \in \mathcal{BP} \cdot \mathsf{C}$ for a complexity class C, there exist $\mathscr{L}' \in \mathsf{C}$ and functions $p(n), q(n) = O(n^*)$ such that*

$$x \in \mathscr{L} \iff (\exists u_1, \ldots, u_{p(|x|)} \in \Sigma^{q(|x|)})(\forall r \in \Sigma^{q(|x|)}) \left[\bigvee_{j=1}^{p(|x|)} \langle x, r +_2 u_j \rangle \in \mathscr{L}' \right],$$

where $+_2$ denotes entrywise addition modulo 2.

PROOF. Let $\mathscr{L} \in \mathcal{BP} \cdot \mathsf{C}$. By the same error-reduction procedure used in the proof of $\mathsf{BPP} \subseteq \mathsf{P/poly}$, there exists a language $\mathscr{L}' \in \mathsf{C}$ and a function $p(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \implies \Pr_{y \in \Sigma^{p(|x|)}} [\langle x, y \rangle \in \mathscr{L}'] \geq 1 - 2^{-|x|}, \tag{4.3}$$

$$x \notin \mathscr{L} \implies \Pr_{y \in \Sigma^{p(|x|)}} [\langle x, y \rangle \in \mathscr{L}'] \leq 2^{-|x|}. \tag{4.4}$$

For each $x \in \Sigma^*$, set

$$A_x = \{y \in \Sigma^{p(|x|)} : \langle x, y \rangle \in \mathscr{L}'\}.$$

Then (4.3) and (4.4) can be rewritten as

$$x \in \mathscr{L} \implies |A_x| \geq 2^{p(|x|)}(1 - 2^{-|x|}), \tag{4.5}$$

$$x \notin \mathscr{L} \implies |A_x| \leq 2^{p(|x|)-|x|}. \tag{4.6}$$

Now for $m \in \mathbb{N}$, $S \subseteq \Sigma^m$, and $u \in \Sigma^m$, set

$$S +_2 u = \{y +_2 u : y \in S\}.$$

Note that for any $n \in \mathbb{N}$, if $S \subseteq \Sigma^{p(n)}$ satisfies $|S| \leq 2^{p(n)-n}$, then for any $u_1, \ldots, u_k \in \Sigma^{p(n)}$ with $k < 2^n$, we have

$$\left|\bigcup_{j=1}^{k} (S +_2 u_j)\right| \leq \sum_{j=1}^{k} |S +_2 u_j| = k|S| \leq k2^{p(n)-n} < 2^{p(n)},$$

so that $\bigcup_{j=1}^{k}(S +_2 u_j) \neq \Sigma^{p(|x|)}$. In particular, if we set $q(n) = \lceil p(n)/n \rceil + 1$ for every $n \in \mathbb{N}$ (we will need the fact that $p(n) < nq(n)$ shortly), then it follows from (4.6) that

$$x \notin \mathscr{L} \Longrightarrow (\forall u_1, \ldots, u_{q(|x|)} \in \Sigma^{p(|x|)}) \left[\bigcup_{j=1}^{q(|x|)} (A_x +_2 u_j) \neq \Sigma^{p(|x|)}\right]. \tag{4.7}$$

On the other hand, suppose that $S \subseteq \Sigma^{p(|x|)}$ satisfies $|S| \geq 2^{p(n)}(1 - 2^{-n})$. Let $u_1, \ldots, u_{q(n)} \in \Sigma^{p(n)}$ be chosen uniformly randomly. Then $r +_2 u_j$ is also randomly chosen for any particular r, and we can compute as follows:

$$\begin{aligned}
\Pr\left[\bigcup_{j=1}^{q(n)} (S +_2 u_j) \neq \Sigma^{p(n)}\right] &= \Pr\left[(\exists r \in \Sigma^{p(n)}) \left[r \notin \bigcup_{j=1}^{q(n)} (S +_2 u_j)\right]\right] \\
&= \Pr\left[(\exists r \in \Sigma^{p(n)}) \left[\bigvee_{j=1}^{q(n)} r \notin S +_2 u_j\right]\right] \\
&= \Pr\left[(\exists r \in \Sigma^{p(n)}) \left[\bigvee_{j=1}^{q(n)} r +_2 u_j \notin S\right]\right] \\
&= \Pr\left[\bigvee_{j=1}^{q(n)} u_j \notin S\right] = \prod_{j=1}^{q(n)} \Pr[u_j \notin S] \leq \prod_{j=1}^{q(n)} 2^{-n} = 2^{-nq(n)} < 2^{-p(n)} < 1.
\end{aligned}$$

Hence $\Pr[\bigcup_{j=1}^{q(n)} (S +_2 u_j) = \Sigma^{p(n)}] > 0$, and so it follows from (4.5) that

$$x \in \mathscr{L} \Longrightarrow (\exists u_1, \ldots, u_{q(|x|)} \in \Sigma^{p(|x|)}) \left[\bigcup_{j=1}^{q(|x|)} (A_x + u_j) = \Sigma^{p(|x|)}\right]. \tag{4.8}$$

(4.7) and (4.8) together are equivalent to the desired result. □

COROLLARY 4.1.1. $\mathsf{AM} \subseteq \Pi_2\mathsf{P}$ *with respect to every oracle.*

PROOF. The definition of AM gives $\mathsf{AM} = \mathcal{BP} \cdot \mathsf{NP}$ and hence $co \cdot \mathsf{AM} = \mathcal{BP} \cdot \mathsf{coNP}$. Thus, by the previous theorem, for $\mathscr{L} \in co \cdot \mathsf{AM}$ there exist $\mathscr{L}' \in \mathsf{coNP}$ and $p(n), q(n) = O(n^*)$ such that

$$x \in \mathscr{L} \iff (\exists u_1, \dots, u_{p(|x|)} \in \Sigma^{q(|x|)})(\forall r \in \Sigma^{q(|x|)}) \left[\bigvee_{j=1}^{p(|x|)} \langle x, r +_2 u_j \rangle \in \mathscr{L}' \right]$$

for every $x \in \Sigma^*$. It follows that $\mathscr{L} \in \Sigma_2\mathsf{P}$ by the quantifier definition of $\Sigma_2\mathsf{P}$. Therefore, $co \cdot \mathsf{AM} \subseteq \Sigma_2\mathsf{P}$. $\square$

We next have $\mathsf{QMA} \subseteq \mathsf{PP}$ [**Vya03**] with respect to every oracle. In fact, $\mathsf{QMA} \subseteq \mathsf{A_0PP}$, where $\mathsf{A_0PP} \subseteq \mathsf{PP}$ is a class defined as follows: for $\mathscr{L} \subseteq \Sigma^*$, we say $\mathscr{L} \in \mathsf{A_0PP}$ if and only if there exist functions $f, g : \Sigma^* \to \Sigma^*$, where g is polynomial-time computable and

$$f(x) = |\{y \in \Sigma^{p(|x|)} : M(x,y) = 1\}| - |\{y \in \Sigma^{p(|x|)} : M(x,y) = 0\}|$$

for a polynomial-time Turing machine M and $p(n) = O(n^*)$, such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \implies f(x) > g(x),$$
$$x \notin \mathscr{L} \implies f(x) < g(x)/2.$$

We now consider the class $\mathsf{P}^{\#\mathsf{P}}$. #P can be considered to be the function class analogue of PP in the same way that FP, the class of polynomial-time computable functions, is the function class analogue of P. In general, including a P-oracle in a computational process is equivalent to including an FP-oracle, because with a P-oracle one can determine the output $f(x)$ of a function $f \in \mathsf{FP}$ by successively computing each bit of $f(x)$. Similarly, a PP-oracle can be used to determine each bit of the output of a function in #P, thereby simulating a #P-oracle. Hence $\mathsf{P}^{\#\mathsf{P}} = \mathsf{P}^{\mathsf{PP}}$. It follows that $\mathsf{PP} \subseteq \mathsf{P}^{\#\mathsf{P}}$ (this only requires the straightforward direction $\mathsf{P}^{\mathsf{PP}} \subseteq \mathsf{P}^{\#\mathsf{P}}$).

$\mathsf{P}^{\#\mathsf{P}}$ also contains the polynomial hierarchy:

THEOREM 4.1.5 (Toda, [**Tod91**]). $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$ *relative to every oracle.*

Another inclusion that makes use of an intermediate class is $\mathsf{QAM} \subseteq \mathsf{PSPACE}$.

THEOREM 4.1.6 ([**MW05**]). $\mathsf{QAM} \subseteq \mathcal{BP} \cdot \mathsf{PP}$ *relative to every oracle.*

Now $\mathcal{BP} \cdot \mathsf{PP} \subseteq \mathsf{PSPACE}$.

LEMMA 4.1.3. $\mathsf{PP} \subseteq \mathsf{PSPACE}$.

PROOF. Since PSPACE is not limited in computational time, given a polynomial-time Turing machine M and a function $p(n) = O(n^*)$, a PSPACE-machine can simulate a PP-machine by keeping a tally of how many $y \in \Sigma^{p(|x|)}$ satisfy $M(x,y) = 1$ for the input x. □

This derandomization also applies to the probabilistic version of PSPACE.

LEMMA 4.1.4. $\mathcal{BP} \cdot \mathsf{PSPACE} \subseteq \mathsf{PSPACE}$.

It now follows that

$$\mathsf{QAM} \subseteq \mathcal{BP} \cdot \mathsf{PP} \subseteq \mathcal{BP} \cdot \mathsf{PSPACE} \subseteq \mathsf{PSPACE}.$$

With one additional lemma, we can also prove that $\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{PSPACE}$.

LEMMA 4.1.5. *If* $\mathscr{L} \in \mathsf{PSPACE}$*, then* $\mathsf{PSPACE}^{\mathscr{L}} \subseteq \mathsf{PSPACE}$.

PROOF. By our convention, oracle calls are limited in length to a polynomial of the imput. Hence any calls to the oracle in a $\mathsf{PSPACE}^{\mathscr{L}}$-machine can be replaced with a direct computation of $\mathscr{L}(x)$ by a PSPACE-machine. □

Now

$$\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{PP}} \subseteq \mathsf{PSPACE}^{\mathsf{PSPACE}} \subseteq \mathsf{PSPACE}.$$

For our final PSPACE inclusion, observe that a PSPACE-machine is limited to $2^{p(n)}$ possible configurations for an input of length n, where $p(n) = O(n^*)$. Thus, any computation that halts must do so within $2^{p(n)}$ steps. Therefore $\mathsf{PSPACE} \subseteq \mathsf{EXP}$.

The results of this subsection are summarized in Figure 4.5.

4.1.3. Oracle Separations and Collapses. Now we present some of the inclusions that fail for some oracle, as well as some inclusions that do not relativize. As far as relativizing inclusions are concerned, the picture presented in Figure 4.5 is complete for the 17 complexity classes shown. We cannot, for instance, show that $\mathsf{P/poly}$ is contained in BQP or that EXP is contained in P for every oracle. These particular examples hold for every oracle: in the former case because BQP is countable while $\mathsf{P/poly}$ is not, and in the latter case because of the *time-hierarchy theorem*.

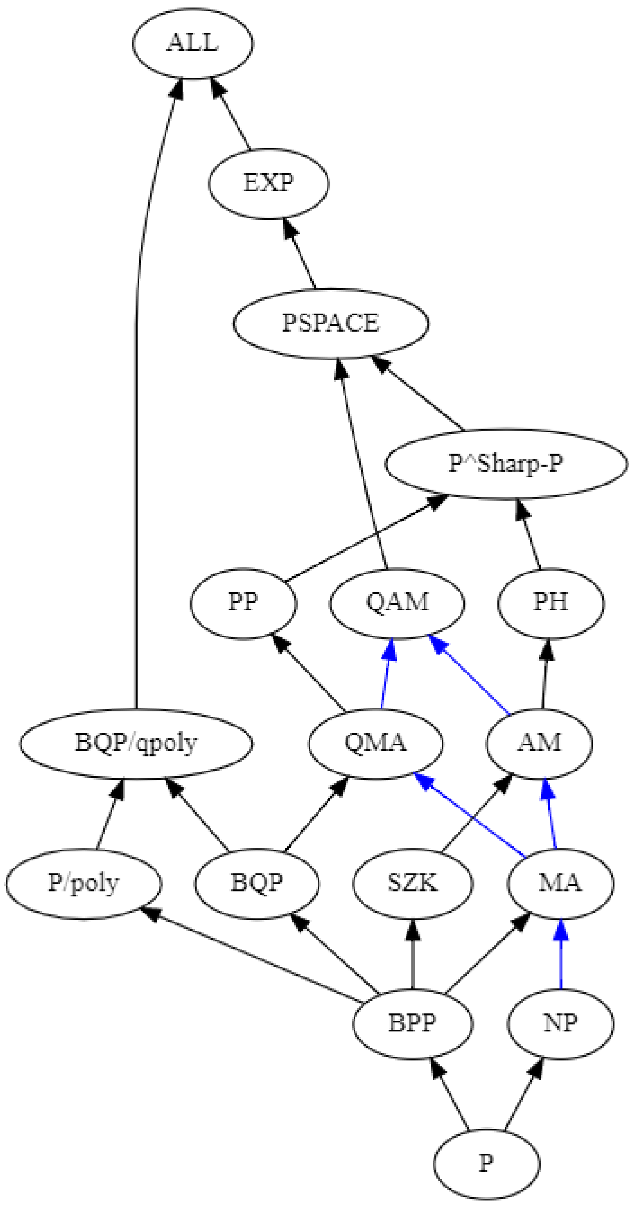

FIGURE 4.1. Inclusions that hold with respect to every oracle for 17 complexity classes. Blue indicates that the lower class is included in the upper class. Black indicates that the lower class and its complement are included in the upper class.

For functions $f, g : \mathbb{N} \to \mathbb{N}$, we write $f(n) = o(g(n))$ if $f(n)/g(n) \to 0$ as $n \to \infty$. Denote by $\mathsf{DTIME}(T(n))$ the class of languages that are computable in $T(n)$-time.

THEOREM 4.1.7 (Time-hierarchy theorem). *If $f, g : \mathbb{N} \to \mathbb{N}$ satisfy $f(n) \log f(n) = o(g(n))$, then*

$$\mathsf{DTIME}(f(n)) \subsetneq \mathsf{DTIME}(g(n))$$

with respect to every oracle.

PROOF. The technique used to prove this theorem is *diagonalization*, inspired by the set theoretic method of the same name. Since $g(n)$ grows much faster than $f(n) \log f(n)$, we have that $\mathsf{DTIME}(f(n)) \subseteq \mathsf{DTIME}(g(n))$. It remains to show that $\mathsf{DTIME}(g(n)) \not\subseteq \mathsf{DTIME}(f(n))$.

Let U indicate the universal Turing machine of Theorem 2.1.2. We carry out the following procedure: on input x, compute $M_x(x)$ on a suitable universal TM for $g(|x|)$ steps, where M_x is the Turing machine encoded by x. If the computation finishes, output $1 - M_x(x)$. Otherwise, output 0. Let D be the Turing machine that follows this algorithm. By the choice of universal Turing machine and the fact that $f(n) \log f(n) = o(g(n))$, we have $D \in \mathsf{DTIME}(g(n))$.

Next, assume the language determined by D lies in $\mathsf{DTIME}(f(n))$. Then, there exists a Turing M that decides the same language in $O(f(n))$-time. Then M has some encoding, and in fact can be assumed to have infinitely many encodings, so we fix some encoding y that is long enough so that $f(|y|) \log f(|y|)$ is much less than $g(|y|)$. Then the universal Turing machine can simulate M_y on input y within $g(|y|)$ steps, and so $D(y) = 1 - M_y(y) = 1 - M(y)$, contradicting the assumption that $D(y) = M(y)$. Therefore, $\mathsf{DTIME}(g(n)) \not\subseteq \mathsf{DTIME}(f(n))$. $\square$

The time-hierarchy theorem proves that there is no oracle collapse from EXP to P. There is, however, an oracle collapse from PSPACE to P; i.e., there is an oracle relative to which P = PSPACE. The usual method to make use of a PSPACE-complete problem—a language $\mathscr{L}$ in PSPACE such that every other question in PSPACE can be reduced to the question of whether $y \in \mathscr{L}$, where y can be computed from the input in polynomial time. Then an oracle for $\mathscr{L}$ does not give PSPACE any additional computational power, while boosting P up to PSPACE. However, in Section 4.2.4 we will show that there is an even larger class that collapses to P.

Such a collapse also shows that there is an oracle relative to which P = NP. There is also an oracle relative to which P $\neq$ NP, which establishes that no relativizing proof can settle the P versus NP question. A *password oracle* is a type of oracle f constructed so that $\mathsf{C}_1^f \not\subseteq \mathsf{C}_2^f$. Typically, f is a function $\Sigma^* \times \Sigma^* \to \Sigma^*$

chosen so that $PW_f = \{x \in \Sigma : P\}$ lies in C_1^f but not in C_2^f, where P is a proposition depending on the values of $f(x,y)$ for $y \in \Sigma^*$ (the *passwords* of x). The oracle f can be adversarially constructed or, in many cases, selected according to a random process.

THEOREM 4.1.8. *Let $f : \Sigma^{2*} \to \Sigma \cup \{\Box\}$ be a function selected randomly according to the following rules:*

- *For every $x \in \Sigma^n$, there exists a unique $y \in \Sigma^n$ such that $f(x,y) \neq \Box$. This y is selected using the uniform distribution on Σ^n.*
- *For every $x \in \Sigma^n$, if y is the unique element of Σ^n such that $f(x,y) \neq \Box$, then* $\Pr[f(x,y) = 1] = \Pr[f(x,y) = 0] = 1/2$.

Then $(\mathit{cocap} \cdot \mathsf{NP})^f \not\subseteq (\mathsf{P/poly})^f$ with probability 1.

PROOF. For each $x \in \Sigma^*$, define $PW_f(x) = f(x,y)$, where y is the unique element of $\Sigma^{|x|}$ such that $f(x,y) \neq \Box$. Then $PW_f \in (\mathit{cocap} \cdot \mathsf{NP})^f$, because for a given $x \in \Sigma^*$ the unique y can be used as the certificate to check that $PW_f(x) = 1$ or $PW_f(x) = 0$.

Fix an enumeration $\{M_k\}$ of Turing machines. For M_k and input of length n, we allow computation times up to $C_k n^{r_k}$ and advice strings up to length $D_k n^{s_k}$, where the coefficients and exponents are unbounded and increasing as functions of k. Then, since for any $x \in \Sigma^n$ there are 2^n possible values of y, while advice and computation time are polynomials of n, we have

$$\Pr[(\forall n)(\exists \text{ advice } a)(\forall |x| = n)[M_k(x,a) = PW_f(x)]] = 0.$$

$\Box$

The above proof actually establishes the stronger claim that $(\mathit{cocap} \cdot \mathsf{UP})^f \not\subseteq (\mathsf{P/poly})^f$, where UP denotes *unambiguous polynomial time* (see Subsection 4.2.80).

4.2. Overview of the Complexity Zoology Data Set

This section enumerates the complexity classes, inclusions, and oracle separations of Complexity Zoology's data set. The data is listed in the same order as in the data itself: alphabetically by complexity class, with relations listed under the class that appears on the left side of the relation. As a result, many classes

World:	P	D	O	up	ud	u
All oracles	3680	9354	460	20	417	111
All algebraic oracles	3022	3332	326	0	3918	114
Some algebraic oracle	3022	2185	0	326	0	5179
Random oracle	2435	5405	1058	508	156	140
Trivial oracle	2566	2021	4725	0	0	0
Some oracle	11344	2542	0	156	0	0

TABLE 4.1. The current status of the Complexity Zoology data set. This table shows the number of inclusions of each type that exist for each world. P indicates *proven*, D indicates *disproven*, O indicates *open*, up indicates *unknown but provable*, ud indicates *unknown but disprovable*, and uu indicates *unknown with no further data*. The terms *unknown*, *provable*, and *disprovable* are explained in Sections 3.4 and 3.5.

are discussed before the subsection in which they are defined. Some trivial inclusions are omitted. Statements of equality appear under the less preferred name for the class. Large parts of this data set, especially those inclusions that are considered to be open, are based on email and in-person communication with Scott Aaronson and Lance Fortnow.

Complexity Zoology studies the relationships between complexity classes in six different modal worlds: all oracles, all algebraic oracles, some algebraic oracle, the random oracle, the trivial oracle, and some oracle. The data set would be considered complete when each possible inclusion is settled as true, false, or open in each of these six worlds. The current status of the data set is shown in Table 4.1. Note that the trivial world is complete (i.e., there are no unknown inclusions).

4.2.1. $\oplus\mathsf{P}$**.** $\oplus\mathsf{P}$ is the class of languages $\mathscr{L}$ such that for an NP-machine M $x \in \mathscr{L}$ if and only if the computational tree of $M(x)$ has an odd number of accepting paths. In operator terms, it is equal to $\oplus \cdot \mathsf{P}$.

$\oplus\mathsf{P} \subseteq \mathsf{MP}$ with respect to every oracle [**GKR$^+$95**]. $\oplus\mathsf{P} \not\subseteq \mathsf{PP/poly}$ with respect to the random oracle, while it is open whether this is the case with respect to the trivial oracle.

4.2.2. $\mathsf{A_0PP}$**.** A function $f : \Sigma^* \to \mathbb{Z}$ lies in GapP if and only if there exists a polynomial-time Turing machine M and a function $p(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$f(x) = |\{y \in \Sigma^{p(|x|)} : M(x,y) = 1\}| - |\{y \in \Sigma^{p(|x|)} : M(x,y) = 0\}|.$$

Now $\mathscr{L} \in \mathsf{A_0PP}$ if and only if there exists a function $f \in \mathsf{GapP}$ and a function $g : \Sigma^* \to \mathbb{N}$ in FP such that

$$x \in \mathscr{L} \Longrightarrow f(x) > g(x),$$

$$x \notin \mathscr{L} \Longrightarrow f(x) < g(x)/2.$$

Note that a language in $\mathscr{L} \in \mathsf{PP}$ can be characterized by the sign of some GapP function. Since GapP is closed under subtraction, it follows that $\mathsf{A_0PP} \subseteq \mathsf{PP}$ for all oracles.

Kuperberg has also shown [**Kup09**] that $\mathsf{A_0PP}$ is equal to the class SBQP of languages $\mathscr{L}$ such that there exists a BQP-machine M and a function $p(n) = O(n^*)$ such that for all $x \in \Sigma^*$,

$$x \in \mathscr{L} \Longrightarrow \Pr[M(x) = 1] \geq 2^{-p(|x|)},$$

$$x \notin \mathscr{L} \Longrightarrow \Pr[M(x) = 1] \leq 2^{-p(|x|)-1}.$$

4.2.3. AH**.** The *arithmetical hierarchy* is analogous to the polynomial hierarchy, except that R and RE are the fundamental complexity classes rather than P and NP. Thus, where PH is a hierarchy of complexity, AH is one of computability. For every $n \in \mathbb{N}$, define Δ_n, Σ_n, and Π_n by recursion as follows: $\Delta_0 = \Sigma_0 = \Pi_0 = \mathsf{R}$, and for every $n \in \mathbb{N}$,

$$\Delta_{n+1} = \mathsf{R}^{\Sigma_n},$$

$$\Sigma_{n+1} = \mathsf{RE}^{\Sigma_n},$$

$$\Pi_{n+1} = co \cdot \mathsf{RE}^{\Sigma_n}.$$

Then set $\mathsf{AH} = \bigcup_{n \in \mathbb{N}} \Sigma_n$.

Since $\mathsf{R} \subseteq \mathsf{RE}$ and R is a symmetric class it is immediate that $\Delta_n \subseteq \Sigma_n$ and $\Delta_n \subseteq \Pi_n$ for all $n \in \mathbb{N}$. Moreover, for $n \geq 1$ we have

$$\Sigma_n = \mathsf{RE}^{\Sigma_{n-1}} \subseteq \mathsf{R}^{\mathsf{RE}^{\Sigma_{n-1}}} = \mathsf{R}^{\Sigma_n} = \Delta_{n+1},$$

and for $n = 0$ we have

$$\Sigma_0 = \mathsf{R} \subseteq \mathsf{R}^{\mathsf{R}} = R^{\Sigma_0} = \Delta_1,$$

so we also have $\Sigma_n \subseteq \Delta_{n+1}$ and $\Pi_n \subseteq \Delta_{n+1}$ for all $n \in \mathbb{N}$. It follows that $\mathsf{AH} = \bigcup_{n \in \mathbb{N}} \Delta_n = \bigcup_{n \in \mathbb{N}} \Pi_n$. Also, just as there is an alternating quantifier definition for the polynomial hierarchy, there is one for the arithmetical hierarchy. For example, $\mathscr{L} \in \Sigma_2$ if and only there exists a computable relation R such that

$$x \in \mathscr{L} \iff (\exists y)(\forall z)R(x,y,z)$$

for each $x \in \Sigma^*$.

AH is closed under exponential padding: $\mathsf{AH} = \mathit{exppad} \cdot \mathsf{AH}$.

4.2.4. AH_{plo}**.** AH_{plo} denotes AH with a polynomially-limited oracle access. The definitions of AH_{plo} and AH in the unrelativized case are identical, but AH^f and AH^f_{plo} have different definitions for an arbitrary oracle f.

For $n \in \mathbb{N}$ and an oracle $f : \Sigma^* \to \Sigma^*$, we define $f|_n : \Sigma^* \to \Sigma^*$ according to the rule

$$f|_n(x) = \begin{cases} f(x) & \text{if } |x| < n, \\ 0 & \text{otherwise.} \end{cases}$$

then we say that $\mathscr{L}$ lies in $\mathsf{RE}^f_{\mathsf{plo}}$ if and only if there exists an oracle Turing machine M and a function $p(n) = O(n^*)$ such that

$$x \in \mathscr{L} \implies M \text{ accepts } x \text{ with } f|_{p(|x|)}\text{-oracle.}$$

Now we can define AH^f_{plo} as we defined AH: use RE^f_{plo} and $\mathsf{R}^f_{plo} = \mathit{cocap} \cdot \mathsf{RE}^f_{plo}$ in the place of RE and R in the unrelativized definition of AH.

AH_{plo} is not contained in NEXP/poly and RE with respect to the random oracle.

Limiting AH to polynomial-length oracle queries has the effect of creating an oracle collapse from AH all the way to P. In fact, we can prove something even stronger: there exists an oracle relative to which $\mathsf{P} = \mathsf{AH}[poly]$, where $\mathsf{AH}[poly]$ is a class containing AH. $\mathsf{AH}[poly]$ is the class of languages $\mathscr{L}$ such that there exists a computable relation R and a function $p(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$\begin{aligned} x \in \mathscr{L} \iff & (Q_1 y_1 \in \Sigma^*) \dots (Q_{p(|x|)} y_{p(|x|)} \in \Sigma^*) R(x, y_1, \dots, y_{p(|x|)}) \text{ or} \\ & (\bar{Q}_1 y_1 \in \Sigma^*) \dots (\bar{Q}_{p(|x|)} y_{p(|x|)} \in \Sigma^*) R(x, y_1, \dots, y_{p(|x|)}), \end{aligned}$$

where Q_k is $\forall$ when k is even and $\exists$ when k is odd, and $\bar{Q}_k$ is $\exists$ when k is even and $\forall$ when k is odd.

The relativizing inclusion $\mathsf{AH} \subseteq \mathsf{AH}[poly]$ is immediate from the alternating quantifier definition of AH. Using the same quantifier definition as above, set

$$\mathtt{AHSAT} = \{\langle \alpha, x, 1^m\rangle : (Q_1 y_1 \in \Sigma^*)\dots(Q_m y_m \in \Sigma^*)[M_\alpha(x, y_1, \dots, y_m) = 1]) \text{ or}$$
$$(\bar{Q}_1 y_1 \in \Sigma^*)\dots(\bar{Q}_m y_m) \in \Sigma^*)[M_\alpha(x, y_1, \dots, y_m) = 1])\},$$

where M_α denotes the Turing machine that α encodes.

PROPOSITION 4.2.1. $\mathtt{AHSAT}$ *is* $\mathsf{AH}[poly]$*-complete.*

PROOF. To see that $\mathtt{AHSAT} \in \mathsf{AH}[poly]$, we can simply take p to be the identity function and R to be the relation

$$R(\langle \alpha, x, 1^m\rangle, y_1, \dots, y_r) \iff r \geq m \,\&\, M_\alpha(x, y_1, \dots, y_m) = 1.$$

Then R and p witness that $\mathtt{AHSAT} \in \mathsf{AH}[poly]$ per the definition.

Now suppose that $\mathscr{L} \in \mathsf{AH}[poly]$, and let the relation R and the function $p(n) = O(n^*)$ witness that $\mathscr{L} \in \mathsf{AH}[poly]$. Then, if α is a code for the Turing machine that computes R, we have

$$x \in \mathscr{L} \iff \left\langle \alpha, x, 1^{p(|x|)} \right\rangle \in \mathtt{AHSAT}.$$

Since p is assumed to be time-constructible, the function $x \mapsto \left\langle \alpha, x, 1^{p(|x|)} \right\rangle$ is polynomial-time computable, and so $\mathscr{L}$ is polynomial-time reducible to $\mathtt{AHSAT}$. Therefore, $\mathtt{AHSAT}$ is a complete problem for $\mathsf{AH}[poly]$. $\square$

Since $\mathsf{P} \subseteq \mathsf{AH}[poly]$ relative to any oracle, in particular $\mathsf{P}^{\mathtt{AHSAT}} \subseteq \mathsf{AH}[poly]_{plo}^{\mathtt{AHSAT}}$. Moreover, adding an oracle for a problem in $\mathsf{AH}[poly]$ does not increase the computational power of $\mathsf{AH}[poly]_{plo}$, because oracle calls can be replaced with references to the alternating-quantifier relation corresponding to the oracle. So $\mathsf{AH}[poly]_{plo}^{\mathtt{AHSAT}} \subseteq \mathsf{AH}[poly] \subseteq \mathsf{P}^{\mathtt{AHSAT}}$. Hence:

THEOREM 4.2.1 (Kuperberg). $\mathsf{P}^{\mathtt{AHSAT}} = \mathsf{AH}[poly]_{plo}^{\mathtt{AHSAT}}$.

4.2.5. ALL. ALL is the “complexity” class of all languages.

While it is not an interesting complexity class in and of itself, ALL is a useful point of reference in the hierarchy of complexity class inclusions. For instance, it can be helpful to know which classes are immediately below ALL in out hierarchy of inclusions, and it is often a nontrivial theorem that a particular class is or is not ALL.

4.2.6. AM**.** AM is the class of language that can be computed using the *Arthur-Merlin protocol*. The protocol consists of these steps:

(1) Arthur queries Merlin in an attempt to determine whether $x \in \mathscr{L}$. Arthur is a polynomial-time Turing machine with access to a polynomial-length, uniformly random string y of bits. This string is sent to Merlin.
(2) Merlin sends a purported proof z (with $|z|$ a polynomial of the length of the input) that $x \in \mathscr{L}$. Merlin is an oracle with access to Arthur's coin tosses, so Merlin can send any z based on the situation.
(3) Arthur decides whether to accept the input x based on $\langle x, y, z \rangle$.

The language $\mathscr{L}$ is considered to lie in AM if the Arthur-Merlin protocol is likely decide the question of whether $x \in \mathscr{L}$ correctly. Formally, we can set $\mathsf{AM} = \mathcal{BP} \cdot \mathsf{NP}$.

Immediately, AM is contained in QAM, IP, and RG(2) for every oracle. It is also the case that $\mathsf{AM} \subseteq \Pi_2\mathsf{P}$ for every oracle [**Bab85**]. On the other hand, there is an oracle relative to which $\mathsf{AM} \not\subseteq \mathsf{PP}$ [**Ver92**], and $\mathsf{AM} \subseteq \mathsf{NP}$ with respect to the random oracle [**BM88**].

4.2.7. AP**.** An *alternating Turing machine* is a non-deterministic Turing machine in which each state is assigned either the symbol $\wedge$ or the symbol $\vee$. Each configuration in a computational tree involving an alternating Turing machine can likewise be assigned $\wedge$ or $\vee$ based on the current state of the machine.

To determine whether an alternating Turing machine M accepts an input x, assign an element of Σ to each node in the computational tree according to the following procedure:

(1) If a configuration c is a halting configuration, set $v(c) = 0$ if the output is 0 and $v(c) = 1$ if the output is 1.
(2) If c is an $\wedge$-configuration, assign $v(c) = 1$ if $v(c') = 1$ for each descendant c' of c in the computational tree. Otherwise, set $v(c) = 0$.
(3) If c is an $\vee$-configuration, assign $v(c) = 1$ if $v(c') = 1$ for some descendant c' of c in the computational tree.

Then, we say that M accepts the input x if and only if $v(s_{\text{start}}) = 1$.

$\mathscr{L} \in \mathsf{AP}$ if there exists a polynomial-time alternating Turing machine M such that

$$x \in \mathscr{L} \Longleftrightarrow M \text{ accepts } x.$$

for every $x \in \Sigma^*$.

$\mathsf{AP} = \mathsf{PSPACE}$ for every oracle [**CKS81**].

4.2.8. AWPP**.** AWPP is the class of "almost wide" PP problems. For a polynomial-time Turing machine M and a function $p(n) = O(n^*)$, define

$$d_{M,p}(x) = |\{y \in \Sigma^{p(|x|)} : M(x,y) = 0\}| - |\{y \in \Sigma^{p(|x|)} : M(x,y) = 1\}|$$

for each $x \in \Sigma^*$. $\mathscr{L} \in \mathsf{AWPP}$ if there exists a polynomial-time Turing machine M, a function $f : \Sigma^* \to \mathbb{N}$, and functions $p(n), q(n) = O(n^*)$ such that

$$x \in \mathscr{L} \Longrightarrow (1 - 2^{-q(|x|)}) f(x) \leq d_{M,p}(x) \leq f(x),$$
$$x \notin \mathscr{L} \Longrightarrow 0 \leq d_{M,p}(x) \leq 2^{-q(|x|)} f(x).$$

$\mathsf{AWPP} \subseteq \mathsf{A_0PP}$ for all oracles [**Vya03**].

4.2.9. BPP**.** This is the class of languages computable in polynomial time using randomness, where the computer is correct with a probability of at least $2/3$. In other words, $\mathsf{BPP} = \mathcal{BP} \cdot \mathsf{P}$.

It is contained in BQP, MA, NISZK, and SZK with respect to every oracle. $\mathsf{BPP} \subseteq \Sigma_2$ for every oracle [**Lau83**], but there is an oracle relative to which $\mathsf{BPP} \not\subseteq \Delta_2\mathsf{P}$. In the random oracle world, $\mathsf{P} = \mathsf{BPP}$ [**BG81**]. $\mathsf{BPP} \not\subseteq \Delta_2\mathsf{P}$ is open in the unrelativized world.

4.2.10. BPP_{path}**.** BPP_{path} is the original name for $\mathsf{PostBPP}$, or BPP with postselection. It was originally defined as the class of all languages $\mathscr{L}$ such that there exists a non-deterministic polynomial-time Turing machine M and a threshold $\varepsilon > 0$ such that for every $x \in \Sigma^*$, the number of paths for which $M(x) = \mathscr{L}(x)$ is at least $(1/2 + \varepsilon) \cdot T(x)$, where $T(x)$ is the total number of computational paths of M with input x [**HHT97**].

4.2.11. BQP**.** Informally, BQP is the set of languages that can be computed efficiently using a quantum computer. A quantum computer can be modeled by a uniform family of quantum circuits or a quantum Turing machines whose configurations can be quantum states.

$\mathsf{BQP} \subseteq \mathsf{BQP/mpoly}$ is immediate. BQP is also contained in AWPP for every oracle [**FR98**], as well as NIQSZK, QCMA, and QSZK. There exist oracles relative to which BQP is not contained in $\oplus\mathsf{P}$, PH, MA [**Wat00**], $\mathsf{NP/poly}$, $\mathsf{PostBPP}$ [**Aar10**] [**Che16**], and SZK [**CCD$^+$03**]. Recently, PH was added to this list of oracle separations [**RT18**].

In the random oracle world, the following inclusions are open:

$$\mathsf{BQP} \subseteq \mathsf{BPP}, \mathsf{IP}, \mathsf{MIP}, \mathsf{P}, \mathsf{PH}, \oplus\mathsf{P}, \mathsf{NP/poly}.$$

It is also open whether BQP is contained in PH and $\mathsf{NP/poly}$ in the unrelativized world and whether there exists an oracle such that $\mathsf{BQP} \not\subseteq \mathsf{MIP}$.

4.2.12. $\mathsf{BQP/mpoly}$**.** $\mathsf{BQP/mpoly}$ is BQP with "Merlinized" polynomial advice.

DEFINITION 4.2.1. $\mathscr{L} \in \mathsf{BQP/mpoly}$ if there exists a BQP-machine M and a polynomial-length advice function $f : \mathbb{N} \to \Sigma^*$ such that

$$x \in \mathscr{L} \iff \Pr[M(x, f(n)) = \mathscr{L}(x)] \geq 2/3.$$

The advice is "Merlinized" in the sense that the probability threshold of $\geq 2/3$ need only be observed when the advice is good, just as Arthur does not need to satisfy the probability gap when Merlin gives a poor argument in the Arthur-Merlin protocol.

$\mathsf{BQP/mpoly}$ is also the class of languages that can be computed using some family of polynomial-sized quantum circuits.

$\mathsf{BQP/mpoly} \subseteq \mathsf{BQP/qpoly}, \mathsf{QMA/qpoly}$ for every oracle.

4.2.13. $\mathsf{BQP/qpoly}$**.** $\mathsf{BQP/qpoly}$ is BQP with a polynomial amount of quantum advice. This means that, rather than a classical advice string, a BQP-machine is given an advice string that exists in a state of superposition. That is, the advice string takes the form $\sum_s \alpha_s |s\rangle$, where the sum is over polynomial-length strings in Σ.

It is immediate that $\mathsf{BQP/qpoly} \subseteq \mathsf{QCMA/qpoly}$ relative to every oracle. In fact, Aaronson and Drucker have shown that $\mathsf{BQP/qpoly} \subseteq \mathsf{QCMA/mpoly}$ [**AD10**]. The relationship between classical and quantum advice in the case of BQP is uncertain: it is unknown whether $\mathsf{BQP/qpoly}$ is any larger than $\mathsf{BQP/mpoly}$ [**AK07**].

4.2.14. $\mathsf{BQPSPACE}$**.** This is the class of languages computable by a *quantum Turing machine* using a polynomial amount of space. A quantum Turing machine is a Turing machine that includes a quantum tape on which qubits can be recorded, as well as a finite register for performing measurements. As with other non-deterministic versions of PSPACE (see Subsections 4.2.37 and 4.2.47), $\mathsf{BQPSPACE}$ is equal to PSPACE for every oracle [**Wat03**].

4.2.15. CH**.** The *counting hierarchy* is the class that results from an infinite stack of PP-oracles. If $C_0\mathsf{P} = \mathsf{P}$ and $C_{k+1}\mathsf{P} = \mathsf{PP}^{C_k\mathsf{P}}$ for each $k \in \mathbb{N}$, then $\mathsf{CH} = \bigcup_k C_k\mathsf{P}$. The classes $\mathsf{P}^{C_k\mathsf{P}}$ are clearly inside CH, and so $\mathsf{CH} = \mathsf{P}^{\mathsf{CH}}$ for every oracle. Moreover, because $\mathsf{PP} \subseteq \mathsf{PSPACE}$, and because giving PSPACE access to an oracle that is already inside PSPACE does not increase the power of the class, we also have the relativizing inclusion $\mathsf{CH} \subseteq \mathsf{PSPACE}$.

It is unknown whether $\mathsf{CH} \subseteq \mathsf{P}^{\#\mathsf{P}}$ (since $\mathsf{P}^{\#\mathsf{P}} = \mathsf{P}^{\mathsf{PP}}$, this would imply that the counting hierarchy collapses) relative to a random oracle, and indeed whether there is any oracle relative to which the counting hierarchy does not collapse.

4.2.16. $\mathit{cocap} \cdot \mathsf{RE}$**.** The class $\mathit{cocap} \cdot \mathsf{RE} = \mathsf{RE} \cap \mathsf{coRE}$ is equal to R, the class of recursive decision problems. RE is the class of languages that a computer can enumerate the elements of. If a computer can enumerate the elements of a language as well as its complement, then the computer can decide whether x lies in the language by enumerating the language and its complement until it is revealed where x lies.

4.2.17. $\mathit{cocap} \cdot \mathsf{RP}$**.** The intersection of RP and coRP is equal to ZPP for every oracle.

4.2.18. $\Delta_2\mathsf{P}$**.** The class $\Delta_2\mathsf{P}$ is a level of the polynomial hierarchy, defined to be P^{NP}. It is therefore contained in $\Sigma_2\mathsf{P} = \mathsf{NP}^{\mathsf{NP}}$. Less obviously, $\Delta_2\mathsf{P}$ lies in $S_2\mathsf{P}$ relative to every oracle [**RS98**]. While it is unknown whether $\Delta_2\mathsf{P} \subseteq \mathsf{PP}$ in the unrelativized world, there exists an algebraic oracle relative to which $\Delta_2\mathsf{P} \subseteq \mathsf{PP}$ [**AW09**]. $\Delta_2\mathsf{P}$ is outside $\mathsf{PostBPP}$ relative to the random oracle.

4.2.19. $\Delta_3\mathsf{P}$**.** This class is $\mathsf{P}^{\Sigma_2\mathsf{P}}$, another level of the polynomial hierarchy. It is unknown whether $\Delta_3\mathsf{P} \subseteq \mathsf{PostBPP}$ relative to the trivial oracle.

4.2.20. EXP**.** This is the class of languages that can be decided in exponential time. It can be succinctly defined as $\mathit{exppad} \cdot \mathsf{P}$. By the time hierarchy theorem, $\mathsf{EXP} \not\subseteq \mathsf{P}$ relative to every oracle, but there is an oracle relative to which $\mathsf{EXP} = \mathsf{ZPP}$ [**Hel84**], as well as an oracle relative to which $\mathsf{EXP} = \mathit{cocap} \cdot \mathsf{UP}$. In the unrelativized world, it is unknown whether either of these equalities is true.

4.2.21. $\mathsf{EXP}^{\mathsf{NP}}$**.** This class is $\bigcup_{f \in \mathsf{NP}} \mathsf{EXP}^{f}$, or EXP relative to an NP-oracle. (It can also be defined to be $\mathsf{EXP}^{\mathsf{SAT}}$, or any other NP-complete problem.) Since $\mathsf{EXP} = \mathit{exppad} \cdot \mathsf{P}$, it follows that $\mathsf{EXP}^{\mathsf{NP}} = \mathit{exppad} \cdot \Delta_2\mathsf{P}$. There exist oracles relative to which this class is equal to BPP [**BT00**] and SPP. The time hierarchy theorem relativizes, so $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \Delta_2\mathsf{P}$ for every oracle. Additionally, $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{NEXP}$ in the random oracle world.

In the unrelativized world, the inclusions $\mathsf{EXP}^{\mathsf{NP}} \subseteq \mathsf{NEXP/poly}, \mathsf{BPP}, \mathsf{SPP}$ are open.

4.2.22. EXP_{plo}**.** As with AH_{plo}, EXP_{plo} is equal to EXP in the unrelativized world, while relative to an oracle, the oracle calls are limited in length to a polynomial of $|x|$ for the input x.

4.2.23. IP**.** An *interactive proof protocol* is an interaction between a probabilistic polynomial-time verifier and a deterministic prover with infinite computational power. The prover can compute any function, but unlike the Arthur-Merlin protocol, the prover does not know the random bits that the verifier uses in his computations. The verifier attempts to determine whether the input x lies in a given language $\mathscr{L}$ within a number of rounds that is a polynomial of $|x|$. The language lies in IP if and only if the protocol is *complete*, meaning that if $x \in \mathscr{L}$ then the verifier can be convinced of this fact with a probability of at least $2/3$, and *sound*, meaning that if $x \notin \mathscr{L}$ then the verifier can be convinced that $x \in \mathscr{L}$ with a probability of at most $1/3$.

While the prover has the power to determine even uncomputable functions, to determine the optimal strategy for the prover, given an input x, only a PSPACE-machine is necessary. Using a polynomial-space computation, the prover can simulate as many interactions with the prover as needed. Therefore, $\mathsf{IP} \subseteq \mathsf{PSPACE}$ relative to every oracle.

IP is also contained in its generalizations QIP and MIP. There is an oracle relative to which $\mathit{cocap} \cdot \mathsf{IP} \not\subseteq \mathsf{PH}$ [**AGH90**]. Open problems in the random oracle world include whether $\mathit{cocap} \cdot \mathsf{IP}$ is contained in PP/poly, QMA/qpoly, CH, or QMA(2).

4.2.24. IPP**.** This class is the unbounded version of IP. More precisely, the completeness and soundness conditions are weakened so that when $x \in \mathscr{L}$, the prover can convince the verifier to accept with a probability of at least $1/2$, while if $x \notin \mathscr{L}$, then in all circumstances the verifier accepts with a probability of at most $1/2$. $\mathsf{IPP} = \mathsf{PSPACE}$ with respect to every oracle [**CCG**$^+$**94**].

4.2.25. MA**.** The Merlin-Arthur protocol is similar to the Arthur-Merlin protocol, except that Merlin sends Arthur a message without knowing Arthur's randomized bits, and Arthur decides whether to accept that the input x lies in the language $\mathscr{L}$ using a polynomial-time randomized algorithm. As with many other interaction protocols, $\mathscr{L} \in \mathsf{MA}$ if and only if $x \in \mathscr{L}$ implies that Arthur can be persuaded that $x \in \mathscr{L}$ with probability $\geq 2/3$, while $x \notin \mathscr{L}$ implies that Arthur can be persuaded that $x \in \mathscr{L}$ with probability $\leq 2/3$.

$\mathsf{MA} \subseteq \mathsf{QCMA}, \mathsf{SBP}$ relative to every oracle. It is also the case that $\mathsf{MA} \subseteq \mathsf{S_2P}$ relative to every oracle [**RS98**]. In the unrelativized world, the inclusion $\mathit{cocap} \cdot \mathsf{MA} \subseteq (\mathsf{NP} \cap \mathsf{coNP})/\mathsf{poly}$ is open.

4.2.26. MAEXP**.** The class MAEXP is the class of languages computable using the Merlin-Arthur protocol, where Arthur has exponential computational power. In operator terms, $\mathsf{MAEXP} = \mathit{exppad} \cdot \mathsf{MA}$. MA is strictly contained in MAEXP relative to every oracle.

There exist oracles relative to which MAEXP is contained in $\oplus\mathsf{P}$, $\Delta_2\mathsf{P}$, and P/poly, while in the unrelativized world, $\mathit{cocap} \cdot \mathsf{MAEXP} \not\subseteq \mathsf{P}/\mathsf{poly}$ [**BFT98**]. In fact, $\mathit{cocap} \cdot \mathsf{MAEXP} \not\subseteq \mathsf{P}/\mathsf{poly}$ relative to every algebraic oracle [**AW09**]. Relative to the trivial oracle, it is open whether MAEXP is contained in $(\mathsf{NP} \cap \mathsf{coNP})/\mathsf{poly}$, BQP, $\Delta_2\mathsf{P}$, SPP, $\mathit{cocap} \cdot \mathsf{NISZK}$, or $\mathit{cocap} \cdot \mathsf{SBP}$.

4.2.27. MIP**.** An interactive proof protocol can involve multiple provers. In the class MIP, there are two provers and one verifier. As in IP, the verifier is a polynomial-time probabilistic Turing machine and the provers have infinite computing power; however, the provers are not allowed to communicate with each other.

MIP is contained in NEXP_{plo} and its quantum counterpart QMIP_{ne} relative to every oracle. Open questions include whether $\mathsf{MIP} \subseteq \mathit{co} \cdot \mathsf{MAEXP}$ for every oracle, whether $\mathsf{MIP} \subseteq \mathit{co} \cdot \mathsf{NEXP}$ for the random oracle, and whether $\mathit{cocap} \cdot \mathsf{MIP} \subseteq \mathsf{EXP}$ for all oracles and for the random oracle.

4.2.28. MP**.** For $x \in \Sigma^*$, denote by $x(k) \in \Sigma$ the bit in x at position k (indexing from zero). Then, we say $\mathscr{L} \in \mathsf{MP}$ if and only if there exists $f \in \#\mathsf{P}$ and a polynomial-time computable $g : \Sigma^* \to \mathbb{N}$ such that $\mathscr{L}(x) = f(x)(g(x))$ for every $x \in \Sigma^*$. MP stands for "middle bit," because it can be assumed that $|f(x)|$ is always odd and that $g(x) = \frac{1}{2}(|f(x)| - 1)$ [**GKR**$^+$**95**].

MP is contained in $\mathsf{P}^{\#\mathsf{P}}$ relative to every oracle, since $\mathsf{P}^{\#\mathsf{P}}$ can use a $\#\mathsf{P}$-oracle to determine the middle bit of a function in $\#\mathsf{P}$.

4.2.29. NEXP**.** This class is the non-deterministic counterpart to EXP; it can be defined to be $\mathit{exppad} \cdot \mathsf{NP}$. Like NP, it is distinct from its complement with respect to an oracle constructed via the password argument. Additionally, $\mathit{cocap} \cdot \mathsf{NEXP} \not\subseteq \mathsf{NP}$ with respect to every oracle by a diagonalization argument.

There exists an oracle relative to which $\mathsf{NEXP} \subseteq \oplus\mathsf{P}$ [**Aar06**]. In the unrelativized world, the inclusions $\mathsf{NEXP} \subseteq \mathsf{SPP}$ and $\mathsf{NEXP} \subseteq \mathit{co} \cdot \mathsf{MAEXP}$ are open.

4.2.30. NEXP_{plo}**.** This class is identical to NEXP, except that oracle calls are limited in length to a polynomial of $|x|$ for input x. It is contained in MIP [**AW09**] and QMIP_{le} [**IV12**] with respect to all algebraic oracles. The question of whether $\mathsf{NEXP}_{plo} \subseteq \mathsf{MP}$ relative to every oracle is open.

4.2.31. $\mathsf{NEXP/poly}$**.** Naturally, $\mathsf{NEXP/poly}$ is defined to be $\mathit{poly} \cdot \mathsf{NEXP}$. The question of whether $\mathsf{NEXP/poly} \subseteq \mathsf{P/poly}$ in the unrelativized world is open.

4.2.32. NIQSZK**.** This class is the non-interactive version of QSZK, just as NISZK is the non-interactive version of SZK. Accordingly, it lies in QSZK for every oracle. The inclusions

$$\mathit{cocap} \cdot \mathsf{NIQSZK} \subseteq \mathsf{CH}, \mathsf{PP/poly}, \mathsf{QMA/qpoly}$$

are open with respect to the trivial oracle.

4.2.33. NISZK**.** There is a modified version of the statistical zero-knowledge protocol in which interaction between the prover and the verifier is disallowed. More precisely, a language lies in NISZK if and only if the language can be computed using a statistical zero-knowledge protocol, with the additional constraints that the prover and verifier share a single randomly generated string as the source of their randomness, and the only communication allowed between the two parties is a single message from the prover to the verifier.

There exist oracles relative to which $\mathit{cocap} \cdot \mathsf{NISZK} \not\subseteq \mathsf{PP}$ [**BCH$^+$17**] and $\mathit{cocap} \cdot \mathsf{NISZK} \not\subseteq \mathsf{BQP/qpoly}$. Unlike SZK, there is an oracle which NISZK is distinct from its complement [**LZ17**]. It is open whether $\mathsf{NISZK} \subseteq \mathsf{NIQSZK}$ in either the world of all oracles or in the unrelativized world. The inclusions

$$\mathit{cocap} \cdot \mathsf{NISZK} \subseteq \mathsf{BQP/qpoly}, \mathsf{PP}, \mathsf{QMA}(2), \mathsf{QRG}(1), (\mathsf{NP} \cap \mathsf{coNP})/\mathsf{poly}$$

are likewise open in the unrelativized world.

4.2.34. NP**.** The class of languages that can be computed in non-deterministic polynomial time can, naturally, be defined to be $\mathcal{N}\cdot\mathsf{P}$. It is immediate that $\mathsf{NP}\subseteq\mathsf{MA}$ for every oracle, since NP is precisely the result of the Merlin-Arthur protocol with a deterministic Arthur. Put another way, NP is the class of languages $\mathscr{L}$ such that if $x\in\mathscr{L}$, then there is a proof of this fact that a computer can check in time $p(|x|)$ for some $p(n)=O(n^*)$.

As it is one of the most thoroughly studied complexity classes, there are many known oracle separations involving NP. With respect to the random oracle, $\mathsf{NP}\not\subseteq\mathsf{UP}$ [**Bei89**] and $\mathsf{NP}\not\subseteq\mathsf{BQP}$ [**BBBV97**]. There exist oracles relative to which $\mathsf{NP}\cap\mathsf{coNP}\not\subseteq\mathsf{BQP}$ [**BBBV97**], $\mathsf{NP}\not\subseteq\mathsf{BQP/qpoly}$ [**Aar04**], $\mathsf{NP}\not\subseteq co\cdot\mathsf{IP}$, $\mathsf{NP}\cap\mathsf{coNP}\not\subseteq\mathsf{AWPP}$, $\mathsf{NP}\not\subseteq co\cdot\mathsf{A_0PP}$, and there exist algebraic oracles relative to which $\mathsf{NP}\not\subseteq\mathsf{BQP}$ and $\mathsf{NP}\not\subseteq co\cdot\mathsf{MA}$ [**AW09**].

In addition to the famous $\mathsf{NP}\subseteq\mathsf{P}$ and $\mathsf{NP}\subseteq\mathsf{coNP}$ in the unrelativized world, open questions include whether $\mathsf{NP}\cap\mathsf{coNP}\subseteq\mathsf{P}$ relative to the random oracle, whether $\mathsf{NP}\subseteq co\cdot\mathsf{A_0PP}$ or $\mathsf{NP}\cap\mathsf{coNP}\subseteq\mathsf{AWPP}$ relative to the trivial oracle, and whether $\mathsf{NP}\subseteq co\cdot\mathsf{AM}$ relative to every algebraic oracle.

4.2.35. $(\mathsf{NP}\cap\mathsf{coNP})/\mathsf{poly}$**.** As the class's name would suggest, $(\mathsf{NP}\cap\mathsf{coNP})/\mathsf{poly} = \mathit{poly}\cdot\mathit{cocap}\cdot\mathsf{NP}$. Complexity Zoology is able to determine the place of this class in the inclusion hierarchy automatically through the properties of complexity class operators. In the random oracle world, the inclusion $(\mathsf{NP}\cap\mathsf{coNP})/\mathsf{poly}\subseteq\mathsf{P/poly}$ is open.

4.2.36. $\mathsf{NP/poly}$**.** Not only is $\mathsf{NP/poly}=\mathit{poly}\cdot\mathsf{NP}$, but by the same derandomization argument used to show $\mathsf{BPP}\subseteq\mathsf{P/poly}$, we also have $\mathsf{NP/poly}=\mathit{poly}\cdot\mathsf{AM}$.

Since $\mathsf{NP}\subseteq\mathsf{MA}$, it follows that $\mathsf{NP/poly}\subseteq\mathsf{QMA/mpoly}$ for every oracle. Likewise, $\mathsf{NP/poly}\subseteq\mathsf{QCMA/qpoly}$ for every oracle. On the other hand, there is an oracle relative to which $\mathsf{NP/poly}\not\subseteq(\mathsf{NP}\cap\mathsf{coNP})/\mathsf{poly}$ [**FFKL03**].

4.2.37. $\mathsf{NPSPACE}$**.** This class is the non-deterministic version of PSPACE: $\mathsf{NPSPACE}=\mathcal{N}\cdot\mathsf{PSPACE}$. However, $\mathsf{NPSPACE}$ is no larger than PSPACE:

THEOREM 4.2.2 (Savitch, [**Sav70**]). $\mathsf{NPSPACE}=\mathsf{PSPACE}$ *for every oracle.*

PROOF SKETCH. Given a directed graph with a set V of n vertices and a pair of vertices v,w, write $\mathrm{STCON}(v,w,k)$ if there exists a path of length $\leq k$ connecting v to w. To check whether $\mathrm{STCON}(v,w,k)$, recursively check whether each vertex x lies halfway along a connecting path:

$$\mathrm{STCON}(v,w,k) \iff \bigvee_{x\in V} (\mathrm{STCON}(v,x,\lfloor k/2\rfloor) \,\&\, \mathrm{STCON}(x,w,\lceil k/2\rceil)) \quad (k \geq 2).$$

The resulting algorithm uses an $O((\log n)^2)$ amount of space.

Thus, given an $O(f(n))$-space non-deterministic algorithm for a language $\mathscr{L}$, we check the $2^{O(f(n))}$-vertex computational tree for a path from the starting configuration to the accepting configuration, resulting in a deterministic $O((f(n))^2)$-space algorithm. □

4.2.38. P. The class of languages that are computable in polynomial time is the smallest class in Complexity Zoology's data set. There are smaller classes studied in complexity theory—such as L, the class of languages that are computable using a logarithmic amount of space—P is chosen as the bottom of the inclusion hierarchy so that, for instance, polynomial-length oracle calls are allowed.

Since P is the smallest class in Complexity Zoology's data set, it has relatively few relations in the input file: we have $\mathsf{P} \subseteq \mathsf{UP}$ and $\mathsf{P} \subseteq \mathsf{ZPP}$ relative to all oracles, both of which are immediate from the classes involved.

4.2.39. $\mathsf{P/poly}$. The class $\mathsf{P/poly} = \mathit{poly} \cdot \mathsf{P}$ is also equal to $\mathit{poly} \cdot \mathsf{P}$ is also equal to $\mathit{poly} \cdot \mathsf{BPP}$ via the usual derandomization argument. $\mathsf{P/poly} \subseteq \mathsf{BQP/mpoly}$ for every oracle, since the latter class is a generalization of $\mathsf{P/poly}$; and $\mathsf{P/poly} \not\subseteq \mathsf{AH}$ for every oracle, because $\mathsf{P/poly}$ is uncountable while AH is countable.

4.2.40. P^{PP}. We have $\mathsf{P}^{\mathsf{PP}} = \bigcup_{f\in\mathsf{PP}} \mathsf{P}^f$. As was discussed in Section 4.1, $\mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\#\mathsf{P}}$ for every oracle, because we can simulate a #P-oracle using multiple PP-oracle calls. We also have $\mathsf{P}^{\mathsf{PP}} \subseteq \mathsf{PP}^{\mathsf{PP}} \subseteq \mathsf{PP}^{\mathsf{C_1P}} \subseteq \mathsf{CH}$.

4.2.41. $\mathsf{P}^{\#\mathsf{P}}$. We have $\mathsf{P}^{\#\mathsf{P}} = \bigcup_{f\in\#\mathsf{P}} \mathsf{P}^f$. The inclusion $\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{MP}$ is open for both the random and trivial oracles.

4.2.42. PH. The *polynomial hierarchy* is the class that results from an infinite hierarchy of NP-oracles. We set $\Sigma_0\mathsf{P} = \mathsf{P}$ and $\Sigma_{k+1}\mathsf{P} = \mathsf{NP}^{\Sigma_k\mathsf{P}}$ for $k \in \mathbb{N}$, as well as $\Pi_k\mathsf{P} = \mathit{co} \cdot \Sigma_k\mathsf{P}$ and $\Delta_0\mathsf{P} = \mathsf{P}$, $\Delta_{k+1}\mathsf{P} = P^{\Sigma_k\mathsf{P}}$. PH

is contained in MP [**GKR$^+$95**] and PP/poly with respect to every oracle, as well as SPP with respect to the random oracle [**For99**]. On the other hand, $\mathsf{PH} \not\subseteq \Delta_3\mathsf{P}$ with respect to the random oracle, and the question of whether $\mathsf{PH} \subseteq \Delta_3\mathsf{P}$ unrelativized is open.

4.2.43. **PostBPP.** The definition of this class is similar to that of BPP, except we allow for *postselection*. Let (M,p) be a polynomial-time Turing machine and $O(n^*)$-function, respectively, such that

$$\Pr_{y\in\Sigma^{p(|x|)}}[M(x,y)=1]>0$$

for every $x\in\Sigma^*$. Then $\mathscr{L}\in\mathsf{PostBPP}$ if and only if there exists a polynomial-time Turing machine M' such that for all $x\in\Sigma^*$,

$$\Pr_{y\in\Sigma^{p(|x|)}}[M'(x,y)=\mathscr{L}(x)|M(x,y)=1]\geq 2/3.$$

The condition $M(x,y)=1$ effectively allows us to consider BPP-algorithms in which there is an option for the program to *quit* before concluding $x\in\mathscr{L}$ or $x\notin\mathscr{L}$. The algorithm is considered successful if it has a high probability of reaching the correct answer when it does *not* quit.

PostBPP is contained in $\Delta_2\mathsf{P}$ with respect to the random oracle and $\Delta_3\mathsf{P}$ and PP with respect to all oracles. The question of whether $\mathsf{PostBPP}\subseteq\Sigma_2\mathsf{P}$ in the unrelativized world is open.

4.2.44. **PostBQP.** This class is the quantum analogue of PostBPP. For a BQP-machine M and input x, denote the value of the kth qubit of the output by $M(x)_k$, $\mathscr{L}\in\mathsf{PostBQP}$ if and only if there exists a BQP-machine M such that for every $x\in\Sigma^*$, $\Pr[M(x)_1=1]>0$ and

$$\Pr[M(x)_0=\mathscr{L}(x)|M(x)_1=1]\geq 2/3.$$

It has been shown that $\mathsf{PostBQP}=\mathsf{PP}$ for every oracle [**Aar05**].

4.2.45. **PP.** Define PP to be $\mathcal{P}\cdot\mathsf{P}$. Like BPP, PP is based on a probabilistic model of computation, except that the machine is only required to obtain the correct answer with a probability of at least $1/2$.

$\mathsf{PP}\subseteq\mathsf{MP}$ with respect to every oracle. Also, $\mathsf{PP}\subseteq\oplus\mathsf{P}$ with respect to the random oracle [**Raz87**].

4.2.46. **PP/poly.** This class is simply PP with polynomial advice: in operator terms, $\mathsf{PP/poly} = \mathit{poly}\cdot\mathcal{P}\cdot\mathsf{P}$.

4.2.47. $\mathsf{PPSPACE}$**.** This is a probabilistic version of PSPACE: $\mathsf{PPSPACE} = \mathcal{P} \cdot \mathsf{PSPACE}$. Ladner [**Lad89**] showed that $\mathsf{PPSPACE} = \mathsf{PSPACE}$, and this result holds with respect to every oracle. (In fact, Ladner proved that the functional counterparts of these classes, $\#\mathsf{PSPACE}$ and $\mathsf{FPSPACE}$, are equal.)

4.2.48. PSPACE**.** A PSPACE-machine is a Turing machine in which the number of cells on each tape is limited to some $O(n^*)$ function. PSPACE is also equal to public-coin RG [**Pap85**], so $\mathsf{PSPACE} \subseteq \mathsf{RG}$ with respect to every oracle.

PSPACE is contained in IP with respect to any algebraic oracle [**AW09**] and $\mathsf{RG}(2)$ in the unrelativized world [**FK97**]. $\mathsf{PSPACE} \not\subseteq \mathsf{PH}$ with respect to the random oracle [**Cai86**].

As with many important complexity classes, it is an open question whether PSPACE is any larger than P with respect to the trivial oracle. In the world of all oracles, the inclusion $\mathsf{PSPACE} \subseteq \mathsf{CH}$ is also open.

4.2.49. $\mathsf{PSPACE/poly}$**.** This is PSPACE with polynomial advice: $\mathsf{PSPACE/poly} = \mathit{poly} \cdot \mathsf{PSPACE}$.

4.2.50. QAM**.** The *quantum Arthur-Merlin protocol* is the same as the classical Arthur-Merlin protocol, except Arthur is a BQP-machine instead of a classical polynomial-time Turing machine, and Merlin provides a quantum certificate.

QAM is contained is contained in CH, $\mathsf{PP/poly}$, QIP, $\mathsf{QIP}(2)$, $\mathsf{QMA/mpoly}$, and $\mathsf{QRG}(2)$ with respect to every oracle. In the random oracle world, QAM and QMA are equal. In the unrelativized world, the inclusion $\mathit{cocap} \cdot \mathsf{QAM} \subseteq \mathsf{P}^{\#\mathsf{P}}$ is open.

4.2.51. QCMA**.** The *quantum-classical Merlin-Arthur protocol* is the same as the protocol used in QMA, except that Merlin's proof must be a classical bit string. It is immediate that $\mathsf{QCMA} \subseteq \mathsf{QMA}$ and $\mathsf{QCMA} \subseteq \mathsf{QCMA/qpoly}$ with respect to every oracle. The inclusion $\mathsf{QCMA} \subseteq \mathit{co} \cdot \mathsf{A_0PP}$ is open in the random oracle world.

4.2.52. $\mathsf{QCMA/qpoly}$**.** The class $\mathsf{QCMA/qpoly}$ is QCMA with a quantum advice string, similar to $\mathsf{BQP/qpoly}$. It is contained in $\mathsf{QMA/mpoly}$ for every oracle [**AD14**].

4.2.53. QIP**.** In the quantum interactive proof protocol, the verifier is a polynomial-time quantum computer, and the prover is a quantum computer with unlimited computational power, represented formally by a family of quantum circuits with no additional restrictions. The verifier and prover exchange quantum messages for a number of rounds that is a polynomial of the length of the input.

For every oracle, we have the immediate inclusions

$$\mathsf{QIP} \subseteq \mathsf{QRG}, \mathsf{QMIP}_{le}, \mathsf{QMIP}_{ne},$$

as well as $\mathsf{QIP} \subseteq \mathsf{PSPACE}$ [**JJUW10**].

4.2.54. $\mathsf{QIP}(2)$**.** This class is the special case of QIP in which two messages are exchanged. Of course, $\mathsf{QIP}(2) \subseteq \mathsf{QIP}(3)$ for every oracle.

4.2.55. $\mathsf{QIP}(3)$**.** The three-message version of QIP is equivalent to the version that allows a polynomial number of rounds. That is, $\mathsf{QIP}(3) = \mathsf{QIP}$ for every oracle [**MW05**].

4.2.56. QMA**.** The quantum Merlin-Arthur protocol is the same as the classical Merlin-Arthur protocol, but Arthur is a polynomial-time quantum computer, and the proof that Merlin sends to Arthur is quantum.

SMA is contained in QAM, $\mathsf{QMA}(2)$, $\mathsf{QMA/mpoly}$, $\mathsf{QRG}(1)$, and SBQP with respect to every oracle. It is open whether there exists an oracle relative to which QMA has a computational advantage over QCMA. In the unrelativized world, the inclusions $\mathsf{QMA} \subseteq \mathsf{QCMA}$ and $\mathit{cocap} \cdot \mathsf{QMA} \subseteq \mathsf{QCMA/qpoly}$ are open.

4.2.57. $\mathsf{QMA}(2)$**.** In this variant of QMA, Merlin sends Arthur two unentangled quantum messages rather than one. $\mathsf{QMA}(2)$ is contained in EXP_{plo} and QMIP_{ne} for all oracles. It is open whether there exists an oracle making QMA and $\mathsf{QMA}(2)$ distinct or making $\mathsf{QMA}(2)$ and QCMA distinct. In the unrelativized world, the inclusions $\mathsf{QMA}(2) \subseteq \mathsf{P}$, $\mathit{cocap} \cdot \mathsf{QMA}(2) \subseteq \mathsf{PSPACE/poly}$, and $\mathit{cocap} \cdot \mathsf{QMA}(2) \subseteq \mathsf{RG}(3)$ are open.

4.2.58. $\mathsf{QMA/mpoly}$**.** This class is QMA with polynomial-length advice, which is "Merlinized" in the sense of $\mathsf{BQP/mpoly}$. It is contained in $\mathsf{PP/poly}$ and $\mathsf{QMA/qpoly}$ with respect to every oracle.

4.2.59. $\mathsf{QMA/qpoly}$**.** This class is QMA with polynomial-length quantum advice. It is contained in $\mathsf{PSPACE/poly}$ with respect to every oracle.

Since $\mathsf{QMA} \subseteq \mathsf{QCMA}$ is open in the world of all oracles, so is $\mathsf{QMA/qpoly} \subseteq \mathsf{QCMA/qpoly}$. In the random oracle world, $\mathsf{QMA/qpoly} \subseteq \mathsf{NP/poly}$ and $\mathit{cocap} \cdot \mathsf{QMA/qpoly} \subseteq \mathsf{P/poly}$ are open, and $\mathit{cocap} \cdot \mathsf{QMA/qpoly} \subseteq \mathsf{PP/poly}$ is open in the unrelativized world.

4.2.60. QMIP$_{le}$**.** QMIP and its related classes are defined similarly to MIP, except that there may be a number of provers bounded by a polynomial of the length of the input, the verifier is a polynomial-time quantum computer, and the provers are quantum computers with infinite computational power. Because the provers are quantum, they may share entangled qubits before the protocol begins. The *le* in QMIP$_{le}$ stands for "limited entanglement;" here, the provers are required to share at most a polynomially limited number of entangled qubits.

QMIP$_{le} \subseteq$ NEXP$_{plo}$ with respect to every oracle. The inclusions QMIP$_{le} \subseteq$ IP is open in the world of all oracles, as are QMIP$_{le} \subseteq$ NP and QMIP$_{le} \subseteq$ $co\cdot$NEXP in the random oracle world and $cocap\cdot$QMIP$_{le} \subseteq$ EXP in the trivial oracle world.

4.2.61. QMIP$_{ne}$**.** This is defined in the same manner as QMIP$_{le}$, except the provers are not allowed to share any entangled qubits. QMIP$_{ne} \subseteq$ NEXP$_{plo}$ with respect to every oracle. The inclusions QMIP$_{ne} \subseteq$ NP and QMIP$_{ne} \subseteq$ IP are open in the random and all oracle worlds, respectively.

4.2.62. QRG**.** This class is the quantum version of RG, in which the verifier is a polynomial-time quantum Turing machine and the prover and disprover are quantum computers with infinite computational power. Unlike QMIP, there are no variants of this class based on entangled qubits, because the prover and disprover are working against each other and so there is no benefit in sharing entangled qubits.

This class equals EXP$_{plo}$ with respect to every oracle [**GW07**].

4.2.63. QRG(1)**.** QRG(1) is QRG with the additional limitation that the prover and disprover can only send one message to the verifier. The verifier sends no messages. The class is, of course, contained in the two-round version QRG(2) with respect to every oracle. In the unrelativized world, it is open whether QRG(1) is contained in CH or PP/poly.

4.2.64. QRG(2)**.** This class is QRG with two rounds: first, the verifier sends separate messages to the prover and disprover, and then the prover and disprover each send one response. The class is immediately contained in QRG with respect to every oracle, and it is also contained in PSPACE with respect to every oracle.

4.2.65. QSZK**.** This is the quantum counterpart to SZK. The verifier is a polynomial-time quantum computer, and the prover is an all-powerful quantum computer. As in the classical case, we require that the verifier's view of his interaction be statistically indistinguishable from one that the verifier could generate himself. The view, in this case, is a transcript of the quantum states the verifier sends and receives.

$\mathsf{QSZK} \subseteq \mathsf{QIP}(2)$ and $\mathsf{QSZK} \subseteq \mathsf{PSPACE}$ for all oracles. $\mathsf{QSZK} \subseteq \mathsf{P}$ and $\mathsf{QSZK} \subseteq \mathit{cocap} \cdot \mathsf{NIQSZK}$ in the random oracle world and all oracle world, respectively.

4.2.66. R**.** The class R is the foundational class of computability theory. It is the class of all languages $\mathscr{L}$ such that $\mathscr{L}(x) = M(x)$ for all $x \in \Sigma^*$ and some Turing machine M. Since R is not limited except by the requirement that languages be computable, it absorbs most of the operators used in Complexity Zoology: $\oplus$, $\mathcal{BP}$, $\mathcal{P}$, $\mathcal{N}$, $(\mathsf{C} \mapsto \mathsf{P}^{\mathsf{C}})$. Diagonalization arguments show that it is larger than most of the time-limited models of computation.

$$\mathsf{R} \not\subseteq \mathsf{MAEXP}, \mathsf{EXP}^{\mathsf{NP}}, \mathsf{NEXP/poly}$$

with respect to every oracle.

4.2.67. RE**.** RE is the set of all recursively enumerable languages. A language is *recursively enumerable* if it is the range of a computable function $f : \mathbb{N} \to \Sigma^*$. Equivalently, RE is the set of all languages $\mathscr{L}$ such that for every $x \in \mathscr{L}$,

$$x \in \mathscr{L} \Longrightarrow M(x) = 1.$$

The famously non-computable halting problem is in RE, and therefore RE and R are distinct. Since $\mathsf{R} = \mathit{cocap} \cdot \mathsf{RE}$, it follows that RE is not a symmetric class [**Tur37**].

4.2.68. RG**.** A *refereed game* involves an interaction between three parties: a prover, a disprover, and a verifier. The verifier is a probabilistic polynomial-time Turing machine, while the prover and disprover have infinite computational power, although they are unable to see the verifier's coin tosses and are similarly isolated from each other. The verifier exchanges messages with the prover and disprover in parallel for a number of rounds that is a polynomial of the length of the input. When the interaction ends, the verifier decides to accept or reject the input. A language lies in RG if and only if there is a refereed game in which optimal play convinces the verifier of the correct answer with a probability of at least $2/3$.

$\mathsf{RG} = \mathsf{EXP}_{plo}$ for all oracles.

4.2.69. $\mathsf{RG}(1)$**.** This is the version of RG that is restricted to one round of interaction, in which the prover and disprover each send a message to the verifier. The verifier does not send any messages to the prover and the disprover. The class is contained in $\mathsf{QRG}(1)$, and $\mathsf{RG}(2)$ for every oracle, and it is equal to $\mathsf{S_2P}$ for every oracle [**FIKU08**].

4.2.70. $\mathsf{RG}(2)$**.** This is the version of RG that is restricted to two rounds of interaction: first, the verifier sends separate messages to the prover and the disprover; then the prover and disprover send a response to the verifier. It is contained in PSPACE, $\mathsf{QRG}(2)$, and $\mathsf{RG}(3)$ for every oracle.

4.2.71. $\mathsf{RG}(3)$**.** This is the version of RG that is restricted to three rounds of interaction. In round one, the prover and disprover separately send a message to the verifier. In round two, the verifier sends one reply to the prover and one reply to the disprover. Finally, the prover and disprover each respond with another message to the verifier. It is open whether $\mathsf{RG}(3)$ is contained in PSPACE/poly or P with respect to the trivial oracle.

4.2.72. RP**.** RP is another probabilistic model of computation. $\mathscr{L} \in \mathsf{RP}$ if and only if there exists a polynomial-time Turing machine M and a function $p(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \implies \Pr_{y \in \Sigma^{p(|x|)}}[M(x,y) = 1] \geq 2/3,$$

$$x \notin \mathscr{L} \implies \Pr_{y \in \Sigma^{p(|x|)}}[M(x,y) = 1] = 0.$$

$\mathsf{RP} \subseteq \mathsf{NP}$ is immediate. We can also inflate the probability of $M(x,y) = 1$ when $x \in \mathscr{L}$ by running multiple RP-machines in parallel. Therefore, $\mathsf{RP} \subseteq \mathsf{BPP}$ as well.

$\mathsf{RP} \subseteq \mathsf{NP}$ is immediate. We can also inflate the probability of $M(x,y) = 1$ when $x \in \mathscr{L}$ by running multiple RP-machines in parallel. Therefore, $\mathsf{RP} \subseteq \mathsf{BPP}$ for every oracle as well.

There is an oracle relative to which $\mathsf{RP} \not\subseteq \mathsf{coNP}$ and an algebraic oracle relative to which $\mathsf{RP} \not\subseteq \mathsf{P}$. In the unrelativized world, the inclusions $\mathsf{RP} \subseteq \mathsf{coNP}$ and $\mathsf{RP} \subseteq \mathsf{P}$ are both open.

4.2.73. $\mathsf{S_2P}$**.** $\mathsf{S_2P}$ is the second level of the *symmetric hierarchy*. The definition is similar to that of $\Sigma_2\mathsf{P}$, except that the ”no” condition is altered to make the class symmetric. $\mathscr{L} \in \mathsf{S_2P}$ if and only if there exists a polynomial-time Turing Machine M and functions $p(n), q(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \implies (\exists y \in \Sigma^{p(|x|)})(\forall z \in \Sigma^{q(|x|)})[M(x,y) = 1],$$

$$x \notin \mathscr{L} \implies (\exists y \in \Sigma^{p(|x|)})(\forall z \in \Sigma^{q(|x|)})[M(x,y) = 0].$$

In addition to being contained in EXP, QRG, and $\Sigma_2\mathsf{P}$ for every oracle, $\mathsf{S_2P}$ is contained in $\Delta_2\mathsf{P}$ with respect to the random oracle.

4.2.74. SBP**.** $\mathscr{L} \in \mathsf{SBP}$ if and only if there exist functions $f \in \#\mathsf{P}$, $g \in \mathsf{FP}$ from Σ^* to $\mathbb{N}$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \Longrightarrow f(x) > g(x)$$
$$x \notin \mathscr{L} \Longrightarrow f(x) > g(x)/2.$$

$\mathsf{SBP} \subseteq \mathsf{AM}, \mathsf{PostBPP}, \mathsf{SBQP}$ for every oracle. There exist oracles relative to which $\mathsf{SBP} \not\subseteq \Sigma_2\mathsf{P}$ [**BGM03**], $\mathit{cocap} \cdot \mathsf{SBP} \not\subseteq \mathsf{QMA}$ [**AKKT19**], and $\mathit{cocap} \cdot \mathsf{SBP} \not\subseteq \mathsf{S_2P}$. In the unrelativized world, the inclusions $\mathsf{SBP} \subseteq \Sigma_2\mathsf{P}$ and $\mathit{cocap} \cdot \mathsf{SBP} \subseteq \mathsf{QMA}(2), \mathsf{QRG}(1)$ are open.

4.2.75. SBQP**.** $\mathscr{L} \in \mathsf{SBQP}$ if and only if there exists a polynomial-time quantum computer M and a function $p(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \Longrightarrow \Pr[M(x) = 1] \geq 2^{-p(|x|)},$$
$$x \notin \mathscr{L} \Longrightarrow \Pr[M(x) = 1] \leq 2^{-p(|x|)-1}.$$

$\mathsf{SBQP} = \mathsf{A_0PP}$ for every oracle [**Kup09**].

4.2.76. $\Sigma_2\mathsf{EXP}$**.** By definition, $\Sigma_2\mathsf{EXP} = \mathsf{EXP}^{\mathsf{NP}}$.

4.2.77. $\Sigma_2\mathsf{P}$**.** The second level of the polynomial hierarchy is $\Sigma_2\mathsf{P} = \mathsf{NP}^{\mathsf{NP}}$. Thus, $\Sigma_2\mathsf{P} = \mathcal{N} \cdot \mathsf{P}^{NP} = \mathcal{N} \cdot \Delta_2\mathsf{P}$. Via the alternating quantifier definition of $\Sigma_2\mathsf{P}$, we also have $\Sigma_2\mathsf{P} = \mathcal{N} \cdot \mathit{co} \cdot \mathsf{NP}$. The inclusions $\Sigma_2\mathsf{P} \subseteq \Delta_3\mathsf{P}, \mathsf{RG}(3)$ with respect to every oracle are immediate. $\Sigma_2\mathsf{P}$ is asymmetric with respect to the random oracle [**RST15**].

4.2.78. SPP**.** SPP stands for stoic-PP, and it is one of three variants of PP named after ancient schools of philosophy. In epicurean-PP, the condition leading to a PP-machine rejecting the input is strengthened so that the number of rejecting computational paths is only slightly greater than the number of accepting paths. Meanwhile, cynical-PP = *co*·epicurean-PP. Stoic-PP combines the traits of epicurean-PP and cynical-PP: acceptance requires that exactly half of all paths accept, and rejection requires that slightly fewer than half of all paths accept [**For02**]. Thus, $\mathscr{L} \in \mathsf{SPP}$ if and only if there exists a polynomial-time Turing machine M and a function $p(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$x \in \mathscr{L} \Longrightarrow |\{y \in \Sigma^{p(|x|)} : M(x,y) = 1\}| = |\{y \in \Sigma^{p(|x|)} : M(x,y) = 0\}|,$$
$$x \notin \mathscr{L} \Longrightarrow |\{y \in \Sigma^{p(|x|)} : M(x,y) = 0\}| - |\{y \in \Sigma^{p(|x|)} : M(x,y) = 1\}| = 2.$$

It follows that $\mathsf{SPP} \subseteq \oplus\mathsf{P}, \mathsf{AWPP}$ for every oracle. $\mathsf{SPP} \not\subseteq \mathsf{PH}$ with respect to the random oracle. The inclusions $\mathsf{SPP} \subseteq \mathsf{PH}$ and $\mathsf{SPP} \subseteq \mathsf{QRG}(1)$ are open with respect to the trivial oracle.

4.2.79. SZK**.** In a statistical zero-knowledge proof protocol, a probabilistic polynomial-time verifier exchanges a polynomial number of messages with an all-powerful prover, who attempts to convince the verifier that the input x lies in a given language $\mathscr{L}$. However, the verifier's view of his interaction with the prover—consisting of his random bits, the messages he sends, and the messages he receives—must be statistically indistinguishable from one that he could generate himself. In other words, the prover tries to convince the verifier that $x \in \mathscr{L}$ while leaving the verifier (statistically) unable to convince anyone else that $x \in \mathscr{L}$.

To rigorously define SZK in the standard way, it is necessary to formalize the concept of an interaction between the verifier and prover. Let $f, g : \Sigma^* \to \Sigma^*$. For $x \in \Sigma^*$, define the sequence $(f,g)(x) = \{a_j\}_{j=0}^{\infty}$ by recursion:

$$a_0 = x,$$
$$a_{j+1} = \begin{cases} f(a_0, \ldots, a_j) & \text{if } j \text{ is even,} \\ g(a_0, \ldots, a_j) & \text{if } j \text{ is odd.} \end{cases}$$

Set $(f,g)_k(x) = \langle a_0, \ldots, a_k \rangle$ for every positive integer k, and set

$$\mathrm{out}_f(f,g)_k(x) = f(x, a_1, \ldots, a_k),$$
$$\mathrm{out}_g(f,g)_k(x) = g(x, a_1, \ldots, a_k).$$

We also need the concept of the *view* that the verifier has of its interaction with the prover. For $y \in \Sigma^*$, let $f_y : \Sigma^* \to \Sigma^*$ denote the function $f_y(a) = f(a,y)$. Then set

$$\mathrm{View}_f(f,g)_k(x,y) = \langle (f_y, g)_k(x), y \rangle \,,$$

$$\mathrm{View}_g(f,g)_k(x,y) = \langle (f, g_y)_k(x), y \rangle \,.$$

Let P denote a function from Σ^* to the set of probability distributions on Σ^*, so that $P(x)$ can be regarded as being randomly chosen. P represents the prover's strategy, which may include coin tosses. P is *statistically zero-knowledge* if for every polynomial-time Turing machine V^* and functions $p^*(n), q^*(n) = O(n^*)$ there exists a polynomial-time Turing machine S and a function $r(n) = O(n^*)$ such that for every $x \in \Sigma^*$,

$$\frac{1}{2} \sum_{\alpha \in \Sigma^*} \left| \Pr[\mathrm{View}_{V^*}(P, V^*)_{q^*(|x|)}(x,y) = \alpha] - \Pr[S(x,z) = \alpha] \right| < \varepsilon(|x|),$$

where $y \in \Sigma^{p^*(|x|)}, z \in \Sigma^{r^*(|x|)}$ are randomly chosen and $\varepsilon : \mathbb{N} \to \mathbb{R}$ is a *negligible function*, meaning that for any $k \in \mathbb{N}$, $\varepsilon(n) < 1/n^k$ when n is sufficiently large. (Note that the sum above is finite, because possible values of α are bounded in length by a polynomial in $|x|$.) This definition is intended to capture the idea that the verifier's view of the interaction between himself and the prover should be statistically indistinguishable from one that the verifier could have generated himself. Let ZK denote the set of all P that are statistically zero-knowledge.

We can now define a *statistical zero-knowledge proof protocol* (V, p, q) for a language $\mathscr{L} \subseteq \Sigma^*$. V is a polynomial-time Turing machine and $p(n), q(n) = O(n^*)$. For every $x \in \Sigma^*$,

$$x \in \mathscr{L} \implies (\exists P \in ZK)[\Pr[\mathrm{out}_V(P, V_y)_{q(|x|)}(x) = 1] \geq 2/3],$$

$$x \notin \mathscr{L} \implies (\forall P \in ZK)[\Pr[\mathrm{out}_V(P, V_y)_{q(|x|)}(x) = 1] \leq 1/3]$$

where $y \in \Sigma^{p(|x|)}$ is randomly chosen. Then define SZK to be the complexity class such that $\mathscr{L} \in \mathsf{SZK}$ if and only if there exists a statistical zero-knowledge protocol (V, p, q) for $\mathscr{L}$. SZK^f for an oracle f is defined in the same way, except the computational model underlying the protocol is a Turing machine with f-oracle.

This definition is quite different from (and considerably more complex than) the definition of SZK we gave in Section 4.1, which was based on a *rhetorical question protocol*. Denote this more simply defined SZK by AM_{rhet}.

THEOREM 4.2.3 (Kuperberg). $\mathsf{SZK} = \mathsf{AM}_{rhet}$ *with respect to every oracle.*

PROOF. The rhetorical question protocol is a special case of the statistical zero-knowledge proof protocol: since Arthur knows the correct answer to his question, he can produce a transcript of his interaction with Merlin without Merlin's assistance. Hence $\mathsf{AM}_{rhet} \subseteq \mathsf{SZK}$. (Here we implicitly assume an *honest verifier*. A dishonest Arthur could attempt to violate statistical zero-knowledge by breaking the rules of the protocol. However, Okamoto has shown that it suffices to assume an honest verifier in the definition of SZK [**Oka00**].)

For the inclusion $\mathsf{SZK} \subseteq \mathsf{AM}_{rhet}$, we use the SZK-complete promise problem of Vadhan [**SV03**]. A *promise problem* is a special kind of decision problem in which not every string $x \in \Sigma^*$ necessarily corresponds to an answer. A promise problem can be written as (Y,N), where Y and N are disjoint subsets of Σ^* representing answers of yes and no, respectively. The relevant promise problem `SD` (`Statistical Difference`) consists of pairs of circuits $\langle C_1, C_2 \rangle$ with the same output length. If the output length is n, let X_1, X_2 be the probability distributions on Σ^n that result from computing $C_1(u), C_2(u)$, respectively, where u is an input chosen uniformly at random. The answer to the problem is yes when X_1 and X_2 have a statistical distance of at least $2/3$, and the answer to the problem is no when X_1 and X_2 have a statistical distance of at most $1/3$.

`SD` can be computed using a rhetorical question protocol as follows. Arthur chooses X_1 or X_2 at random and uses the chosen random variable to create a sequence of strings in Σ^n. He presents the sequence to Merlin and asks which of the two distributions produced the sequence. If the answer to the input is yes with respect to `SD`, then Merlin can distinguish between X_1 and X_2 with high probability, while if the answer is no then X_1 and X_2 are too close for Merlin to reliably distinguish between X_1 and X_2.

It follows from the SZK-completeness of `SD` and the closure of AM_{rhet} under polynomial-time reductions that $\mathsf{SZK} \subseteq \mathsf{AM}_{rhet}$. □

SZK is contained in AM and QSZK for every oracle. On the other hand, there is an oracle relative to which $\mathsf{SZK} \not\subseteq \mathsf{BQP}$ [**Aar02**]. It is open whether there is an oracle relative to which $\mathsf{SZK} \subseteq \mathsf{S_2P}$.

4.2.80. UP. *Unambiguous polynomial time* is a special case of non-deterministic polynomial time. $\mathscr{L} \in \mathsf{UP}$ if and only if there is a polynomial-time Turing machine M and a function $p(n) = O(n^*)$ such that for all $x \in \Sigma^*$,

$$x \in \mathscr{L} \implies (\exists! y \in \Sigma^{p(|x|)})[M(x,y) = 1],$$

$$x \notin \mathscr{L} \implies (\forall y \in \Sigma^{p(|x|)})[M(x,y) = 0].$$

Hence $\mathsf{UP} \subseteq \mathsf{SPP}$ and $\mathsf{UP} \subseteq \mathsf{NP}$ with respect to every oracle.

There exist oracles relative to which $\mathit{cocap} \cdot \mathsf{UP}$ is not contained in $\mathsf{BQP/qpoly}$, $\mathsf{P/poly}$, and QSZK. With respect to the random oracle, UP is not contained in $\mathit{co} \cdot \mathsf{QMIP}_{le}$, $\mathit{co} \cdot \mathsf{QMIP}_{ne}$, and $\mathit{co} \cdot \mathsf{QMA/qpoly}$. In the unrelativized world, the following are open: $\mathsf{P} = \mathsf{UP}$, $\mathsf{UP} \subseteq \mathit{co} \cdot \mathsf{QIP}(2)$, $\mathsf{UP} \subseteq \mathit{co} \cdot \mathsf{QMA}(2)$, $\mathsf{UP} \subseteq \mathit{co} \cdot \mathsf{QMA/qpoly}$, $\mathit{cocap} \cdot \mathsf{UP} \subseteq \mathsf{BQP/qpoly}$, and $\mathit{cocap} \cdot \mathsf{UP} \subseteq \mathsf{QSZK}$.

4.2.81. ZPP. A pair (M, f) denotes an *expected polynomial-time Turing machine* if M is a Turing machine and $f : \mathbb{N} \to \Sigma^*$ such that

$$E(x) = 2^{-f(|x|)} \sum_{y \in \Sigma^{f(|x|)}} T_M(x,y)$$

satisfies $\max_{|x|=n} E(x) = O(n^*)$, where $T_M(x,y)$ indicates the computation time of the Turing machine M with input $\langle x, y \rangle$. $\mathscr{L} \in \mathsf{ZPP}$ if and only if there exists an expected polynomial-time Turing machine (M, f) such that for every $x \in \Sigma^*$,

$$(\forall y \in \Sigma^{f(|x|)})[M(x,y) = \mathscr{L}(x)].$$

There is an oracle relative to which $\mathsf{ZPP} \not\subseteq \oplus\mathsf{P}$ [**BBF98**], but it is open whether this is true in the unrelativized world.

4.3. Conclusion and Possible Future Work

By now we have seen the power and limitations of Complexity Zoology. Along the way, also outlined the landscape of complexity theory. We discussed class operators and the role they play in the Zoology program. We explained the software's logic and how it can infer whether an inclusion is proven, disproven, or open. We explored a simplified version of the system's data set with a reduced number of complexity classes to consider. Finally, we surveyed all the classes currently in the project while highlighting some of the key definitions and results.

Just as an inquisitive student can inspire new lines of thought with an insightful question, Complexity Zoology has brought attention to some challenging but not widely studied open problems. Completing the system's data set by identifying which inclusions should be regarded as truly open and which inclusions can be marked as proven or disproven will comprise a large amount of the potential future work on the project. Another possibility for further expanding Complexity Zoology is considering a different collection of complexity classes or of different oracle worlds. Several results in complexity theory are conditioned on a possible outcome of an open problem: for example: "If $\mathsf{P} = \mathsf{NP}$, then..." Such results could be studied with Complexity Zoology's help by including an "if $\mathsf{P} = \mathsf{NP}$" or similar world.

We have been careful to emphasize that Complexity Zoology does not understand complexity theory as such. Even so, the software could be a helpful tool in a true formalization of complexity theory based on computer-assisted of computer-verified proofs. In particular, Zoology's ability to weed out redundant inclusions or separations could be an invaluable time-saver.

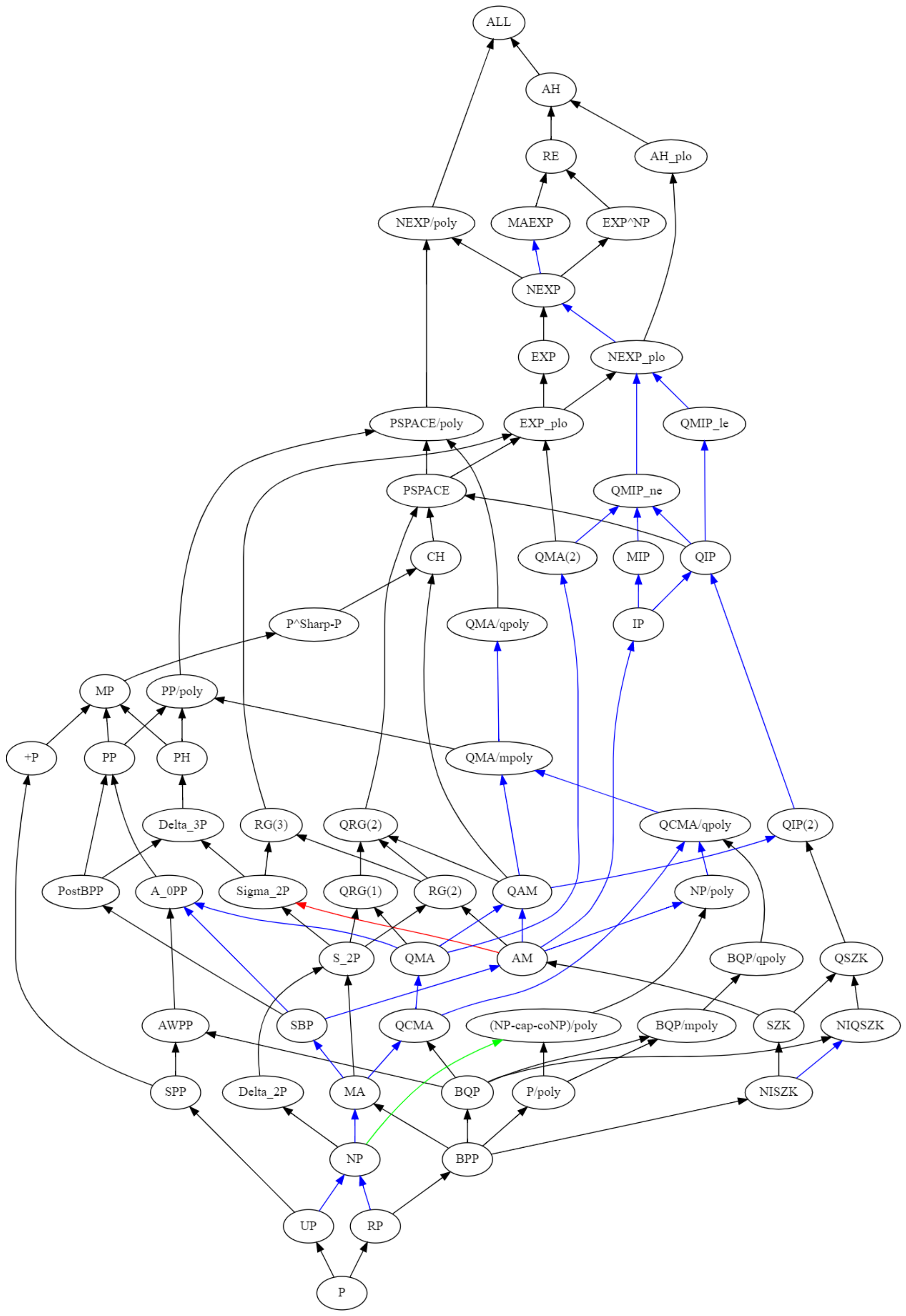

FIGURE 4.2. Inclusions that hold with respect to every oracle. Blue arrows denote containment, black arrows denote symmetric containment, red arrows indicate that the complement of the first class is contained in the second, and green arrows indicate that the intersection of the first class with its complement is contained in the second class.

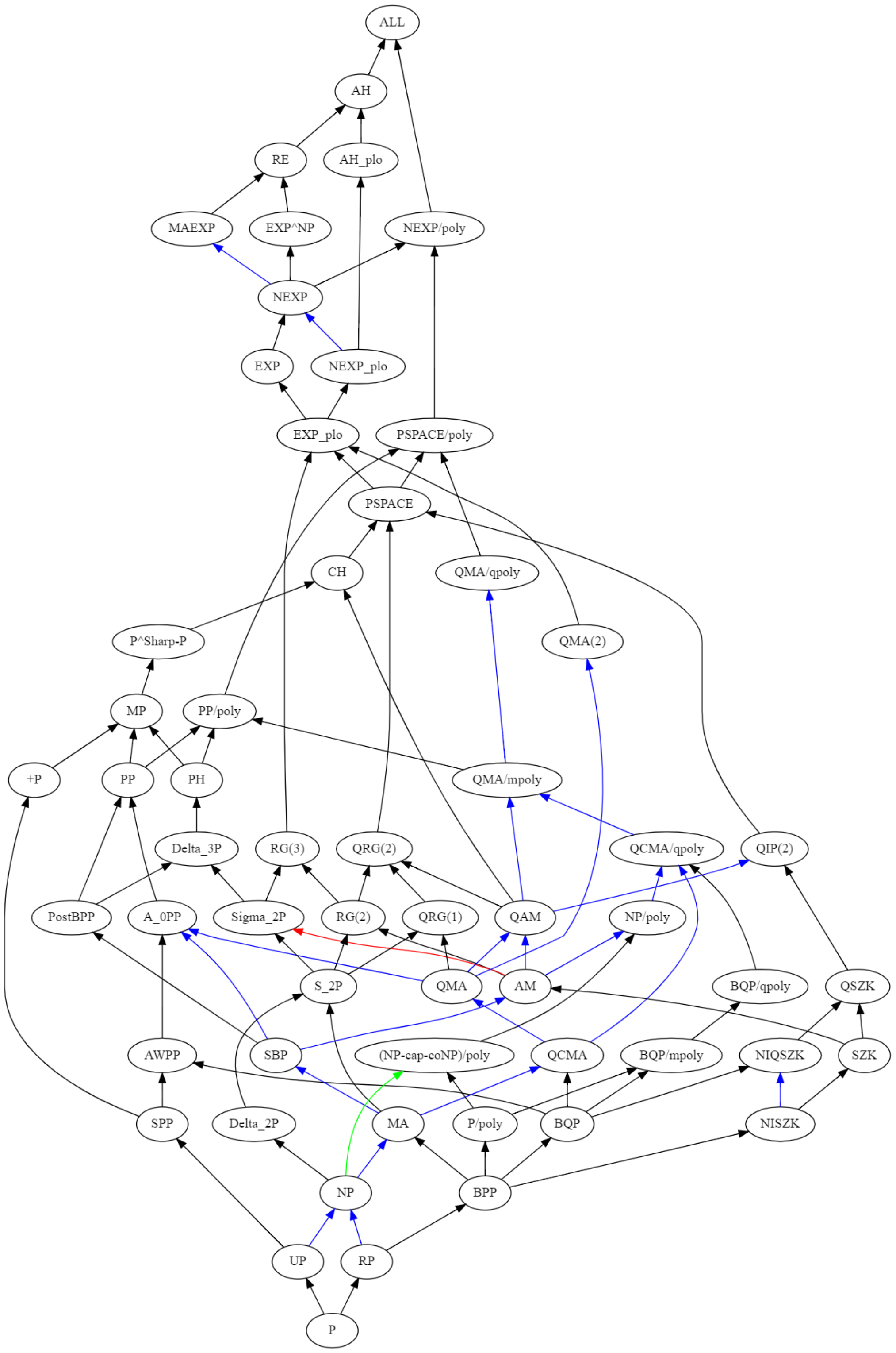

FIGURE 4.3. Inclusions that hold with respect to all algebraic oracles.

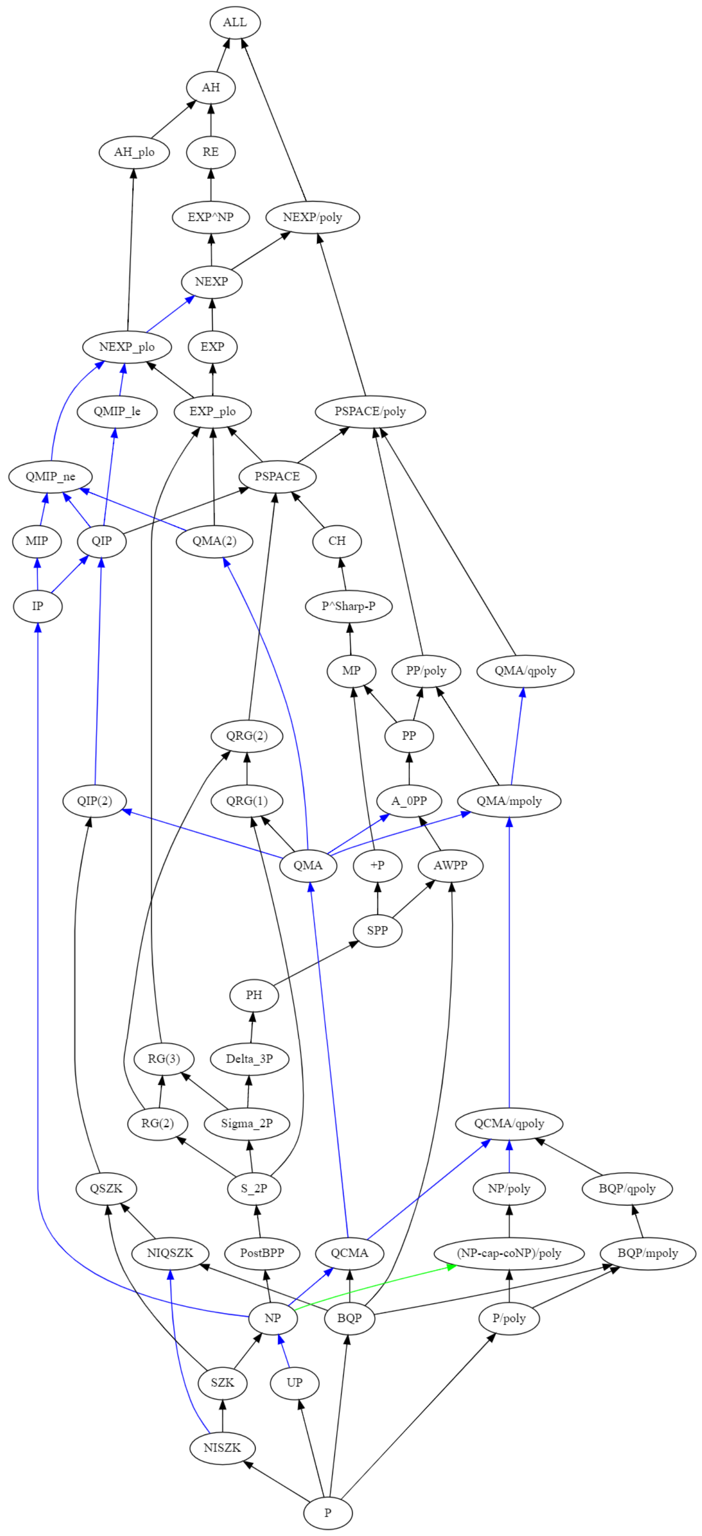

FIGURE 4.4. Inclusions that hold with respect to the random oracle.

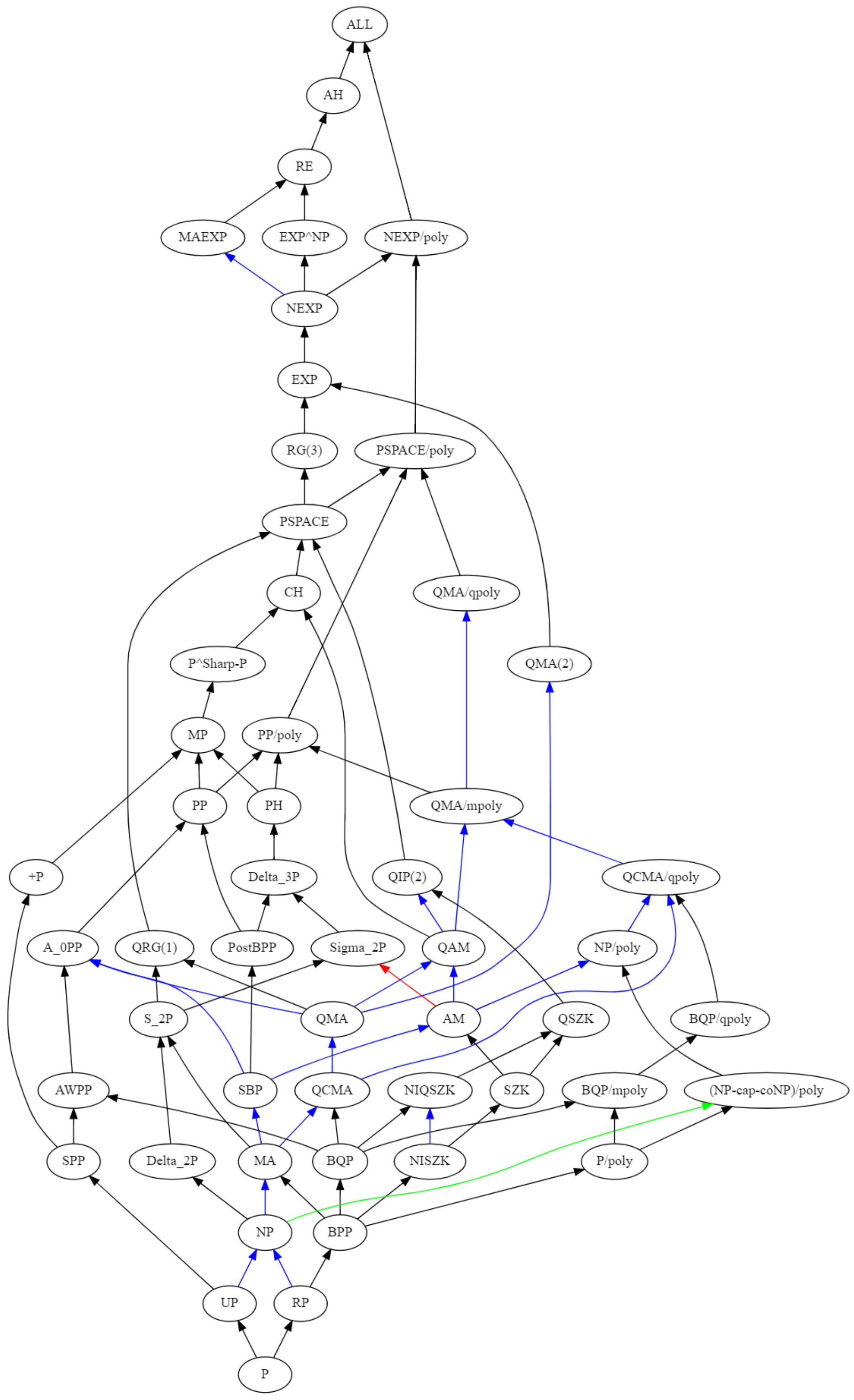

FIGURE 4.5. Inclusions that hold with respect to the trivial oracle (the unrelativized world).

Bibliography

[AA09] S. Aaronson and A. Ambainis, *The need for structure in quantum speedups*, arXiv preprint arXiv:0911.0996 (2009).

[Aar02] S. Aaronson, *Quantum lower bound for the collision problem*, Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, ACM, 2002, pp. 635–642.

[Aar04] ______, *Limitations of quantum advice and one-way communication*, Computational Complexity, 2004. Proceedings. 19th IEEE Annual Conference on, IEEE, 2004, pp. 320–332.

[Aar05] ______, *Quantum computing, postselection, and probabilistic polynomial-time*, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, vol. 461, The Royal Society, 2005, pp. 3473–3482.

[Aar06] ______, *Oracles are subtle but not malicious*, 21st Annual IEEE Conference on Computational Complexity (CCC'06), IEEE, 2006, pp. 15–pp.

[Aar10] ______, *BQP and the polynomial hierarchy*, Proceedings of the forty-second ACM symposium on Theory of computing, ACM, 2010, pp. 141–150.

[Aar19] ______, *Complexity Zoo*, `https://complexityzoo.uwaterloo.ca/Complexity_Zoo`, 2019.

[AB09] S. Arora and B. Barak, *Computational complexity: A modern approach*, 1st ed., Cambridge University Press, New York, NY, USA, 2009.

[AB18] B. Aydinlioğlu and E. Bach, *Affine relativization: Unifying the algebrization and relativization barriers*, ACM Trans. Comput. Theory **10** (2018), no. 1, 1:1–1:67.

[AD10] S. Aaronson and A. Drucker, *A full characterization of quantum advice*, 2010, 1004.0377.

[AD14] S. Aaronson and A. Drucker, *A full characterization of quantum advice*, SIAM Journal on Computing **43** (2014), no. 3, 1131–1183.

[Adl78] L. Adleman, *Two theorems on random polynomial time*, 19th Annual Symposium on Foundations of Computer Science (sfcs 1978), IEEE, 1978, pp. 75–83.

[AGH90] W. Aiello, S. Goldwasser, and J. Hastad, *On the power of interaction*, Combinatorica **10** (1990), no. 1, 3–25.

[AK07] S. Aaronson and G. Kuperberg, *Quantum versus classical proofs and advice*, Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07), IEEE, 2007, pp. 115–128.

[AKKT19] S. Aaronson, R. Kothari, W. Kretschmer, and J. Thaler, *Quantum lower bounds for approximate counting via laurent polynomials*, arXiv preprint arXiv:1904.08914 (2019).

[AW09] S. Aaronson and A. Wigderson, *Algebrization: A new barrier in complexity theory*, ACM Transactions on Computation Theory (TOCT) **1** (2009), no. 1, 2.

[Bab85] L. Babai, *Trading group theory for randomness*, Proceedings of the seventeenth annual ACM symposium on Theory of computing, ACM, 1985, pp. 421–429.

[BBBV97] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, *Strengths and weaknesses of quantum computing*, SIAM journal on Computing **26** (1997), no. 5, 1510–1523.

[BBF98] R. Beigel, H. Buhrman, and L. Fortnow, *NP might not be as easy as detecting unique solutions*, Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '98, ACM, 1998, pp. 203–208.

[BCH$^+$17] A. Bouland, L. Chen, D. Holden, J. Thaler, and P. N. Vasudevan, *On the power of statistical zero knowledge*, 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2017, pp. 708–719.

[Bei89] R. Beigel, *On the relativized power of additional accepting paths*, [1989] Proceedings. Structure in Complexity Theory Fourth Annual Conference, IEEE, 1989, pp. 216–224.

[Bei94] ______, *Perceptrons, PP, and the polynomial hierarchy*, Computational complexity **4** (1994), no. 4, 339–349.

[BFL91] L. Babai, L. Fortnow, and C. Lund, *Non-deterministic exponential time has two-prover interactive protocols*, Computational complexity **1** (1991), no. 1, 3–40.

[BFT98] H. Buhrman, L. Fortnow, and T. Thierauf, *Nonrelativizing separations*, Proceedings. Thirteenth Annual IEEE Conference on Computational Complexity (Formerly: Structure in Complexity Theory Conference)(Cat. No. 98CB36247), IEEE, 1998, pp. 8–12.

[BG81] C. H. Bennett and J. Gill, *Relative to a random oracle A, $P^A \neq NP^A \neq co-NP^A$ with probability 1*, SIAM Journal on Computing **10** (1981), no. 1, 96–113.

[BGM03] E. Böhler, C. Glaßer, and D. Meister, *Error-bounded probabilistic computations between ma and am*, International Symposium on Mathematical Foundations of Computer Science, Springer, 2003, pp. 249–258.

[BGS75] T. Baker, J. Gill, and R. Solovay, *Relativizations of the $P = ?NP$ question*, SIAM Journal on computing **4** (1975), no. 4, 431–442.

[BM88] L. Babai and S. Moran, *Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes*, Journal of Computer and System Sciences **36** (1988), no. 2, 254–276.

[BT00] H. Buhrman and L. Torenvliet, *Randomness is hard*, SIAM Journal on Computing **30** (2000), no. 5, 1485–1501.

[Cai86] J. Y. Cai, *With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy*, Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '86, ACM, 1986, pp. 21–29.

[CCD$^+$03] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D. A. Spielman, *Exponential algorithmic speedup by a quantum walk*, Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, ACM, 2003, pp. 59–68.

[CCG$^+$94] R. Chang, B. Chor, O. Goldreich, J. Hartmanis, J. Håstad, D. Ranjan, and P. Rohatgi, *The random oracle hypothesis is false*, Journal of Computer and System Sciences **49** (1994), no. 1, 24–39.

[Che16] L. Chen, *A note on oracle separations for bqp*, 2016, 1605.00619.

[CKS81] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer, *Alternation*, J. Assoc. Comput. Mach. **28** (1981), no. 1, 114–133.

[FFK94] S. A. Fenner, L. J. Fortnow, and S. A. Kurtz, *Gap-definable counting classes*, Journal of Computer and System Sciences **48** (1994), no. 1, 116–148.

[FFKL03] S. Fenner, L. Fortnow, S. A. Kurtz, and L. Li, *An oracle builder's toolkit*, Information and Computation **182** (2003), no. 2, 95–136.

[FIKU08] L. Fortnow, R. Impagliazzo, V. Kabanets, and C. Umans, *On the complexity of succinct zero-sum games*, Computational Complexity **17** (2008), no. 3, 353–376.

[FK97] U. Feige and J. Kilian, *Making games short*, STOC, vol. 97, 1997, pp. 506–516.

[For99] L. Fortnow, *Relativized worlds with an infinite hierarchy*, Information Processing Letters **69** (1999), no. 6, 309–313.

[For02] ———, *Complexity class of the week: SPP, part I*, Computational Complexity, `https://blog.computationalcomplexity.org/2002/10/`, Oct 2002.

[For18] ———, *The zero-one law for random oracles*, Computational Complexity, `https://https://blog.computationalcomplexity.org/2018/08/the-zero-one-law-for-random-oracles.html`, Aug 2018.

[FR98] L. Fortnow and J. Rogers, *Complexity limitations on quantum computation*, Computational Complexity, 1998. Proceedings. Thirteenth Annual IEEE Conference on, IEEE, 1998, pp. 202–209.

[GKR$^+$95] F. Green, J. Kobler, K. W. Regan, T. Schwentick, and J. Torán, *The power of the middle bit of a #P function*, Journal of Computer and System Sciences **50** (1995), no. 3, 456–467.

[GW07] G. Gutoski and J. Watrous, *Toward a general theory of quantum games*, Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, ACM, 2007, pp. 565–574.

[Hel84] H. Heller, *Relativized polynomial hierarchies extending two levels*, Mathematical systems theory **17** (1984), no. 1, 71–84.

[HHT97] Y. Han, L. A. Hemaspaandra, and T. Thierauf, *Threshold computation and cryptographic security*, SIAM Journal on Computing **26** (1997), no. 1, 59–78.

[IV12] T. Ito and T. Vidick, *A multi-prover interactive proof for NEXP sound against entangled provers*, 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science, IEEE, 2012, pp. 243–252.

[JJUW10] R. Jain, Z. Ji, S. Upadhyay, and J. Watrous, *$QIP = PSPACE$*, Communications of the ACM **53** (2010), no. 12, 102–109.

[Kit97] A. Y. Kitaev, *Quantum computations: algorithms and error correction*, Uspekhi Matematicheskikh Nauk **52** (1997), no. 6, 53–112.

[Kup09] G. Kuperberg, *How hard is it to approximate the Jones polynomial?*, arXiv preprint arXiv:0908.0512 (2009).

[Lad89] R. E. Ladner, *Polynomial space counting problems*, SIAM Journal on Computing **18** (1989), no. 6, 1087–1097.

[Lau83] C. Lautemann, *BPP and the polynomial hierarchy*, Information Processing Letters **17** (1983), no. 4, 215–217.

[LZ17] S. Lovett and J. Zhang, *On the impossibility of entropy reversal, and its application to zero-knowledge proofs*, Theory of Cryptography Conference, Springer, 2017, pp. 31–55.

[MW05] C. Marriott and J. Watrous, *Quantum arthur-merlin games*, Computational Complexity **14** (2005), no. 2, 122–152.

[Oka00] T. Okamoto, *On relationships between statistical zero-knowledge proofs*, Journal of Computer and System Sciences **60** (2000), no. 1, 47–108.

[Pap85] C. H. Papadimitriou, *Games against nature*, Journal of Computer and System Sciences **31** (1985), no. 2, 288–301.

[Raz87] A. A. Razborov, *Lower bounds on the dimension of schemes of bounded depth in a complete basis containing the logical addition function*, Mat. Zametki **41** (1987), no. 4, 598–607, 623.

[RS98] A. Russell and R. Sundaram, *Symmetric alternation captures BPP*, Computational Complexity **7** (1998), no. 2, 152–162.

[RST15] B. Rossman, R. A. Servedio, and L.-Y. Tan, *An average-case depth hierarchy theorem for boolean circuits*, Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on, IEEE, 2015, pp. 1030–1048.

[RT18] R. Raz and A. Tal, *Oracle separation of BQP and PH*, Electronic Colloquium on Computational Complexity, Report No. 107 (2018).

[Sav70] W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, Journal of computer and system sciences **4** (1970), no. 2, 177–192.

[Sha92] A. Shamir, *IP* $=$ *PSPACE*, Journal of the ACM (JACM) **39** (1992), no. 4, 869–877.

[SV03] A. Sahai and S. Vadhan, *A complete problem for statistical zero knowledge*, Journal of the ACM (JACM) **50** (2003), no. 2, 196–249.

[Tod91] S. Toda, *PP is as hard as the polynomial-time hierarchy*, SIAM Journal on Computing **20** (1991), no. 5, 865–877.

[Tur37] A. M. Turing, *On computable numbers, with an application to the entscheidungsproblem*, Proceedings of the London mathematical society **2** (1937), no. 1, 230–265.

[Ver92] N. Vereschchagin, *On the power of PP*, 1992 Seventh Annual Structure in Complexity Theory Conference, IEEE, 1992, pp. 138–143.

[Vya03] M. Vyalyi, *QMA* $=$ *PP implies that PP contains PH*, ECCCTR: Electronic Colloquium on Computational Complexity, technical reports, Citeseer, 2003.

[Wat00] J. Watrous, *Succinct quantum proofs for properties of finite groups*, Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on, IEEE, 2000, pp. 537–546.

[Wat03] ______, *On the complexity of simulating space-bounded quantum computations*, Computational Complexity **12** (2003), no. 1-2, 48–84.

[Wat09] ______, *Quantum computational complexity*, Encyclopedia of complexity and systems science (2009), 7174–7201.

[ZP03] S. Zachos and A. Pagourtzis, *Combinatory complexity: Operators on complexity classes*, Proc. Panhellenic Logic Symposium, 2003.