Complexity Zoology

Robert Sanders

March 5, 2019

1 Overview and Preliminaries

Complexity Zoology is an expert system for studying inclusions and oracle separations of complexity classes. The system reads a text file describing an initial set of inclusions and separations, then deduces all logical consequences. For a potential inclusion $\mathsf{A} \subseteq \mathsf{B}$, the system is capable of understanding that the inclusion is true, false, or unknown relative to (a) all oracles, (b) the random oracle, (c) the trivial oracle, (d) all affine oracles (see [AB18]), (e) some affine oracle, or (f) an arbitrary oracle.

We use the notation $\Sigma = \{0, 1\}$, and for a set S we use S^* to indicate the set of all finite sequences in S. For definiteness, meanings of standard concepts in complexity theory, such as Turing machines and commonly arising complexity classes are taken from [AB09].

2 Complexity Classes and Oracles

For the purposes of this system, a complexity class is not a set of languages but a family of sets of languages indexed by the set of all oracles, where oracles are considered to be decision functions $f : \Sigma^* \to \Sigma$. For example, we consider P to be the class $\mathsf{P} = \{\mathsf{P}^f : f : \Sigma^* \to \Sigma\}$, where P^f is P relative to the oracle f. This allows for a consistent notion of what it means for an inclusion or separation to hold for all, none, or some oracles, avoiding the need for a universal definition of A^f for an arbitrary complexity class A and an oracle f.

If a complexity class A is defined in terms of Turing machines, then A^f can be defined in the same way, replacing references to Turing machines with references to oracle Turing machines that query the f-oracle. Formally, let TM denote the set of all Turing machines, and let TM_f denote the set of all f-oracle Turing machines. Then if A is defined to be

$$\mathsf{A} = \{L \subseteq \Sigma^* : (\exists M \in TM)\varphi(L, M)\}$$

for a binary predicate $\varphi(x, y)$, we consider A^f to be defined to be

$$\mathsf{A}^f = \{L \subseteq \Sigma^* : (\exists M \in TM_f)\varphi(L, M)\}.$$

3 Operators on Complexity Classes

An *operator* on the set of all complexity classes is an inclusion-preserving automorphism.

We first define a complexity class operator that, while not implemented in Complexity Zoology, is necessary for defining a key property of the classes in the system.

Definition 3.1. For a complexity class A, define

$$\mathsf{P}_{\mathsf{prep}} \cdot \mathsf{A} = \{f(\mathcal{L}) : \mathcal{L} \in \mathsf{A} \,\&\, f : \Sigma^* \to \Sigma^* \text{ is computable in polynomial-time.}\}.$$

For the complexity classes in this project, it is the case that $\mathsf{P}_{\mathsf{prep}} \cdot \mathsf{A} = \mathsf{A}$. The operator inclusions and quadratic relations that the system uses for knowledge propogation can fail if this condition is not satisfied.

Definition 3.2. The operators id, co, and cocap are given by

$$\begin{aligned} \mathsf{id} \cdot \mathsf{A} &= \mathsf{A}, \\ \mathsf{co} \cdot \mathsf{A} &= \{\mathcal{L} \subseteq \Sigma^* : \Sigma^* \setminus \mathcal{L} \in \mathsf{A}\}, \\ \mathsf{cocap} \cdot \mathsf{A} &= \mathsf{A} \cap \mathsf{co} \cdot \mathsf{A}. \end{aligned}$$

It is immediate from these definitions that $\mathsf{co}\cdot\mathsf{co}\cdot\mathsf{A} = \mathsf{A}$, $\mathsf{cocap}\cdot\mathsf{A} \subseteq \mathsf{A}$, and $\mathsf{cocap}\cdot\mathsf{A} \subseteq \mathsf{co}\cdot\mathsf{A}$ for every A. Additionally, the cocap operator absorbs the co operator, because

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

and

$$\begin{aligned} \mathsf{co} \cdot \mathsf{cocap} \cdot \mathsf{A} &= \{\mathcal{L} \subseteq \Sigma^* : \Sigma^* \setminus \mathcal{L} \in \mathsf{cocap} \cdot \mathsf{A}\} \\ &= \{\mathcal{L} \subseteq \Sigma^* : \Sigma^* \setminus \mathcal{L} \in \mathsf{A} \,\&\, \Sigma^* \setminus \mathcal{L} \in \mathsf{co} \cdot \mathsf{A}\} \\ &= \{\mathcal{L} \subseteq \Sigma^* : \mathcal{L} \in \mathsf{co} \cdot \mathsf{A} \,\&\, \mathcal{L} \in \mathsf{A}\} \\ &= \mathsf{cocap} \cdot \mathsf{A}. \end{aligned}$$

It follows that cocap is idempotent, because $\mathsf{cocap} \cdot \mathsf{cocap} \cdot \mathsf{A} = \mathsf{cocap} \cdot \mathsf{A} \cap \mathsf{co} \cdot \mathsf{cocap} \cdot \mathsf{A} = \mathsf{cocap} \cdot \mathsf{A}$.

The next operator defines *polynomial advice.*

Definition 3.3. For a complexity class A, define

$$\mathsf{poly}(\mathsf{A}) = \{\mathcal{L} \subseteq \Sigma^* : (\exists \mathcal{L}' \in \mathsf{A}, f : \mathbb{N} \to \Sigma^*)[|f(n)| = \mathcal{O}(n^k) \,\&\, (x \in \mathcal{L} \Leftrightarrow \langle x, f(|x|)\rangle \in \mathcal{L}')]\}.$$

For the models of computation in this project, it is possible to ignore the advice string $f(n)$, so in practice $\mathsf{A} \subseteq \mathsf{poly} \cdot \mathsf{A}$ for each complexity class A of interest.

Furthermore, observe that $\mathcal{L} \subseteq \mathsf{poly} \cdot \mathsf{poly} \cdot \mathsf{A}$ if and only if there exist $\mathcal{L}' \in \mathsf{A}$ and $f, g : \mathbb{N} \to \Sigma^*$ with $|f(n)|, |g(n)| = \mathcal{O}(n^k)$ such that

$$x \in \mathcal{L} \Longleftrightarrow \langle \langle x, f(|x|)\rangle, g(|\langle x, f(|x|)\rangle|)\rangle \in \mathcal{L}'.$$

The models of computation we consider are unaffected by slight changes to the encoding of tuples, so for the complexity classes studied here, the above is equivalent to the existence of $\mathcal{L}', f, g$ such that

$$x \in \mathcal{L} \iff \langle x, \underbrace{\langle f(|x|), g(|\langle x, f(|x|)\rangle|)\rangle}_{h(|x|)}\rangle \in \mathcal{L}'.$$

Then $h : \mathbb{N} \to \Sigma^*$ satisfies $|h(n)| = \mathcal{O}(n^k)$, and hence $\mathcal{L} \in \mathsf{poly} \cdot \mathsf{poly} \cdot \mathsf{A} \Rightarrow \mathcal{L} \in \mathsf{poly} \cdot \mathsf{A}$. Thus, we also regard poly as an idempotent operator.

The remaining operators are well defined for all complexity classes but are intended for those involving polynomial-time computations.

Definition 3.4. For a complexity class A, say that $\mathcal{L} \in \mathsf{N} \cdot \mathsf{A}$ if there is exists a polynomial $p : \mathbb{N} \to (0, \infty)$ and a language $\mathcal{L}' \in \mathsf{A}$ such that

$$x \in \mathcal{L} \iff \langle x, y\rangle \in \mathcal{L}' \text{ for some } y \in \Sigma^{p(|x)}.$$

Definition 3.5. For a complexity class A, a language $\mathcal{L}$ lies in $\oplus \cdot \mathsf{A}$ if there exists a language $\mathcal{L}' \in \mathsf{A}$ and a function $p : \mathbb{N} \to (0, \infty)$ satisfying $p(n) = \mathcal{O}(n^k)$ for some positive integer k such that for every $x \in \Sigma^*$,

$$x \in \mathcal{L} \iff |\{y \in \Sigma^{p(|x|)} : \langle x, y\rangle \in \mathcal{L}'\}| \equiv 1 \pmod 2.$$

BP and P are the probabilistic operators:

Definition 3.6. For a complexity class A, say that $\mathcal{L} \in \mathsf{BP} \cdot \mathsf{A}$ if there is exists a polynomial $p : \mathbb{N} \to (0, \infty)$ and a language $\mathcal{L}' \in \mathsf{A}$ such that

$$x \in \mathcal{L} \Longrightarrow \langle x, y\rangle \in \mathcal{L}' \text{ for } > 2/3 \text{ of all } y \in \Sigma^{p(|x)};$$
$$x \notin \mathcal{L} \Longrightarrow \langle x, y\rangle \notin \mathcal{L}' \text{ for } > 2/3 \text{ of all } y \in \Sigma^{p(|x)}.$$

The P is defined similarly, except that 2/3 is replaced with 1/2.

4 Annotations

This list consists of annotations used in the input files of Complexity Zoology. These annotations denote recurring footnotes too small for their own section in this document.

- [**count**]: The complexity classes can be separated because they have different cardinalities.
- [**def**]: This statement is true by definition.
- [**immediate**]: This result is immediate from the definitions of the respective complexity classes.
- [**probably**]: This result is believed to be true, but it needs to be checked.
- [**rel?**]: It should be double-checked that this statement relativizes.

5 Diagonalization (diag)

The method of diagonalization can be used to unconditionally separate several complexity classes. In its standard form, it can be used to prove the *time hierarchy theorem.*

Theorem 5.1. *If $f, g : \mathbb{N} \to \mathbb{N}$ are time-constructible functions satisfying $f(n) \log f(n) = o(g(n))$, then*

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

To prove this theorem, construct a TM D that carries 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. Next, assume the language $\mathcal{L}$ that D determines lies in $\mathsf{DTIME}(f(n))$. Then, there exists a TM M that decides $\mathcal{L}$ in $\mathcal{O}(f(n))$-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 $D(y) = 1 - M_y(y) = 1 - M(y)$. But since D and M both decide $\mathcal{L}$, we should have $D(y) = M(y)$, so this is a contradiction.

6 $\mathsf{AWPP} \subseteq \mathsf{PP}$(AWPPvPP)

AWPP can be defined in terms of functions in GapP; i.e., functions $f : \Sigma^* \to \mathbb{Z}$ such that for a non-deterministic TM M, $f(x)$ is the difference between the number of accepting paths and the number of rejecting paths for M with input x [FFK94]. Specifically, a language $\mathcal{L}$ lies in AWPP if and only if for every polynomial p, there exists a function $f \in \mathsf{GapP}$ and an everywhere nonzero function $g : \Sigma^* \to \mathbb{N}$ in FP such that

$$x \in \mathcal{L} \implies 1 - 2^{-p(|x|)} \leq f(x)/g(x) \leq 1,$$
$$x \notin \mathcal{L} \implies 0 \leq f(x)/g(x) \leq 2^{-p(|x|)}.$$

Also, note that $\mathcal{L} \in \mathsf{PP}$ if and only if there is a function $f \in \mathsf{GapP}$ such that

$$x \in \mathcal{L} \implies f(x) \geq 0,$$
$$x \notin \mathcal{L} \implies f(x) < 0.$$

Suppose that $\mathcal{L} \in \mathsf{GapP}$. Then fix $p = 2$ in the definition of AWPP; there must exist functions $f \in \mathsf{GapP}$ and $g : \Sigma^* \to \mathbb{N}$ in FP such that

$$x \in \mathcal{L} \implies 3/4 \leq f(x)/g(x) \leq 1,$$
$$x \notin \mathcal{L} \implies 0 \leq f(x)/g(x) \leq 1/4.$$

Since $\mathsf{FP} \subseteq \mathsf{GapP}$ and GapP is closed under multiplication and subtraction, $h(x) = 4f(x) - 2g(x)$ is in GapP, and hence

$$x \in \mathcal{L} \implies g(x) \leq 4f(x) - 2g(x) \leq 2g(x) \implies h(x) \geq 0,$$
$$x \notin \mathcal{L} \implies -2g(x) \leq 4f(x) - 2g(x) \leq -g(x) \implies h(x) < 0.$$

So $\mathcal{L} \in \mathsf{PP}$, and we have $\mathsf{AWPP} \subseteq \mathsf{PP}$.

7 Oracle Access (oap)

For a pair of classes A and B, it may be the case that $\mathsf{A}^O \not\subseteq \mathsf{B}^O$ because the computational model of A^O is able to make longer oracle calls than that of B^O. For example, B^O could be polynomially limited in space or time and therefore be unable to write long questions for the oracle, while A^O has no such limitations.

8 Password Oracles (pass)

A *password oracle* is a type of oracle f constructed so that $\mathsf{A}^f \not\subseteq \mathsf{B}^f$. Typically, f is a function $\Sigma^* \times \Sigma^* \to \Sigma^*$ chosen so that $PW_f = \{x \in \Sigma : P\}$ lies in A^f but not in B^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 8.1. *Let $f : \Sigma^{2*} \to \Sigma \cup \{\square\}$ 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 $f(x, y) \neq \square$. 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 $f(x, y) \neq \square$, then* $\Pr[f(x, y) = 1] = \Pr[f(x, y) = 0] = 1/2$.

Then $(\mathsf{cocap} \cdot \mathsf{UP})^f \not\subseteq (\mathsf{P/poly})^f$ 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 $f(x, y) \neq \square$. Then $PW_f \in (\mathsf{cocap} \cdot \mathsf{UP})^f$, because for a given $x \in \Sigma^*$ the unique y can be used as a 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

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

□

9 $\mathsf{ZPP} = \mathsf{cocap} \cdot \mathsf{RP}$ (ZRP)

The equality $\mathsf{ZPP} = \mathsf{cocap} \cdot \mathsf{RP}$ is sometimes taken to be the definition of ZPP. Otherwise, ZPP is defined as the class of decision problems computable by a probabilistic Turing machine that always computes the correct solution and is expected to run in polynomial-time. With this definition, one can show $\mathsf{ZPP} \subseteq \mathsf{RP}$ by running a ZPP-machine for twice its expected running time, then returning the output 0 if an answer has not yet been determined. Since ZPP is symmetric, it follows that $\mathsf{ZPP} \subseteq \mathsf{cocap} \cdot \mathsf{RP}$. Conversely, to show that $\mathsf{cocap} \cdot \mathsf{RP} \subseteq \mathsf{ZPP}$, run an RP- and $\mathsf{co} \cdot \mathsf{RP}$-machine in parallel, and repeat as necessary until either the RP-machine returns 1 or the $\mathsf{co} \cdot \mathsf{RP}$-machine returns 0.

10 Bibliography

References

[AA09] Scott Aaronson and Andris Ambainis. The need for structure in quantum speedups. *arXiv preprint arXiv:0911.0996*, 2009.

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

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

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

[Aar10] Scott Aaronson. Bqp and the polynomial hierarchy. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 141–150. ACM, 2010.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[AB18] Bariş Aydinlioğlu and Eric Bach. Affine relativization: Unifying the algebrization and relativization barriers. *ACM Trans. Comput. Theory*, 10(1):1:1–1:67, January 2018.

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

[AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory (TOCT)*, 1(1):2, 2009.

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

[BBBV97] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[BBF98] Richard Beigel, Harry Buhrman, and Lance Fortnow. NP might not be as easy as detecting unique solutions. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 203–208, New York, NY, USA, 1998. ACM.

[BCH$^+$17] Adam Bouland, Lijie Chen, Dhiraj Holden, Justin Thaler, and Prashant Nalini Vasudevan. On the power of statistical zero knowledge. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 708–719. IEEE, 2017.

[Bei94] Richard Beigel. Perceptrons, PP, and the polynomial hierarchy. *Computational complexity*, 4(4):339–349, 1994.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity*, 1(1):3–40, 1991.

[BFT98] Harry Buhrman, Lance Fortnow, and Thomas Thierauf. Nonrelativizing separations. In *Proceedings. Thirteenth Annual IEEE Conference on Computational Complexity (Formerly: Structure in Complexity Theory Conference)(Cat. No. 98CB36247)*, pages 8–12. IEEE, 1998.

[BG81] Charles H Bennett and John Gill. Relative to a random oracle a, $p^a \neq np^a \neq \text{co} - np^a$ with probability 1. *SIAM Journal on Computing*, 10(1):96–113, 1981.

[BM88] László Babai and Shlomo Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

[BT00] Harry Buhrman and Leen Torenvliet. Randomness is hard. *SIAM Journal on Computing*, 30(5):1485–1501, 2000.

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

[CCD$^+$03] Andrew M Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 59–68. ACM, 2003.

[FFK94] Stephen A Fenner, Lance J Fortnow, and Stuart A Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

[FFKL03] Stephen Fenner, Lance Fortnow, Stuart A Kurtz, and Lide Li. An oracle builder's toolkit. *Information and Computation*, 182(2):95–136, 2003.

[For99] Lance Fortnow. Relativized worlds with an infinite hierarchy. *Information Processing Letters*, 69(6):309–313, 1999.

[FR98] Lance Fortnow and John Rogers. Complexity limitations on quantum computation. In *Computational Complexity, 1998. Proceedings. Thirteenth Annual IEEE Conference on*, pages 202–209. IEEE, 1998.

[GKR$^+$95] Frederic Green, Johannes Kobler, Kenneth W Regan, Thomas Schwentick, and Jacobo Torán. The power of the middle bit of a $\#P$ function. *Journal of Computer and System Sciences*, 50(3):456–467, 1995.

[Hel84] Hans Heller. Relativized polynomial hierarchies extending two levels. *Mathematical systems theory*, 17(1):71–84, Dec 1984.

[Lau83] Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1983.

[RS98] Alexander Russell and Ravi Sundaram. Symmetric alternation captures BPP. *Computational Complexity*, 7(2):152–162, 1998.

[RST15] Benjamin Rossman, Rocco A Servedio, and Li-Yang Tan. An average-case depth hierarchy theorem for boolean circuits. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 1030–1048. IEEE, 2015.

[RT18] Ran Raz and Avishay Tal. Oracle separation of BQP and PH. *Electronic Colloquium on Computational Complexity, Report No. 107*, 2018.

[Sha92] Adi Shamir. $IP = PSPACE$. *Journal of the ACM (JACM)*, 39(4):869–877, 1992.

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

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

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