Research Results Rough

Alex Creiner

July 24, 2021

In the following we fix $\{0, 1\}$ as the input alphabet for all Turing machines, and also fix an enumeration of all finite binary strings. Under this fixed enumeration, we will often refer to Turing machine inputs as integers. When we write something like $M(n)$, where M is a Turing machine, we really mean that the input is the n^{th} string under the fixed enumeration.

Let $\mathbf{R}$ denote the collection of all recursive relations on ω. These sets can be regarded as **decision problems**, or **languages**, in the sense that $n \in R$ means that the answer to the 'n' instance of the problem is yes, and $n \notin R$ means the answer is no. Of course, we can also regard decision problems as sets of binary strings, based on our fixed enumeration, as well as infinite binary sequences, with the interpretation that the n^{th} digit of the sequence is a 1 if $n \in R$, and 0 otherwise. Thus decision problems can and will be interchangeably viewed as subsets of ω, subsets of $\{0, 1\}^*$, and elements of Cantor space $\mathcal{C}$.

1 Preliminary Stuff from Complexity Theory

The following very simple lemma is essential to our results working:

Lemma 1.1. *Let $\boldsymbol{C}$ be a nontrivial complexity class. If E is $\boldsymbol{C}$ complete, then both $E - F$ and $E \cup F$ are $\boldsymbol{C}$ complete for any finite set F.*

This is obviously true for classes like $\mathbf{P}$ and $\mathbf{NP}$, but I wanted to know what the minimal assumptions about a complexity class would have to be in order to get 'nice facts' like this one. Toward this, I tried formally defining what a 'nontrivial complexity class' was based on the descriptive set theory definition of a pointclass, and it led to a lot of nice results. This was a lot of fun to do, but you might want to skip past this section at first and come back to it later.

We define a **complexity class** to be a nonempty subset $\mathbf{C} \subseteq \mathbf{R}$ of decision problems which is closed under log-space reductions. That is, if $f : \{0, 1\}^* \to \{0, 1\}^*$ is log space computable, and L_1, L_2 are language such that $L_2 \in \mathbf{C}$ and $n \in L_1 \iff f(n) \in L_2$, then $L_1 \subseteq \mathbf{C}$. Since not every computable function is log space computable, complexity classes are not necessarily pointclasses in the lightface sense. (I would like to show this and maybe explore it a little.) $\mathbf{P}$, $\mathbf{NP}$. $\mathbf{BPP}$, $\mathbf{PSPACE}$, $\mathbf{NL}$, and so forth are all complexity classes via this definition.

I've never seen a textbook formally define a complexity class this way, and the reason is clear. I chose log-space reductions in the definition because it would serve well for a larger set of classes than choosing poly-time reductions. For instance, soon I'll show that no nontrivial proper subset of $\mathbf{P}$ can possibly be a complexity class under a poly-time definition. Furthermore, sticking exclusively with log-space reductions isn't perfect either. If a set is closed under polynomial time reductions, then it is closed under log-space reductions, but the converse is false unless $\mathbf{L} = \mathbf{P}$, and so there are likely complexity classes under the log space definition which would fail to be remain classes under the poly-time definition.

There is a special very small class, $\mathbf{FO}$, the class of boolean first order queries, which is known to be properly contained in $\mathbf{L}$. This class is also known to be equal to $\mathbf{AC}^0$, the class of problems which are decidable by polynomial sized families of boolean circuits (one for each input length) of constant depth, but unbounded $FAN - IN$, and is pretty much the smallest class which anyone cares about. To define complexity classes to be closed under $\mathbf{AC}^0$ reductions would be an even finer definition, and the sensibly 'smallest' nontrivial complexity class would be a complexity class under this definition. However, since $\mathbf{AC}^0$ is proven to be proper in $\mathbf{L}$, most if not all classes containing $\mathbf{L}$ would fail to be complexity classes.

The best course of action would probably to define complexity classes conditionally, in terms of their size. For sets containing **P**, define complexity classes to be sets closed under poly-time reductions. For sets containing **L** but not contained in **P**, define complexity classes to be sets closed under log-space reductions, and for sets containing $\mathbf{AC}^0$ but not contained in **L**, define complexity classes to be sets which are closed under $\mathbf{AC}^0$ reductions. Here I take the log-space reductions definition as the exclusive one, since that's all we need, and point out what would change under the poly-time definition. I'd be curious what sticks in the $\mathbf{AC}^0$ definition.

Before looking at these properties, an important fact that is trivial for poly-time reductions is not so trivial for log-space reductions. I'll outline a proof but won't go into detail.

Lemma 1.2. *The composition of two log-space computable functions is itself a log-space computable function.*

Proof. Let M_1 and M_2 be machines which each compute a function in logarithmic space. The reason it isn't obvious that the composition isn't log space computable is that if we simply do the naive thing: run first machine and then feed the output into the second machine, then the first intermediate output is only guaranteed to be polynomial in the length of the input. We don't necessarily have enough space to write this middle output down on the tape! The issue is resolved by taking extra time - whenever the second Turing machine needs to consult a symbol of the first machine's output, we simply repeat the entire computation of the first machine up to the point when we obtain the symbol that is needed. This way, we never have to store the first machine's output in it's entirety. □

Lemma 1.3. *Every complexity class except for* $\{\varnothing\}$ *contains an infinite set, namely* $\mathbb{N}$ *(or alternatively* $\{0,1\}^*$ *or even more alternatively the sequence* $111\ldots$*). Furthermore, every complexity class except for the one above as well as* $\{\varnothing, \mathbb{N}\}$ *and* $\{\mathbb{N}\}$ *contains every finite set, and is thus countably infinite. We will refer to any complexity class which isn't one of these three as* ***nontrivial****. We will also refer to sets which aren't* $\varnothing$ *or* $\mathbb{N}$ *as nontrivial.*

Proof. Suppose $\mathbf{C} \neq \{\varnothing\}$ a complexity class. Let $L \in \mathbf{C}$, and $x \in L$. Consider the 'always yes' relation $\mathbb{N}$, and define the function f to simply be the constant function $f(n) = x$. This is clearly computable in log-space, and clearly $n \in \mathbb{N} \iff f(n) \in L$. Thus by closure under reductions it must be that $\mathbb{N} \in \mathbf{C}$.

Now let L be a nontrivial complexity class, let $L \in \mathbf{C}$ be a nontrivial language with $y_1 \in L, y_0 \notin L$, and let $F = \{x_1, x_2, ..., x_n\}$ be an arbitrary finite language. Define the function f by $f(x_i) = y_1$ for $i = 1, ..., n$, and $f(x) = y_0$ otherwise. This function is computable in log-space, because the finite nature of the mapping means all of the 'computation' of the function can be hardcoded into a Turing machine via a finite set of states - we only need space to write the final answer, which doesn't count as space use, and time to read the input. Furthermore it is clear that $x \in F \iff f(x) \in L$, so by closure under reductions it follows that $F \in \mathbf{C}$. Thus $\mathbf{C}$ is infinite, and that it is countable is inherited from $\mathbf{R}$ being countable. □

Lemma 1.4. *If* $\boldsymbol{C}$ *is a nontrivial complexity class with a complete set, then that complete set cannot be trivial. Furthermore, there exists a complexity class with no complete sets.*

Proof. Intuitively, this is because $\varnothing$ and $\mathbb{N}$ represent the 'always no' and 'always yes' relations, respectively. To have a reduction from a language L to say, $\mathbb{N}$ would be to say that L itself is $\mathbb{N}$, since $f(x) \in \mathbb{N} \Rightarrow x \in L$, but $f(x) \in \mathbb{N}$ is true all of the time. So the only language which can reduce to $\mathbb{N}$ is $\mathbb{N}$ itself, and the same is true of $\varnothing$ by an identical argument. It follows that the only complexity class for which $\mathbb{N}$ is complete is $\{\mathbb{N}\}$, and the only complexity class for which $\varnothing$ is complete is $\{\varnothing\}$.

The easy example of a complexity class with no complete set is the third trivial class, $\{\varnothing, \mathbb{N}\}$. This follows from the previous paragraph - there is no way to reduce $\varnothing$ to $\mathbb{N}$, and vice versa, so neither can be complete. □

Lemma 1.5. *All nontrivial sets in* $\boldsymbol{L}$ *are* $\boldsymbol{L}$*-complete. (If complexity classes were defined in terms of polynomial time reductions, then* $\boldsymbol{L}$ *would be replaced with* $\boldsymbol{P}$*.)*

Proof. Let E be a nontrivial set in **L**, and let $L \in \mathbf{L}$ be arbitrary. Let M_L be the log-space Turing machines deciding L and E. Let $x_1 \in E$ and $x_0 \notin E$. Define the mapping f by

$$f(x) = \begin{cases} x_1 & \text{if } M_L(x) \text{ halts in acceptance} \\ x_0 & \text{otherwise} \end{cases}$$

Clearly $x \in L \iff f(x) \in E$, and clearly f is log-space computable by virtue of L iteself being log-space computable. Thus this is a very stupid reduction from L to E. ☐

Conjecture 1.1. *$\boldsymbol{L}$ is the smallest nontrivial complexity class under our definition. (Again, under a polynomial time reduction definition of complexity class, $\boldsymbol{P}$ would replace $\boldsymbol{L}$.)*

Proof. Suppose $\mathbf{C} \subseteq \mathbf{L}$ be nontrivial. Let $L \in \mathbf{C}$ be nontrivial. Then $L \in \mathbf{L}$, so L is $\mathbf{L}$ complete. Thus any language $L' \in \mathbf{L}$ must reduce to L, and so by closure under reductions it must follow that $L' \in \mathbf{C}$. Thus $\mathbf{L} \subseteq \mathbf{C}$, i.e. $\mathbf{L} = \mathbf{C}$. ☐

Lemma 1.6. *If $\boldsymbol{C}$ is a nontrivial complexity class with a finite or a cofinite complete problem, then $\boldsymbol{C} = \boldsymbol{L}$.*

Proof. Let M be a finite complete problem for a nontrivial complexity class $\mathbf{C}$. Note that any finite problem can be solved in log-space (and linear time), via just 'hardcoding' computations into the states of a Turing machine. The only computational resource used is the time required to scan the input. Thus, any problem in the class $\mathbf{C}$ can be simply reduced to M in log-space, and then solved in log-space. Thus, any problem in $\mathbf{C}$ can be solved via two sequential log-space reductions, and this can be remade into one overall log-space computation by our results, and so $\mathbf{C} \subseteq \mathbf{L}$, but since $\mathbf{L}$ is the smallest nontrivial complexity class, we must have that $\mathbf{C} = \mathbf{L}$.

Next suppose that $\mathbf{C}$ is a nontrivial complexity class with a cofinite complete problem E. Then E^c would be finite and complete for $\mathbf{coC}$, meaning that $\mathbf{coC} = \mathbf{L}$, and since $\mathbf{L}$ is self dual, this would imply that $\mathbf{C} = \mathbf{L}$. ☐

The reason for all this was to derive the minimal requirements for a class which are needed to assume that complete problems must be infinite and coinfinite, and we now have that:

Corollary 1.1. *Let $\boldsymbol{C}$ be a complexity class properly containing $\boldsymbol{L}$. Then any complete problem for $\boldsymbol{C}$ must be both infinite and coinfinite. Furthermore, every nontrivial complexity class has an infinite complete problem which isn't $\mathbb{N}$.*

Proof. If E were a complete problem for such a class, then assuming it were finite would require that it be equal to $\mathbf{L}$ by the above results, a contradiction. Furthermore, it goes without saying that $\mathbf{L}$ has nontrivial infinite problems. ☐

It is sort of interesting to know that $\mathbf{L}$ is the only complexity class which can have finite complete problems. To prove $\mathbf{P} \neq \mathbf{L}$, it suffices to show that no finite problem in $\mathbf{P}$ can be $\mathbf{P}$ complete.

Lemma 1.7. *Let $\boldsymbol{C}$ be a nontrivial complexity class, and let E be an infinite $\boldsymbol{C}$ complete problem. Then $E - F$ is still complete for any finite $F \subseteq E$, and with the additional assumption that E is not cofinite, we also have that $E \cup G$ is still complete for any finite set G*

Proof. Note that in the statement I get to assume that my arbitrary class has an infinite complete problem, but I can't hold E completely arbitrary because in the case that $\mathbf{C} = \mathbf{L}$ it may be the case that E is finite. So I got some but not all of what I wanted.

Let L be an arbitrary language in $\mathbf{C}$, and f be the reduction from L to E. For the case of $E - F$, since E is infinite $E - F$ is still itself infinite. Fix $x_1 \in E - F$. Note that since F is finite, we can compute easily in log space whether or not $f(x) \in F$. Define

$$m(x) = \begin{cases} x_1 & \text{if } f(x) \in F \\ f(x) & \text{otherwise} \end{cases}$$

Since f is log space computable, we can compute m by computing f, and then compute m conditionally on the subroutine deciding if $m \in F$. (This requires that lemma about composition of log space functions, but it works.) So m is log space computable. If $x \in L$ and $f(x) \in F$, then $m(x) = x_1 \in E - F$, so $m(x) \in E - F$. If $x \in L$ and $f(x) \notin F$, then $x \in L \implies m(x) = f(x) \in E - F$, so either way we have $x \in L \implies m(x) \in E - F$. Conversely, if $x \notin L$, then $f(x) \notin E$, and so $f(x) \notin F$, i.e. $m(x) = f(x) \notin E$. So m is a log space reduction from L to $E - F$, and thus $E - F$ is complete.

Next, let G be finite, and without loss of generality, assume it is disjoint from E. (If it isn't, then we just replace G with $G - E$ and repeat the following argument.) Consider $E \cup G$. Let L and f be as they were before. Also similar to before, we note that in log space it can easily be computed if a string belongs to G. Since E is not cofinite, we can choose an $x_0 \notin E \cup G$. Define

$$p(x) = \begin{cases} x_0 & \text{if } f(x) \in G \\ f(x) & \text{otherwise} \end{cases}$$

By the same argument as above, p is easily seen to be log-space computable. If $x \in L$ then $f(x) \in E$, so it cannot be the case that $f(x) \in G$ since $E \cap G = \varnothing$. Thus $m(x) = f(x) \in E \cup G$. If $x \notin L$, then we have two cases. If $f(x) \in G$, then $m(x) = x_0 \notin E \cup G$, and if $f(x) \notin G$, then since $f(x) \notin E$, we have that $m(x) = f(x) \notin E \cup G$. The proof is complete. □

The complexity class for which we wish to exploit a complete problem is **EXP**, which contains **PSPACE**, which by the space hierarchy theorem properly contains **L**. Thus any complete problem for **EXP** is closed under unions or differences with finite sets. The last remedial fact from complexity theory that we need is this.

Lemma 1.8. *If E is $\boldsymbol{EXP}$ complete, then $\boldsymbol{P}^E = \boldsymbol{NP}^E$.*

Proof. Suppose $L \in \mathbf{P}^E$. Then L is decidable in deterministic polynomial time by a Turing machine M^E equipped an oracle for E, that is, it halts in time $O(n^{k_1})$ for some positive integer k_1. But then the machine M' which simulates M^E on steps which aren't queries to E, and then on query steps simply simulates an exponential time machine which decides E on the query string's input (say time $O(2^{n^{k_2}})$. This machine clearly decides L, and query operates in time $O(n^{k_1}2^{n^{k_2}})$, clearly still exponential time, i.e. $L \in \mathbf{EXP}$. Thus, $\mathbf{P}^E \subseteq \mathbf{EXP}$. Obviously $\mathbf{EXP} \subseteq \mathbf{P}^E$, since we can simply reduce any problem to E in polynomial time and consult the oracle. Thus $P^E = \mathbf{EXP}$.

Now, let $L \in \mathbf{NP}^E$. Let N^E be the relativized nondeterministic machine which decides L, in time $O(n^{k_1})$. Let M^E be the relativized deterministic Turing machine which simulates N^E in order to decide L in time $O(2^{n^{k_1}})$. Repeat the same words in the above paragraph to obtain a nonrelativized deterministic machine M' which decides L in time $O(2^{n^{k_1}}2^{n^{k_2}})$, clearly still exponential time. So $\mathbf{NP}^E \subseteq \mathbf{EXP}$. Finally we have

$$\mathbf{EXP} \subseteq \mathbf{P}^E \subseteq \mathbf{NP}^E \subseteq \mathbf{EXP}$$

I.e. $\mathbf{P}^E = \mathbf{NP}^E$. I want to rewrite this proof and make it more direct. That is, I want to start with a nondeterministic machine N^E running in time n^{k_1}, and show that there is a deterministic machine M^E running in time n^{k_2} which decides the same language as N^E. First, let M_i^E be the relativized deterministic Turing machine which simulates N^E in time $2^{n^{k_1}}$. Then we can eliminate the relativization by simply, rather than querying the oracle, solving the problem in exponential time, say time $2^{n^{k_3}}$. This yields a machine M_i which decides $L(N^E)$ and operates in time $O(2^{n^{k_1}}2^{n^{k_3}})$, still exponential time. Finally, since M_i operates in exponential time, it is immediately in $\boldsymbol{P}^E$, since we can decide any exponential time language in a single step. □

Lemma 1.9. *Consider the following languages:*

$H_u = \{m; x; k$: *The machine coded by m halts in acceptance on in the input x within k steps, with k in unary*$\}$

$H_b = \{m; x; k$: *The machine coded by m halts in acceptance on the input x within k steps, with k in binary*$\}$

Then H_u is $\boldsymbol{P}$-complete, and H_b is $\boldsymbol{EXP}$-complete.

Proof. Suppose that $L \in \boldsymbol{EXP}$, decided by the machine M in time $c2^{n^k}$ for some integers c, k. Let m be the string coding M. Then $x \in L$ iff $M(x)$ halts in acceptance in time $c2^{n^k}$. Note that the number of symbols required to store the integer $c2^{|x|^k}$ is polynomial in $|x|$ - it's just $O(|x|^k)$. Let $l(x)$ be the binary representation of this number, as a function of x. Then consider the mapping $x \mapsto m; x; l(x)$. Clearly $x \in L$ iff $m; x; l(x) \in H_b$, and is computable in constant space, so this is a reduction from L to H_b, proving that H_b is $\boldsymbol{EXP}$-hard. To see that $H_b \in \boldsymbol{EXP}$, note that we can decide H_b by simply simulating 2^k steps of the machine m on the input x, and seeing if it halts. Clearly this can be done with negligible time overhead relative to the number of steps being simulated, so $H_b \in \boldsymbol{EXP}$. □

Some more facts about complexity classes and complete languages which I have thought of to maybe be useful later:

Definition 1.1. Let C be a language. Define the **span** of C, denoted $\boldsymbol{s}(C)$, to be the set of all languages L which are reducible to C.

Lemma 1.10. *$\boldsymbol{s}(C)$ is a complexity class, with C being $\boldsymbol{s}(C)$-complete.*

Proof. This follows from the transitivity of the 'is reducible to' relation. If $L_1 \in \boldsymbol{s}(C)$ and $L_2 \leq L_1$, then since $L_1 \leq C$, it follows that $L_2 \leq C$, so $L_2 \in \boldsymbol{s}(C)$. Thus $\boldsymbol{s}(C)$ is closed under reductions. The second part of the claim is trivial. $\square$

Note that for any complexity class $\mathbf{C}$ with a complete problem L, $\mathbf{C} = \boldsymbol{s}(L)$. We next denote the disjoint union of two languages A and B as $A \sqcup B$, and use the convention that, rigorously,

$$A \sqcup B = \{0;x : x \in A\} \cup \{1;y : y \in B\}$$

where ; denotes string concatenation.

Lemma 1.11. *Let A,B be languages. Then $s(A) \cup s(B) \subseteq s(A \sqcup B)$*

Proof. If $L \in A$, i.e. there is a reduction f from L to A, then we can immediately define a reduction f' from L to $A \sqcup B$ by simply letting $f'(x) = 0;f(x)$. Clearly $x \in L \iff f(x) \in A \iff 0;f(x) \in A \sqcup B$, so $L \in \boldsymbol{s}(A \sqcup B)$. $\square$

This is a nice result, but it would be better if we knew that the union of two complexity classes was a complexity class! Let us confirm these kinds of basic properties.

Lemma 1.12. *Complexity classes are closed under arbitrary unions, intersections, and complements.*

Proof. Let $\mathbf{C}, \mathbf{D}$ be complexity classes. Suppose that $L_1 \in \mathbf{C} \cup \mathbf{D}$, and L_2 has a reduction f to L_1. Then since $\mathbf{C}$ is closed under reductions, $L_2 \in \mathbf{C} \subseteq \mathbf{C} \cup \mathbf{D}$. Clearly this argument works for arbitrary unions as well. If $L_1 \in \mathbf{C} \cap \mathbf{D}$, and L_2 is as above, then by closure under reductions L_2 is in both $\mathbf{C}$ and $\mathbf{D}$, so $L_2 \in \mathbf{C} \cap \mathbf{D}$. Again, this argument seems to extend naturally to arbitrary intersections. Finally, if $L_1 \in \boldsymbol{coC}$, i.e. $L_1^c \in \boldsymbol{C}$, and L_2 is as it was, then $x \in L_2 \iff f(x) \in L_1$ iff $x \notin L_2 \iff f(x) \notin L_1$ iff $x \in L_2^c \iff f(x) \in L_1^c$. Thus $L_2^c \in \boldsymbol{C}$ by closure under reductions, and so $L_2 \in \boldsymbol{coC}$. $\square$

2 Main Results

We wish to characterize the set

$$\mathcal{O} := \{\epsilon \in \mathcal{C} : \mathbf{P}^\epsilon \neq \mathbf{NP}^\epsilon\}$$

Solovay proved the existence of such languages. It has further been proven, and is in fact implicit to Solovay's own constructive proof, that the number of languages with this property is quite plentiful.

Lemma 2.1. *The set $H = \{\epsilon \in \mathcal{C} : \epsilon$ is infinitely often 1$\}$ is $\mathbf{\Pi}_2^0$ complete*

Lemma 2.2. $\mathcal{O} \in \mathbf{\Pi}_2^0$

Proof. Let U be a nondeterministic Turing machine which, given an input $x;m$, simulates $|x|$ many steps of the m^{th} nondeterministic machine. More clearly, U is a machine such that

$$L(U) = \{x;m : \text{The configuration graph of the } m^{th} \text{ nondeterministic Turing machine on input } x \text{ contains an accepting computational path of length less than or equal to } |x|\}$$

This isn't exactly completely truthful. U has the additional property that there is a special symbol in it's alphabet, call it p, not contained in the alphabet of Turing machines which U is universal for, which the machine ignores, but which still count as input length. I.e. appending an input x with some number of p's acts as 'padding' which instructs the machine to 'compute for longer' in spite of the length of x. Thus, if a machine N_m, on an input x, has an accepting path of length, say $2^{|x|}$, then $U(x; p^{2^{|x|}-|x|}; m)$ will accept as well, since it simulates in linear time relative to it's first argument.

It is well known that there is a universal nondeterministic Turing machine which operates with linear simulation overhead. To account for the extra sophistication of our padding symbol, we can have the machine first deterministically scan through the input, counting the symbols in it's first argument (something it would likely have to do anyway in order to cut off nonlinear computational paths), and replace all of the p's with actual blanks. Thus such a machine U surely exists. For simplicity, we will say that U is exactly linear, i.e. it is a precise machine with runtime equal to input length. Again, this is only for notational simplicity. Clearly the machine we just designed to prove the existence of U doesn't actually have this property). Suppose that ϵ is an oracle such that there exists a polynomial time (say time n^{k_1}) deterministic oracle machine M^ϵ which decides U^ϵ, i.e. $L(U) \in \boldsymbol{P}^\epsilon$. Let N^ϵ be an arbitrary nondeterministic polynomial time machine, operating in time n^{k_2}. Let m be the code of this machine. Consider the machine M_N^ϵ which has the code m prewritten on an ancillary tape string. On an input x, M_N^ϵ first 'pads' the input string x with $|x|^{k_2} - |x|$ many p's, and then simulates the deterministic computation of $M^\epsilon(x; p^{|x|^{k_2}-|x|}; m)$. If $N^\epsilon(x)$ has an accepting path, then this path must be of length less than or equal to n^{k_2}. Now to summarize, M^ϵ simulates U^ϵ which simulates $N_m^\epsilon(x)$ for $|x|^{k_2}$ many steps. The deterministic overhead of simulating U is polynomial as well, i.e. this entire process happens in time $(|x|^{k_2})^{k_1} = |x|^k$, where $k = k_1 k_2$, i.e. M_N^ϵ operates in polynomial time. We have M_N^ϵ halt on x iff $N^\epsilon(x)$ has an accepting path, i.e. M_N^ϵ decides $L(N^\epsilon)$ in polynomial time. Thus, we have that $P^\epsilon = NP^\epsilon$.

Fix an oracle ϵ. Let u be the integer coding the machine U^ϵ above. What we showed above is that $\boldsymbol{P}^\epsilon = \boldsymbol{NP}^\epsilon$ if there exists a deterministic Turing machine index i such that M_i operates in polynomial time and decides the same language as U. The converse is clearly trivial - if $\boldsymbol{P}^\epsilon = \boldsymbol{NP}^\epsilon$, then there is a polynomial time deterministic machine for everything, including U. Stated in terms of the functions we've defined earlier, we have

$$\epsilon \in \mathcal{O}^c \iff \exists i, c, k \forall n[(\mathcal{M}_\epsilon^t(i, n, k, c) = 1) \wedge (\mathcal{N}_\epsilon^a(u, n, 1) = 1 \iff \mathcal{M}_\epsilon^a(i, n, k, c))]$$

Thus, $\mathcal{O}^c$ is a $\mathbf{\Sigma}_2^0$ set, and subsequently $\mathcal{O}$ is a $\mathbf{\Pi}_2^0$ set. □

Theorem 2.1. *$\mathcal{O}$ is $\mathbf{\Pi}_2^0$ hard.*

Proof. We will define a continuous function $f : \mathcal{C} \to \mathcal{C}$ such that $\epsilon \in H \iff f(\epsilon) \in \mathcal{O}$, i.e. f maps a binary sequence to an oracle relative to which $\mathbf{P} \neq \mathbf{NP}$ iff ϵ is infinitely often 1. For the proof, fix E to be an $\mathbf{EXP}$ complete language, and fix an effective numbering of all relativized Turing machines $\{M_m^A\}$, where A is an oracle language yet to be defined. By effective numbering, we mean one in which the mapping from integers to codes of Turing machines is computable, and such that every Turing machine appears infinitely often in the numbering.

Let $\epsilon \in \mathcal{C}$ be arbitrary. For any $i \in \mathbb{N}$, let $d(i) = |\{j \leq i : \epsilon(j) = 1\}|$. Building off of Solovay's original proof, we construct the output language $f(\epsilon)$ in stages, adding new strings to $f(\epsilon)$ at each stage. What we do at stage i depends on $\epsilon(i)$. In the end, it is our hope that the language

$$U = \{1^i : f(\epsilon) \text{ contains a string } x \text{ of length } i\}$$

will be in $\mathbf{NP}^{f(\epsilon)}$ but not in $\mathbf{P}^{f(\epsilon)}$. If $\epsilon(i) = 0$, then we add to $f(\epsilon)$ all of the strings $x \in E$ of length i.

Suppose that we are at a stage i such that $\epsilon(i) = 1$. In this case, we 'simulate' the $d(i)^{th}$ relativized Turing machine $M_{d(i)}^{f(\epsilon)}$ for $i^{log(i)}$ steps on input 1^i (i.e. i many 1's concatenated together). Suppose that during this computation, the machine queries the oracle $f(\epsilon)$ on some input x. We need to decide what the answer to the query is, if we haven't already. It will be clear during this description that at stage i, we will never add any strings to $f(\epsilon)$ which are not of length i. Thus, if $|x| < i$, the result of the query will have been settled on at an earlier step. If $|x| \geq i$, then we first check to see if $\epsilon(i) = 0$ and $x \in E$. If true, then we would have added x to the language earlier, and the query should result in a yes, so we decide to be proactive and add it in early. Otherwise, we decide that x should not be included, i.e. the query results in a no. To keep track of the strings which we have committed to never adding to $f(\epsilon)$, we add x to a list of exceptions, which we denote X.

Suppose that the machine $M_{d(i)}^{f(\epsilon)}$ halts in rejection on input 1^i in less than or equal to $i^{log(i)}$ steps. We wish to add a string to $f(\epsilon)$ of length i, so that if this machine halts in polynomial time, then the language it decides cannot possibly be U. Note that there are 2^i possible binary strings which we could add to $f(\epsilon)$. The only strings which we are not allowed to add are those in X. Towards obtaining an upper bound for $|X|$, we consider the worse than worst case in which for all $k \leq i$, $\epsilon(k) = 1$, and all of the Turing machines which have been simulated have queried the oracle at every step of the simulation, on a string which was added to X. In this case, $|X| = \sum_{j=0}^{i} j^{log(j)}$. A simple induction shows that this sum is strictly less than 2^i for all $i \in \mathbb{N}$, and so there is guarunteed to be an available string which we have no yet committed to, which we can add to $f(\epsilon)$. We do this, and move on to the next stage. We choose explicitly to add the minimum of the strings (lexicographically) not in X to acknowledge that which string would have been chosen is computable.

Suppose next the case that the machine $M_{d(i)}^{f(\epsilon)}$ halts in acceptance on input 1^i in less than or equal to $i^{log(i)}$ steps. In this case, we add nothing, simply moving on to the next stage. Since we are only adding strings of length i at stage i, we have assurance that this machine disagree with U at this input. If the machine doesn't halt at all, we also do nothing. This completes the construction of $f(\epsilon)$. We turn now to showing that it works like it's supposed to.

Suppose first that $\epsilon(i) = 1$ only finitely often. Then there exists a $k \in \mathbb{N}$ such that for all $j > k$, $f(\epsilon) = 0$. Then up to a finite number of strings which are added in the stages where ϵ is 1, call it P, and the finite number of strings from E which are not added at those same stages, call it Q, we have $f(\epsilon) = (E - Q) \cup P$, which is still **EXP** complete by the results of the first section. Also by results of the first section, this is enough to be certain that $\mathbf{P}^{f(\epsilon)} = \mathbf{NP}^{f(\epsilon)}$.

Conversely suppose that $\epsilon(i) = 1$ infinitely often. To see that $U \in \mathbf{NP}^{f(\epsilon)}$, just note that we can decide if a string of length i is in $f(\epsilon)$ by simply nondetermistically guessing from the set of all such strings, and querying the oracle. Thus this language is in fact decidable in constant $O(1)$ nondeterministic time. By way of contradiction suppose that for some $m \in \mathbb{N}$, M_m is a machine which decides U in polynomial time, say time n^k. Let $j \in \mathbb{N}$ be such that for all $i \geq j$, $i^k < i^{log(i)}$. Note that because ϵ is infinitely often one, the function d is surjective, and so along with our assumption that all Turing machines occur in our numbering infinitely often, there must exist an $l \geq j$ so that $M_m = M_{d(l)}$. Note that $M_m(1^l)$ then has to halt in less than $l^{log(l)}$ steps. If it rejects, then by our construction we will have added a string of length l to U, meaning that M_m is not correctly deciding U. If M_m accepts, then we have the same issue - M_m can't possibly give the correct answer about U at 1^l, i.e. M_m cannot decide U, a contradiction. We conclude that if ϵ is 1 infinitely often then $U \notin \mathbf{P}^{f(\epsilon)}$, and so $\mathbf{P}^{f(\epsilon)} \neq \mathbf{NP}^{f(\epsilon)}$.

It remains to show that our function f is continuous. Towards this end, we show first that membership in $f(\epsilon)$ always depends on a finite initial segment of ϵ. Let x be a string of length n. We can decide if $x \in f(\epsilon)$ by running through the first n stages of the construction of $f(\epsilon)$, where $n = |x|$. Suppose that $\epsilon(n) = 0$. Then $x \in f(\epsilon)$ if and only if $x \in E$, which can be decided by consulting one of the first $2^{|x|}$ entries of ϵ. If $\epsilon(n) = 1$, then we would only add x to $f(\epsilon)$ if the machine $M_n^{f(\epsilon)}$ halted in rejection on 1^n, and x is the smallest string of length n which has not been queried at any stage i of the construction for which $\epsilon(i) = 1$. Care needs to be taken here, since during these stages, the machine may need to decide if a string of length greater than n needs to be added to $f(\epsilon)$ early. However, finitely many such queries need to be considered, so choosing the maximum of all of these gives an initial segment of ϵ which completely determines whether $x \in f(\epsilon)$.

Subsequently, if $B(\sigma, N)$ is a basic open set in $\mathcal{C}$, i.e. the collection of sequences extending the initial segment which is the first N entries of σ, then $\epsilon \in f^{-1}(B(\sigma, N))$ iff the first N entries of $f(\epsilon)$ agree with σ. For each $m \leq N$, deciding if $f(\epsilon)(m) = \sigma(m)$ amounts to determining if the m^{th} string is in $f(\sigma)$. If the m^{th} string is of length n, then we can determine this via conditions on some of the initial $g(n)$ entries of σ itself, for some function g, as shown earlier. Define

$$U_m = \bigcup \{B(\epsilon, g(|x_m|)) : \text{first } g(n) \text{ entries of } \epsilon \text{ meet the requirements for } f(\epsilon)(m) = \sigma(m)\}$$

(Where x_m is the m^{th} binary string.) Then

$$f^{-1}(B(\sigma, N)) = \bigcap_{m=1}^{N} U_m$$

So the inverse image of a basic open set is a finite intersection of open sets in $\mathcal{C}$, i.e. open. $f(\epsilon)$ is continuous, completing the proof. □

Theorem 2.2. *$\mathcal{O} \in \Sigma^0_3$. (And consequently $\mathcal{O} \in \mathbf{\Sigma}^0_3$.)*

Proof. Let ϵ be some fixed language. $\{M^\epsilon_i\}_{i\in\omega}$ be an enumeration of all deterministic Turing machines (relativized to ϵ), and $\{N^\epsilon_j\}_{j\in\omega}$ be an enumeration of all nondeterministic Turing machines (also relativized to ϵ). If there is no ϵ superscript, then we are referring to an enumeration over all unrelativized Turing machines. Define the functions

$$\mathcal{N}^a(i,n,k) = \begin{cases} 1 & \text{if } N_i(n) \text{ halts in acceptance in time lss than } |n|^k \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$\mathcal{N}^t(i,n,k) = \begin{cases} 1 & \text{if } N_i(n) \text{ halts in time less than } |n|^k \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$\mathcal{M}^a(j,n,k) = \begin{cases} 1 & \text{if } M_j(n) \text{ halts in acceptance in time less than } |n|^k \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$\mathcal{M}^t(j,n,k) = \begin{cases} 1 & \text{if } M_j(n) \text{ halts in less than } |n|^k \text{ steps} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$\mathcal{M}^a$ and $\mathcal{M}^t$ are both clearly recursive. $\mathcal{N}^t$ and $\mathcal{N}^a$ are as well, but should be clarified a bit. What we mean by '$N_i(n)$ halts in acceptance' is that there exists an accepting path in the configuration graph of $N_i(n)$, and what we mean by '$N_j(n)$ halts in time less than $|n|^k$ steps' is that all paths of uniquely occurring vertices in the configuration graph of $N_i(n)$ (i.e. all of those paths without loops) have length less than $|n|^k$. Since both functions only require checking a finite number of paths, both of these claims are easily recursive to check for any i,n,k, so the functions $\mathcal{N}^a$ and $\mathcal{N}^t$ are recursive. For all of these functions, denote $\mathcal{M}^t_\epsilon$ and so forth to be the same functions, but with machines relativized to ϵ.

Now note that

$$\epsilon \in \mathcal{O} \iff \exists i,k_1 \forall j,n_1 \left[\mathcal{N}^t_\epsilon(i,n_1,k_1) \wedge \left(\left(\exists k_2 \forall n_2 \mathcal{M}^t_\epsilon(j,n_2,k_2)\right) \Rightarrow \exists n_3(\mathcal{N}^a_\epsilon(i,n_3,k_2) \neq \mathcal{N}^a_\epsilon(j,n_3,k_1))\right)\right] \tag{5}$$

This sentence makes the claim that there exists some relativized nondeterministic Turing machine N^ϵ_i which halts in time polynomially bounded (degree k_1) by it's input for all inputs (n_1 being the dummy input), meaning that it decides some language $L_1 \in \mathbf{NP}^\epsilon$, and which has the property that if $M^{epsilon}_j$ is any polynomial time bounded Turing machine (degree k_2) (meaning it decides some language $L_2 \in \mathbf{P}^\epsilon$), then there must exist an input (n_3) on which N^ϵ_i and M^ϵ_j disagree (meaning that $L_1 \neq L_2$). Thus this is a way of stating that $\mathbf{P}^\epsilon \neq \mathbf{NP}^\epsilon$ without quantifying over any non-type 0 objects. Putting this expression into prenex normal form gives the equivalence:

$$\epsilon \in \mathcal{O} \iff \exists i,k_2 \forall j,n_1,k_2 \exists n_2,n_3 \left[\mathcal{N}^t_\epsilon(i,n_1,k_1) \wedge \neg\mathcal{M}^t_\epsilon(j,n_2,k_2) \vee (\mathcal{N}^a_\epsilon(i,n_3,k_2) \neq \mathcal{N}^a_\epsilon(j,n_3,k_1))\right] \tag{6}$$

Someone more acquainted with this kind of thing would likely be convinced at this point that $\mathcal{O} \in \Sigma^0_3$ and thus $\mathbf{\Sigma}^0_3$, but I am still coming to grips with this stuff so I'm going to keep going. According to Moschovakis, a set $P \subseteq \mathcal{X}$ is Σ^0_1 iff there is a semirecursive $P^* \subseteq \omega$ such that

$$P(x) \iff \exists s[x \in N(\mathcal{X},s) \wedge P^*(s)]$$

Fix the integers i,k_2,j,n_1. Define the relation $P_{i,k_2,j,n_1} \subseteq \mathcal{C}$ by $\epsilon \in P_{i,k_2,j,n_1}$ iff the existential subexpression to the right of the first two quantifiers in (6) is true, and for an integer n, let $(n)_i$ denote the i^{th} digit of the sequence coded by n, under the typical primitive recursive encoding using the prime numbers. Then we can let $Q'(n_1,n_2,\epsilon)$ be the subexpression within the scope of the quantifier for P_{i,k_2,j,n_1}, and immediately define $Q^* \subseteq \omega \times \mathcal{C}$ by $Q^*(n,\epsilon) \iff Q((n)_1,(n)_2,\epsilon)$.

Take a recursive enumeration of the eventually 0 sequences in $\mathcal{C}$, $\{\sigma_s\}_{s\in\omega}$ (the typical dense set in a recursive presentation). Note that for any $s \in \omega$, the s^{th} basic neighborhood of $\mathcal{C}$ is going to be the collection of all binary sequences extending some initial segment, whose digits are computable, where the length of the initial segment is also computable - call this function $r(s)$, for $s \in \omega$. We define the partial recursive relation $P^* \subseteq \omega^2$ via the computation of a Turing machine, which works in the following way: Given a pair (n, s), the Turing machine begins by writing the first $r(n)$ digits of the binary string σ_s, and then proceeding to compute the relation $Q^*(n, \sigma_t)$, where σ_t is the string it just wrote as a 'partial' oracle (everything to the right of that is assumed to be 0. What we mean by this is that if the relativized machine deciding Q^* were to make a query, it uses whatever oracle information it has on the 'initial segment', and if the information it needs is not available, it goes into an 'error' state in which it simply moves it's cursor right forever, never halting. We define $(n, s) \in P^*$ iff the machine just describes halts and accepts. This relation is clearly semirecursive, and if the machine accepts, then the 'relativized' version of the relation Q^* would halt and accept relative to ϵ, using only the initial $r(n)$ length segment of ϵ. Of course, a Turing machine halting in finite time can only ever use a finite initial segment of it's oracle, so we have that

$$P_{i,k_2,j,n_1}(\epsilon) \iff \exists n[\epsilon \in N(\mathcal{C}, (n)_1) \wedge P^*((n)_1, (n)_2)]$$

In other words, we simply replaced 'computable relative to an oracle' with 'computable using some initial segment of an oracle which may vary'. Okay, I'm starting to see the connections here and why this could be obvious. Anyway, this confirms that P_{i,k_2,j,n_1} is Σ^0_1, and it immediately follows that $\mathcal{O}$ is itself Σ^0_3. □

The $\mathcal{M}$ and $\mathcal{N}$ functions from the above proof can be slightly modified and used to quickly classify a lot of other sets. For instance, note that for any k. For instance, let

$$\mathcal{M}^{a'}(j, n, k, c) = \begin{cases} 1 & \text{if } M_j(n) \text{ halts in acceptance in time less than } c|n|^k \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

$$\mathcal{M}^{r'}(j, n, k, c) = \begin{cases} 1 & \text{if } M_j(n) \text{ halts in rejection in time less than } c|n|^k \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Then

$$\epsilon \in \boldsymbol{TIME}(n^k) \iff \exists j, c \forall l[(\epsilon(l) = 1 \Rightarrow \mathcal{M}^{a'}(j, l, k, c)) \wedge (\epsilon(l) = 0 \Rightarrow \mathcal{M}^{r'}(j, l, k, c))] \tag{9}$$

Clearly the expression within the quantifiers is computable relative to ϵ for a fixed j, c, l, and computable unrelativized using only the first l entries of ϵ, so by the same argument as above it follows that $\boldsymbol{TIME}(n^k) \in \Sigma^0_2$. Since the Σ classes are closed under existential quantification, it follows that $\mathbf{P} \in \Sigma^0_2$, and by an identical argument we have the exact same facts for the analogous nondeterministic classes, as well as (possibly?) **BPP** and **BQP**.

Actually, $\mathbf{P}$ has to be a Δ set, if anything.

3 The Mathias Forcing

The fundamental inspiration for the above was Solovay's proof that there exists an oracle $\mathcal{O}$ relative to which $\boldsymbol{P}^{\mathcal{O}} \neq \boldsymbol{NP}^{\mathcal{O}}$. A related but stronger claim would be the existence of an oracle relative to which $\boldsymbol{NP}^{\mathcal{O}} \neq \boldsymbol{coNP}^{\mathcal{O}}$. This modification isn't too difficult to do:

Lemma 3.1. *There exists an oracle $\mathcal{C}$ relative to which $\boldsymbol{NP}^{\mathcal{C}} \neq \boldsymbol{coNP}^{\mathcal{C}}$.*

Proof. Let $\{N_k^{\mathcal{C}}\}_{k=1}^{\infty}$ be an enumeration of all NP machines, relativized to $\mathcal{C}$. At stage n, run the n^{th} machine on the input $0^{f(n)}$, for $f(n)^{\log(f(n))}$ steps, where $f(n)$ is a function to be discussed shortly. Before that, however, let's take a moment and consider what actually needs to be done. We would like to create an oracle such that the language

$$U = \{0^n : \text{there exists an } x \in \mathcal{C} \text{ of length } n\}$$

which is obviously in $\boldsymbol{NP}^{\mathcal{C}}$, to not be in $\boldsymbol{coNP}^{\mathcal{C}}$. That is to say, we need to diagonalize out so that the language coU is not decided by any $\boldsymbol{NP}^{\mathcal{C}}$ machine. Working off of the original Solovay process, this appears simple enough: Run the $f(n)^{th}$ machine on $0^{f(n)}$ for $f(n)^{\log(f(n))}$ steps. This is a nondeterministic machine, so to 'halt within this time' means to have an accepting path of that length. If this is the case, then we can ensure that this machine does not decide *coU* by adding a string x of length n to the oracle, thus ensuring that $0^n \notin coU$, and guaranteeing an error for this machine. Of course, as before, we need to be careful about our commitments - if the machine queries strings along the way of ANY of the $2^{f(n)}$ many paths, we must have either made a decision already or commit to not adding it later. Specifically, if the queried string x is length less than $f(n)$, then we already have an answer by what follows, and if the string of length greater or equal to n, we commit to never adding it to the oracle and return a no. So far nothing is different. However, suppose we add a string x at stage $n+1$. If $f(n)$ is not sufficiently fast growing, it could be the case that there are no strings which we can add without violating prior commitments. This problem can be avoided by making $f(n)$ sufficiently fast growing that anything we would add at stage $n+1$ is larger than anything which the machine N_n could possible wish to query in $f(n)^{log(f(n)}$ many steps. Specifically, we need $f(n+1) > f(n)^{log(f(n)}$, which can be achieved explicitly by $f(n) = 2^{2^{2^{2^n}}}$ ($2^{2^{2^n}}$ is not fast enough!) The only remaining discrepancy to consider in the process is the possibility that, because the machine has $2^{f(n)}$ many computational paths, and can easily query many strings of length $f(n)$ along any of them, that we end up making negative commitments on every string of this length. In this situation it appears we have nothing to add, but all is not lost. On the *actual* accepting path, the machine can only query $f(n)^{\log(f(n)}$ many strings, far less than the total number of strings of length $f(n)$, $2^{f(n)}$. By adding a string of this length which is possibly queried along some other computational path might change the outcome of that path from a rejecting path to an accepting path, but that doesn't change the 'output' of the computation, so is fine. Noting that from all of this, we always have strings to add, and never add a string at stage n which is not length $f(n)$, we have finished describing the oracle. It remains to show that this works. Suppose there is some machine N_m which decides *coU*, i.e. for any n, $N_m(0^f(n))$ is accepting iff $0^{f(n)} \in coU$. Since this is an $\boldsymbol{NP}$ machine, it has a runtime bound $f(n)^k$ for some k. Pick n sufficiently large that $f(n)^k < f(n)^{log(f(n))}$. As before, we are assuming an infinitely redundant enumeration, so there must be an index $m' > n$ such that N_m and N'_m are equivalent machines. Consider $N_{m'}(0^{f(m')}$. If an accepting path exists of length less than $f(m')^k$, then since this length is less than the number of steps we ran the machine for, we must have detected it, and thus we must have specifically added a string x of length $f(m')$ to the oracle $\mathcal{C}$, so that $0^{f(m')} \in U$, so that $0^{f(m')} \notin coU$. Thus the machine N_m, identical to $N_{m'}$ but for the index, can't possibly decide *coU*. Otherwise, no accepting path exists. If we are assuming that these machines are precise, [NEED TO FINISH] ☐

Subsets of natural numbers can be naturally associated with oracles, by virtue of each number being naturally associated with a string. Alternatively, each natural number can be seen as representing a position in the infinite binary string coding the oracle. Either way, for what follows we will see oracles as first and foremost, subsets of ω.

Lemma 3.2. *There exists a set $C \subseteq \omega$ such that for all $D \subseteq C$, $\boldsymbol{NP}^D \neq \boldsymbol{coNP}^D$. That is to say, C is a homogeneous set for the collection of oracles relative to which $\boldsymbol{P}$ is not equal to $\boldsymbol{NP}$.*

Proof. Suppose we found a set A which can't possibly contain a homogeneous for $\boldsymbol{NP} = \boldsymbol{coNP}$. Then by the Prikry property, it has to be the case that it contains a homogeneous set for $\boldsymbol{NP} \neq \boldsymbol{coNP}$, and we'll have demonstrated the existence of one. Thus, this will be our starting point. We must create a set $A \subseteq \omega$ such that for each $B \subseteq A$, there exists a $C \subseteq B$ such that $\boldsymbol{NP}^C \neq \boldsymbol{coNP}^C$.

To do so, our set A will have exactly 1 string of length $f(n)$ for each n, where again $f(n)$ is a fast enough growing function that at any step of the construction that follows no strings of this length could possibly be queried in any previous machine's simulated computation (see the above proof). Suppose we have A restricted to $f(n)$ defined already. At step n, take all $m \leq n$ $\boldsymbol{NP}$ machines, run them each for $f(n)^{log(f(n))}$ many steps on the input $1^{f(n)}$. In fact, we will do this for each of the possible 2^n many *subsets* of A restricted to $f(n)$ - that is, for each subset of the partially constructed oracle. Suppose one of these machines, for one of these subsets, encounters an accepting path on $1^{f(n)}$. Then what we would like to do is add a string x of length $f(n)$ which, for any partial oracle, and given any of the partial oracles which have accepting paths, is never queried along at least one of each such path, so that these paths remain accepting regardless of what partial subset of A is chosen in what follows. Can this be done? Along any particular path of length $f(n)^{log(f(n)}$, we can of course query a maximum of $f(n)^{log(f(n))-1} = O(f(n)^{log(f(n))}$ many strings of length

$f(n)$. There are at most 2^n such paths for each machine, and at most n machines, and therefore the total number of strings which we cannot add is $n2^n f(n)^{log(f(n))} << 2^{f(n)}$. Thus we will always have such a string to add. This process inductively defines the set A.

Now, let $B \subseteq A$ be infinite. Let $g(n)$ be the length of the n^{th} element of B. We now begin to construct a subset $C \subseteq B$ such that $\boldsymbol{NP}^C \neq \boldsymbol{coNP}^C$. The process is virtually the same as above except that we have to delay the diagonalization further depending on what is available in the set B. More precisely, at stage n, we run the n^{th} nondeterministic machine on the input $1^{g(n)}$ for $g(n)^{log(g(n))}$ many steps. If the machine encounters an accepting path, we wish to add a string of length $g(n)$ from B which has not been queried previously previously for any earlier machine, and also which is not queried at all along at least one of the accepting paths of this n^{th} machine. Of course, the single string of length $g(n)$ which fits this criteria is in B, because $g(n) = f(k)$ for some $k \geq n$, and at stage k of the construction of A, we ran machine $n \leq k$ for $1^{f(k)} = 1^{g(n)}$ for $f(k)^{log(f(k))} = g(n)^{log(g(n))}$ many steps. Since machine n had an accepting path on $1^{g(n)}$, that's the same as saying that machine n had an accepting path on $1^{f(k)}$, and so in the construction of A we definitely added a string of length $f(k) = g(n)$. By the premise of the construction we know that there is a string of length $g(n)$ in B, and since $B \subseteq A$ and A contains exactly one string of each length for any length represented, that this string must be the one in B. Thus we can deposit this string into C. This is how we construct C.

Now, we consider the language $U = \{1^n : \text{there exists a string } x \text{ of length } n \text{ in } C\}$. This language is obviously in $\boldsymbol{NP}^C$. We can also show that it is not in $\boldsymbol{coNP}^C$. Suppose it were. Then coU would be in $\boldsymbol{NP}^C$, and there would be a polynomial time $\boldsymbol{NP}$ machine, say the m^{th} one, N_m, which decides coU in time n^k for some k. Choose $m' \geq m$ such that m' indexes the same machine as m, but also such that $g(m')^{log(g(m'))} > g(m')^k$, and consider $N_{m'}(1^{g(m')})$. If this accepts, then of course there is an accepting path of length $g(m')^k$, and so in our construction when we ran this machine for $g(m')^{log(g(m'))}$ many steps we will have added a string to C from B of length $g(m')$, making it that $1^{g(m')} \notin coU$. However since $N_{m'}(1^{g(m')}) = N_m(1^{g(m')}$, we have then that this machine makes an error on this input. Thus is must be the case that N_m rejects $1^{g(m')}$, i.e. there is no accepting path of length $g(m)^k$. Since we are assuming the machine decides the language coU in time n^k, we must assume no paths are any longer than $g(m')^k$, meaning that it could not be the case that $N_{m'}(1^{g(m')})$ has any accepting paths either. Thus we never added a string of length $g(m')$ to U, and thus $1^{g(m')} \in coU$, meaning that again, our machine makes an error on this input. Thus it can't possibly be the case that N_m decides coU, and we are done. □

What we've effectively shown above is that with respect to the Mathias forcing, our principle set $\mathcal{O}$ 'isn't measure 0'. It would of course be much more satisfying if we could show that it is 'measure 1'. To do this, we would need to be able to construct a homogeneous set directly from an arbitrary set A, rather than starting with a particular A suited for the job, as we did above. Unfortunately, the following shows that this question is as difficult as the $\boldsymbol{P}$ vs $\boldsymbol{NP}$ problem itself:

Theorem 3.1. *Suppose that $f : \mathbb{N} \to \mathbb{N}$ is any increasing time constructible function, and consider the set $A = \{1^{f(n)} : n \in \omega\}$, and that $\boldsymbol{P} = \boldsymbol{NP}$. Then for all $O \subseteq A$, we have that $\boldsymbol{P}^O = \boldsymbol{NP}^O$.*

Proof. Assume the above hypotheses, and let N be an arbitrary machine deciding a language in $\boldsymbol{NP}$, say in time n^k (and we will assume for the sake of simplicity that this machine is precise - that is to say, all computational paths are exactly length n^k.) Let $O \subseteq A$, for A as above. Consider the input string x where $|x| = m$. Note that in time m^k, since the machine is precise m^k is the longest length of any string which could be possibly queried along any computational path. Furthermore, since we know that O is drawn out of a very simplistic set with exactly one string of each possible length, and thus we can, in polynomial time, have a deterministic machine create a *partial lookup table* which can substitute for the oracle O in the computation of $N^O(x)$. Specifically, let M_1^O be a machine which, on input an input of length m, computes $f(n)$ for each n such that $f(n) \leq m^k$, queries O as to whether or not $1^m \in O$, and creates a lookup table with all of these answers. This is a polynomial number of computations, all of which can be done in polynomial time, and thus this machine clearly operates in polynomial time. We denote this output table T_m, and assume it to take the form of a string of 0's and 1's, like a partial oracle. It is probably worth noting that this output cannot be exactly seen as a partial oracle, because to do that we would need an exponential length string of mostly 0's. Since we know that our "master set" is just A, the n^{th} digit of the string can be seen as denoting whether or not $1^{f(n)} \in O$, rather than simply that the n^{th} string is in O, lexicographically enumerating them all. We need the output to be polynomial length, and this ensures that we have it that way.

Despite the above consideration, we will treat these partial oracles like actual oracles in the sense that N^{T_m} will be a slight abuse of notation, really denoting N relativized to the oracle whose bits are determined by T_m in the obvious way (meaning that eventually the string is, of course, all 0's, hence being "partial"). With this notation in mind, we note that for our non-relativized N, there must exist a machine N' such that for any x, we have $N'(T_{|x|}, x) = N^{T_{|x|}}(x)$. To see this precisely, suppose the machine $N^{T_{|x|}}(x)$, along some computation path, encounters a query state. We can, instead of querying the oracle, instead add a deterministic extension of the path, which first checks to see if the string being queried is of the form $1^{f(n)}$ for some n, and then consults the input $T_{|x|}$ to confirm whether or not it is in "the oracle". Since $T_{|x|}$ is polynomial length in $|x|$, as are the computations of $f(n)$, these extensions are themselves polynomial length, and so our machine N' operates in polynomial time. Note however that N' is a nonrelativized machine, and thus we can now invoke the hypothesis that $\boldsymbol{P} = \boldsymbol{NP}$: There must exist a polynomial time machine M_2 such that $M_2(T_{|x|}, x) = N'(T_{|x|}, x)$ for all x. Trivially, we may assume that $M_2 = M_2^O$ (simply add a query string and query states which are never encountered by the transition function).

Note now that $N^O(x) = M_2^O(M_1^O(x), x)$. This this concatenation of machines is a deterministic polynomial time machine which decides the same language as N^O, and thus we have that $\boldsymbol{P}^O = \boldsymbol{NP}^O$. $\square$

It of course follows from this that if $\mathcal{O}$ were "measure 1" with respect to the Mathias forcing - that is to say, for all $A \subseteq \omega$, there exists an $O \subseteq A$ such that $\boldsymbol{P}^O \neq \boldsymbol{NP}^O$, we will also have proven by the above contrapositively that $\boldsymbol{P} \neq \boldsymbol{NP}$. Thus, until such as a time as the *bmP* vs $\boldsymbol{NP}$ question is officially settled, we have shown as much as can be shown about $\mathcal{O}$ under the Mathias forcing. It is "not measure 0", but we will not know that it is "measure 1" until the great question is answered.

3.1 Further Directions

Theorem 3.2. *Assuming* $\boldsymbol{P}$ *is complete at some level of the lightface hierarchy,* $\boldsymbol{P}$ *is* Δ^0_2 *complete.*

Proof. Suppose it weren't. Then since we've shown that $\boldsymbol{P} \in \Sigma^0_2$, and we're assuming it is complete for some pointclass, we have that it must be complete for Σ^0_2. But then **coP** would be Π^0_2 complete (right?) and since these classes are distinct over $\mathcal{C}$, we have that $\mathbf{coP} \neq \mathbf{P}$, which is false. $\square$

Thus there must be a way to express membership in **P** as a Π_2 statement as well. Because expressing membership in **P** is so similar to expressing membership in **NP**, it stands to reason that both of these classes are Δ^0_2 complete. But if this were true, then **coNP** would also be Δ^0_2 complete. Thus, it would seem at first glance that distinguishing **P**, **NP**, **coNP**, and similarly natural 'simple classes' is going to be harder than simply classifying where they sit in the lightface hierarchy. However, what about more complicated classes, such as **PH**, **PSPACE**, **AM**, **PP**, and $\boldsymbol{P}^{\#\boldsymbol{P}}$? None of these classes are known to be different than **P** and **NP**. If even one of these is complete for a higher up class, then we will have proven the difference, no? **PH** in particular, the union of all levels of the polynomial hierarchy, seems like it might 'more complicated enough' to be different, and if $\mathbf{PH} \neq \mathbf{P}$ then $\mathbf{P} \neq \mathbf{NP}$. I'm no expert on this stuff yet, but this seems like a potentially very lucrative direction to go in. Below is an attempted proof sketch that **P** is Σ^0_2 complete. Obviously there has to be a flaw somewhere in the argument, but regardless of whether it's sound or not, I thought it was interesting in it's simplicity. If you haven't read the lemmas and definitions starting at 1.9, you should do that before reading this proof.

Proof. Note that the set

$$F = \{\epsilon \in \mathcal{C} : \epsilon \text{ is 1 finitely often}\}$$

This set is clearly Σ^0_2 complete by it's complementary set being Π^0_2 complete. Let $\epsilon \in \mathcal{C}$ be arbitrary. Let E and P be **EXP**-complete and **P**-complete problems, respectively. We define the language $f(\epsilon)$ in stages. Let $d(i)$ be the number of times that a 1 has occurred in ϵ by the i^{th} digit, and $e(i) = i - d(i)$ be the number of 0's. At stage i, if $\epsilon(i) = 1$, then we add to $f(\epsilon)$ all strings from E which are of length $d(i) - 1$, appended in the beginning with the character 1. And, if $\epsilon(i) = 0$, then we append to $f(\epsilon)$ all strings from P which are of length $e(i)$, appended with the character 0. This defines the functions f.

Now, if ϵ is infinitely often 1, then $E \subseteq f(\epsilon)$, in particular $f(\epsilon) = E \sqcup A$ for some $A \subseteq P$. This A is a subset of P but it is not necessarily in **P**. In fact, whether or not it is even computable is dependent on the computability of ϵ. However, this isn't necessarily a problem. All that matters is that $\boldsymbol{EXP} = \boldsymbol{s}(E) \subseteq$

$\boldsymbol{s}(E \sqcup A)$. Since **P** is proper in **EXP**, it is therefore impossible that $E \sqcup A \in \mathbf{P}$, since then $\boldsymbol{s}(E \sqcup A) \subseteq \mathbf{P}$, and we would have $\mathbf{EXP} = \mathbf{P}$. Thus, if $\epsilon \notin F$, then $f(\epsilon) \in \mathbf{P}$.

If ϵ is only finitely often 1, then $f(\epsilon)$ is a **P** complete language minus minus a finite set, so still a **P** complete language. It seems then that $\epsilon \in F \iff f(\epsilon) \in \mathbf{P}$.

We now call into question the continuity of F. As before, it should suffice to show that for any n, the membership of the n^{th} string in $f(\epsilon)$ depends on a finite initial segment of ϵ. It clearly does though, since we only ever add strings of length i at the i^{th} step of the construction, to determine if $x \in f(\epsilon)$, where $n = |x|$, we see if $\epsilon(n) = 1$, and if it is, then $x \in f(\epsilon) \iff x \in E$, and if $\epsilon(n) = 0$, then $x \in f(\epsilon) \iff x \in P$.

These continuity arguments are still where I am weakest, so I have a feeling this is where the issue is. □

A natural measure on $\mathcal{C}$ by simply considering the probability experiment of drawing a sequence at random, where all are equally likely. The induced probability measure is clearly equivalent to the Lebesgue measure on $[0,1] \subseteq \mathbb{R}$ via the typical homeomorphism between these spaces. Equipped with this, we can talk about the probability that two classes are equal, by setting the randomly drawn language as an oracle which our classes are relativized. We can thus view the set $\mathcal{O}$ as an event, i.e.

$$P(A \in \mathcal{O}) = P(\boldsymbol{P}^A \neq \boldsymbol{NP}^A)$$

We wish to show that this probability is in fact 1. To show this, it suffices to show that given a random oracle A, we can out of A define a *test language* L^A, which has the property that $L^A \in \boldsymbol{NP}^A$ for any A, but $L^A \notin \boldsymbol{P}^A$ with probability 0. For a particular complexity class $\boldsymbol{C}^A$, we will let $\boldsymbol{C} = \{M_1^A, M_2^A, ...\}$ denote a recursive enumeration of the class, and will generally want to assume some particular well-behavedness regarding the relationship between the test language and this enumeration. These conditions have been called **oracle conditions**:

(1) The function $A \mapsto L^A$, as well as $A \mapsto M_j^A$ for each j, are all recursive for any chosen A. The result of this condition is that L^A is always computable given we are in a world relativized to A, and M_j^A is always not just computable, but in fact is in $\boldsymbol{C}^A$. (Isn't $M_j^A \in \boldsymbol{C}^A$ regardless?)

(2) For a finite string s, let s^*A denote the language obtained from replacing the first $|s|$ characters of A with the string s. For any oracle A, given any j and any finite string s, we require that there always exists an index k such that $M_k^A = M_j^{s^*A}$. This is clearly going to be true in any remotely robust complexity class. If a class has this property, we say that the class is **finitely patchable with respect to** A. Thus property 2 simply is the requirement that the class $\boldsymbol{C}$ be finitely patchable with respect to any oracle.

(3) For any oracle A, any $m \in \mathbb{N}$, any index j, and any A-recursive $0-1$ valued function ϕ^A, there should exist an index k such that
$$M_k^A(x) = \begin{cases} \phi^A(x) & \text{if } x < m \\ M_j^A(x) & \text{otherwise} \end{cases}$$
To augment our terminology above, we could summarize this condition by saying that the class $\boldsymbol{C}$ is finitely patchable with respect to any initial portion of any uniformally A-recursive language. Of particular importance is going to be that we let ϕ^A be the characteristic function of our test language L^A. The idea then will be that a machine which accepts the language M_k^A would agree with L^A for all inputs of length less than some number m, and gives the same answer as M_j^A otherwise.

(4) For any A, the language L^A has the property that each bit of the oracle A affects only finitely many bits of the language. (I'm back to being a little confused. Why isn't this the same requirement as condition 1?)

The following lemma based on these conditions will allow us to prove various separation results.

Lemma 3.3. *Let L^A be a test language and $\boldsymbol{C}^A = \{M_1^A, M_2^A, ...\}$ be a complexity class, and suppose that these two things satisfy conditions (1) through (4) above. Suppose further that there exists an $\epsilon > 0$ such that $P(M_j^A \neq L^A) > \epsilon$ for any j. Then $P(L^A \in \boldsymbol{C}) = 0$.*

Proof. Assume the conditions (1)-(4) for a randomly chosen test language L^A, a resulting relativized class $\boldsymbol{C}^A$, and existence of a constant ϵ satisfying the conditions described. Fix a machine M_j^A deciding a language in $\boldsymbol{C}^A$. For an integer $m \in \omega$, consider the class of oracles (i.e. the event) $\boldsymbol{C}_m = \{A : (\forall x < m) : L^A(x) = M_j^A(x)\}$ (where $x < m$ is shorthand for saying that x is a string of length < m). I.e. for a fixed m, $\boldsymbol{C}_m$ is the event that, at least up to strings of length m, the machine M_j^A agrees with the test language. Clearly it is the case that if $\lim_{m\to\infty} P(\boldsymbol{C}_m) = 0$ for any M_j^A, then the result follows. We show this by making use of our constant ϵ, by showing that for each m, there exists an $n > m$ such that $P(\boldsymbol{C}_n) \leq (1-\epsilon)P(\boldsymbol{C}_m)$.

By condition 1, M_j^A is a recursive function of A, i.e. it's output for each input requires a finite initial segment of A, and since the event $\boldsymbol{C}_m$ itself only requires a finite number of inputs, membership in this event is contingent on only a finite initial segment of A. Let Γ denote the set of all finite initial segments s of an oracle A such that $A \in \boldsymbol{C}_m$. For each of these, let $\boldsymbol{Z}_s$ denote the set of oracles which extend the string s. Then clearly each of these events is disjoint, and so we have partitioned the set

$$\boldsymbol{C}_m = \bigcup_{s\in\Gamma} \boldsymbol{Z}_s$$

Now we make use of condition 2 and 3. For a string $s \in \Gamma$, define

$$\psi_s^A(x) = \begin{cases} 1 & \text{if } L^{s^*A}(x) \neq L^A(x) \\ 0 & \text{otherwise} \end{cases}$$

Clearly this is a recursive function, via the recursiveness of L^A guaranteed by condition 1. Thus by condition 3, there exists an index k such that

$$M_k^A(x) = \left\{ \psi_s^A(x) \right.$$

□

For the following, given a random oracle A, define $\xi_A(x) = A(x1)A(x10)A(x100)\ldots A(x10^{|x|-1})$, i.e. it is the length $|x|$ string whose first bit is a 1 iff $x1 \in A$, whose second bit is 1 iff $x10 \in A$, and so forth. It will be worthwhile to make some observations about probabilities associated with this language. For a random oracle A, we must suppose that the probability that any particular bit is a 1 is exactly $\frac{1}{2}$, and that these events are all independent. For an integer n, consider the experiment of drawing a random $n - bit$ string y, and let X_n be the random variable representing the number of strings x such that $\xi_A(x) = y$. Note that $X_n = \sum B_n^x$, for each string x of length n, B_n^x equals 1 if $\xi_A(x) = y$ and 0 otherwise. Now, for any particular x, $\xi_A(x) = y$ iff $A(x1) = y_0$ and $A(x10) = y_1$ and ... and $A(x10^{n-1}) = y_{n-1}$, each of which is a probability $\frac{1}{2}$. Thus $P(\xi_A(x) = y) = (\frac{1}{2})^n$. Thus, considerations of independence aside, X_n is a sum of 2^n identically distributed Bernoulli random variables, and thus at least strongly approximates a Binomial random variable, i.e. $X_n\ Binomial(2^n, (\frac{1}{2})^n)$. Since the number of trials here is quite high while the probability of success is quite low, it follows that X_n approximates a Poisson distribution with $\lambda = 1$, the mean of X_n. Thus $P(X_n = k) \approx \frac{1}{ek!}$. These approximations get better as the length of the string drawn gets large. Of particular interest for us will be the fact that $P(X_n = 0) = P(X_n = 1) \approx \frac{1}{e}$, which represents the probabilities that a randomly drawn string will have either no inverses images or exactly 1 inverse image under ξ_A, respectively.

Theorem 3.3. *For a randomly drawn oracle A, $P(\boldsymbol{NP}^A \neq \boldsymbol{coNP}^A) = 1$. Of course it follows then that $P(\boldsymbol{P}^A \neq \boldsymbol{NP}^A) = 1$ as well.*

Proof. Begin by defining from A the language $RANGE^A = \{y : \exists x \xi_A(x) = y\} = \xi_A[\pm^*]$. It is clear that $RANGE^A \in \boldsymbol{NP}^A$ for any oracle A: simply consider the machine N^A which, upon some input y, guesses a random string x of the same length as y, and then computes $\xi_A(x)$ through $|x|$ queries of the oracle A, then checks to see if the result is y. Thus the test language $CORANGE^A = \{y : \forall x \xi_A(x) \neq y\} \in \boldsymbol{coNP}^A$. We wish to show now that this language $CORANGE^A$ is almost surely not in $\boldsymbol{NP}^A$. Note that the test language as well as a standard enumeration of the languages in $\boldsymbol{NP}$ clearly satisfy the oracle conditions specified above: thus by the lemma proven under those assumptions, to show that $P(CORANGE^A \in \boldsymbol{NP}^A) = 0$ it suffices to show that for any NP^A machine, there is an input z such that the probability that this NP^A machine disagrees with $CORANGE^A$ on this input is greater than $\frac{1}{3}$. In particular, consider a particular machine N^A, which halts in say, time n^k. Then there must exist an n sufficiently large that $n^k \leq \frac{2^n}{100}$. Then fixing $z = 0^n$, we note that, since the maximum length of any computational path is n^k, the maximum number of oracle queries to A which can be made along a particular path is no more than 1 percent of the total number of strings of length n. This will be significant for later.

Now, let $\boldsymbol{C}_0 = \{A : (\forall x)\xi_A(x) \neq 0^n\}$. Note that alternatively, $\boldsymbol{C}_0 = \{A : 0^n \notin RANGE^A\} = \{A : 0^n \in CORANGE^A\}$. Thus, from the standpoint of drawing an oracle at random, this set $\boldsymbol{C}_0$ represents the event that 0^n is in $CORANGE^A$. Similarly, define $\boldsymbol{C}_1 = \{A : \xi_A(0^n) \neq 0^n \wedge (\exists! x)\xi_A(x) = 0^n\}$. I.e. it is the set of oracles A such that 0^n has a single unique inverse image, which isn't itself. Note that since $A \in \boldsymbol{C}_1 \implies 0^n \notin CORANGE^A \implies A \notin \boldsymbol{C}_0$, we have that these events are mutually exclusive. We also have already the probabilities of each of these events: In the case of $\boldsymbol{C}_0$, we need only note that $P(\boldsymbol{C}_0) = P(X_n = 0) \approx \frac{1}{e}$, where X_n is the random variable defined above with $y = 0^n$. Similarly, $P(\boldsymbol{C}_1) = P(X_n = 1) = \frac{1}{e}$ as well (the extra condition that $\xi_A(0^n)$ is so minor as to pose no probabilistic difference in the events). Now, note then subsequently define

$$P(L(N^A) \neq CORANGE^A) \geq P((N^A(0^n) = 1 \wedge 0^n \notin CORANGE^A) \vee (N^A(0^n) = 0 \wedge 0^n \in CORANGE^A)) \tag{10}$$

$$:= \epsilon \tag{11}$$

Clearly this inequality holds since it's possible for the machine N^A to disagree with $CORANGE^A$ on other inputs besides 0^n. Thus the goal here is to show that $\epsilon > \frac{1}{3}$. We will do this by conditioning on our events $\boldsymbol{C}_0$ and $\boldsymbol{C}_1$. Suppose we are in the case of $\boldsymbol{C}_0$, i.e. $0^n \in CORANGE^A$. Then for N^A to disagree with $CORANGE^A$ on the input 0^n, it would be necessary for $N^A(0^n) = 0$. Let $\alpha_i = P(N^A(0^n) = 1 | A \in \boldsymbol{C}_i)$, for $i = 0, 1$. We now realize the sum of all our observations:

$$P(L(N^A) \neq CORANGE^A) \geq \epsilon = P((N^A(0^n) = 1 \wedge 0^n \notin CORANGE^A) \vee (N^A(0^n) = 0 \wedge 0^n \in CORANGE^A)) \tag{12}$$

$$= P(N^A(0^n) = 1 \wedge 0^n \notin CORANGE^A) + P(N^A(0^n) = 0 \wedge 0^n \in CORANGE^A) \tag{13}$$

$$> P(N^A(0^n) = 1 \wedge \boldsymbol{C}_1) + P(N^A(0^n) = 0 \wedge \boldsymbol{C}_0) \tag{14}$$

$$= P(N^A(0^n) = 1 | \boldsymbol{C}_1)P(\boldsymbol{C}_1) + P(N^A(0^n = 0 | \boldsymbol{C}_0)P(\boldsymbol{C}_0) \tag{15}$$

$$\approx \frac{\alpha_1}{e} + \frac{1 - \alpha_0}{e} = \frac{1 + \alpha_1 - \alpha_0}{e} \tag{16}$$

We complete the proof by comparing α_1 with α_0. Specifically, we will show that $\alpha_1 \geq \alpha_0$, so that $1 + \alpha_1 - \alpha_0 \geq 1$, and subsequently that $\epsilon > \frac{1}{e} \approx 0.36 > \frac{1}{3}$.

Consider the following transformation $A \mapsto A'$: Upon drawing the oracle A, we choose randomly an n-bit string $z \neq 0^n$, and then create A' by deleting from A all strings of the form $z10^i$ for $i < n$. By recalling the definition of ξ_A, this has the effect of forcing $\xi_{A'}(z) = 0^n$. In fact, for any other string $y \neq z$ (is z random?), we won't be changing the result of $A(y10^i)$, and so $\xi_{A'}(y) = \xi_A(y)$ for all other strings y, i.e. we disturb no other inputs except possibly z. Note that for any $A \in \boldsymbol{C}_0$, it will follow that $A' \in \boldsymbol{C}_1$. Suppose that $E \subseteq \boldsymbol{C}_1$ is some event, and consider $P(E)$. Note that every outcome in E is, up to a single digit of difference (where the selection on which they can differ bounded by a finite segment of the oracle), equal to an oracle in $\boldsymbol{C}_0$. Thus drawing a random oracle from $\boldsymbol{C}_1$ is probabilistically equivalent to drawing a random oracle from $\boldsymbol{C}_0$, and then drawing a random $n - bit$ string not equal to 0^n, and performing the above-described transformation. Effectively, what we've done here is altered the nature of the experiment, and have a correspondence of equality between the probabilities events of the original experiment, and events in the new experiment. To be more formal, we can replace the experiment of drawing a random oracle from $\boldsymbol{C}_1$ with that of drawing at random from $\boldsymbol{C}_0 \times \Gamma$, where $\Gamma = \{z \in \mathcal{C} : |z| = n \wedge z \neq 0^n\}$, and have the relation that for any event $E \subseteq \boldsymbol{C}_1$, $P(E) = P(\{(A, z) \in \boldsymbol{C}_0 \times \Gamma : A' \in E\})$. Thus, what we've defined is a probability preserving transformation between the uniform measure on $\boldsymbol{C}_1$, and the uniform measure on $\boldsymbol{C}_0 \times \Gamma$.

Now suppose we do what was just described: we draw a random oracle A from $\boldsymbol{C}_0$, along with a random n-bit string $z \neq 0^n$, and perform the transform $A \mapsto A'$, which by the above discussion belongs to $\boldsymbol{C}_1$.

With probability α_0, there is at least one accepting path in the computation tree of $N^A(0^n)$. Consider the lexicographical first of these paths. By our choice of n way back in the first paragraph of this proof, the conditional probability that this computation path does not include an oracle consultation of the random string z is at least 0.99. Since this is the only possible string on which A and A' can differ, with probability at least 0.99 then, this path will remain unaltered with respect to the oracle A', i.e. $P(N^{A'}(0^n) = 1 | N^A(0^n) = 1) \geq 0.99$. This is what we need to complete the proof. Note now that α_1 is an event in $\boldsymbol{C}_1$ (it's a conditional probability). Thus, combining the observations in the previous paragraph with those made in this one:

$$\begin{aligned} \alpha_1 &= P(\{(A,z) \in \boldsymbol{C}_0 \times \Gamma : N^{A'}(0^n) = 1\}) && (17) \\ &\geq 0.99 P(\{(A,z) \in \boldsymbol{C}_0 \times \Gamma : N^{A}(0^n) = 1\}) && (18) \\ &= 0.99\alpha_0 && (19) \end{aligned}$$

This completes the proof. Phew! (Revisions: I think that it doesn't really matter that we reach 0.99 in precision. Anything less than 1 will do, right?) □

Clearly this result is a statement about our protagonist class $\mathcal{O}$, namely that it has Lebesgue measure 1. With a similar approach, we can show not only that it is large in measure, but also large in category:

Theorem 3.4. *The set $\mathcal{O}$ is comeagre*

Proof. □

We now prove similar comparisons for other classes. Before doing this though we must address some complication regarding oracle complexity for space classes. How should the query tape count towards space usage? There are several different conventions for this. Two noteworthy ways are as follows:

- The query tape counts as much as a work tape as any other. Thus it is two-way and read/write.
- The query tape doesn't count as space usage. *However*, it is one way, write only, and erased automatically after a query.

It can be shown (should show) that it is only with the second definition that for a random oracle A, that $A \in \boldsymbol{L}^A$ with probability 1. It is for this reason that some complexity theorists consider this convention to be the more natural one; the point of an oracle is to trivialize certain types of problems, via trivializing one particular problem. If the supposedly trivialized language isn't in the class, this would seemingly go against the idea of oracle computing. We will thus choose to go with the second convention here. Define

$$\mathcal{S} = \{A : \boldsymbol{L}^A \neq \boldsymbol{P}^A\}$$

$$\mathcal{P} = \{A : \boldsymbol{P}^A \neq \boldsymbol{BPP}^A\}$$

Theorem 3.5. *$\mathcal{S}$ is measure 1, i.e. for a random oracle A, $P(\boldsymbol{L}^A \neq \boldsymbol{P}^A) = 1$.*

Proof. Define the following test language:

$$BIGQUERY^A = \{x : \xi_A(x) \in A\}$$

I.e. the problem of $BIGQUERY^A$ is the problem of determining if a given strings image under ξ_A is in the oracle itself A. Note first that $BIGQUERY^A \in \boldsymbol{P}^A$ for any A: The machine M, on input x, can compute $\xi_A(x)$ and write the result on query tape in $O(|x|)$ time, and halt in acceptance or rejection based on the result of the query, in one additional step.

However, why is this not necessarily in $\boldsymbol{L}^A$ trivially as well, under our convention? The problem is that the computation of $\xi_A(x)$ requires a query of the oracle for each bit, and each query erases the contents of the query tape. Thus in order to compute $\xi_A(x)$, we must eventually come to store the full result on an actual work tape, and this will in every instance use $|x|$ space. This problem is in fact intuitively why it will be the case that the desired result is true. Let us now actually prove it, though.

We are going to show that for any log-space bounded machine M^A, and for sufficiently large x, $P((M^A(x) = 1 \wedge x \notin BIGQUERY^A) \vee (M^A(x) = 0 \wedge x \in BIGQUERY^A)) \approx \frac{1}{2}$, from which it would clearly follow that any log-space bounded machine decides a language different from $BIGQUERY^A$ with probability 1, the largest possible ϵ for use in our lemma, accomplishing our goal in dramatic fashion.

Let us say that a string y of length n is *queriable* by a machine M if there exists an oracle X such that, at some point in it's computation, M^X queries the string y on input 0^n. If M is a logspace bounded machine, as we know quite well, there are only cn^k possible distinct configurations (for some constants c, k depending on the machine) for any input string of length n. Thus there are this many distinct configurations for an oracle logspace machine under the condition that we require the query tape to be blank. This will be the case at the initial step, and also following any step in which the machine performs a query. Now, after any query (as well as initially), the machine M^X will be in some configuration, one of the cn^k many. Note that the steps leading up to the next query are independent of the oracle X. Thus there are only at most cn^k unique strings that could be queried on the next query, and this collection is the same for the next query, and the previous query, and so forth. Thus we conclude that at most cn^k strings are queriable by and logspace bounded machine M.

Next we observe that, for a randomly drawn oracle A, $\xi_A(0^n)$ takes on one of 2^n possible values, all of which are equally likely. For a fixed logspace bounded machine M, consider the event

$$\mathbb{C} = \{A : M^A(0^n) \text{ queries } \xi_A(0^n)\}$$

This is a subclass of the set

$$\begin{aligned}\mathbb{Q} &= \{A : \xi_A(0^n) \text{ is queriable by } M\} \\ &= \{A : \exists X \text{ st } M^X(0^n) \text{ queries } \xi_A(0^n)\}\end{aligned}$$

From what we noted above, we can see that $P(\mathbb{Q}) = \frac{cn^k}{2^n}$ for some constants c, k, and so

$$P(\mathbb{C} \leq P(\mathbb{Q}) = \frac{cn^k}{2^n} \xrightarrow{n} 0$$

Thus, the class of oracles A such that $M^A(0^n)$ does not query $\xi_A(0^n)$, which is precisely $\mathbb{C}^c$, approaches 1 for large n. This of course bodes very poorly for the prospects of any logspace bounded machine deciding whether $0^n \in BIGQUERY$. To finish things off we make use of a measure preserving transformation like we did in the previous proof. Fix a large n (exactly how large will be specified in a moment), and consider the oracle transformation which, if $\xi_A(0^n) \in A$, removes it, and adds it in otherwise. Note that this transformation always the truth value of $0^n \in BIGQUERY^A$, i.e. if $0^n \in BIGQUERY^A$, then $0^n \notin BIGQUERY^{A'}$, and vice versa. Next, it maps $\mathbb{C}^c$ onto itself: For an oracle A in $\mathbb{C}^c$, if $M^A(0^n)$ doesn't query $\xi_A(0^n)$, then it makes no difference whether this was removed or not, and so the transformed oracle A' remains in the class $\mathbb{C}^c$. Furthermore, if B is a random oracle in $\mathbb{C}^c$, and $\xi_A(0^n) \in B$, then the oracle $A = B - \{\xi_A(0^n)$ maps to B, and since $\xi_A(0^n)$ is never queried, removing it makes no difference, so $A \in \mathbb{C}^c$. On the other hand, if $\xi_A(0^n) \notin B$, then the class $A = B \cup \{\xi_A(0^n)\}$ maps to B, and for the same reason is still in $\mathbb{C}^c$. This proves the claim, and from this it follows that, since $A \in \mathbb{C}^c \iff A' \in \mathbb{C}^c$, that $P(A \in \mathbb{C}^c) = P(A' \in \mathbb{C}^c)$. We can see now that for any $A \in \mathbb{C}^n$, the transformation we've defined inverts the value of $0^n \in BIGQUERY$ without changing the machine's answer to $M^A(0^n)$. Since the measure of $\mathbb{C}^c$ goes to 1 for large n, we for large enough n assume that all events are conditioned on this event without altering their probabilities. Thus, given we are conditioning on $\mathbb{C}^c$, and given that $M^A(0^n) = 1$, it is equally likely that 0^n is or isn't in $BIGQUERY^A$, since this probability is preserved under the oracle transformation, and since $M^A(0^n) = 1 \iff M^{A'}(0^n) = 1$. The same is true given that $M^A(0^n) = 0$. Thus the probability that the machine M^A disagrees with $BIGQUERY^A$ on the input 0^n for a randomly drawn A approaches the same probability conditioning on $\mathbb{C}^c$, which is as likely as it isn't, i.e. with probability $\frac{1}{2}$. We thus conclude that the probability approaches $\frac{1}{2}$ for large n (although we only really need it to be greater than 0), completing the proof. □

Definition 3.1. An **α-coloring** is a partition of some set into α categories, or **colors**. We are concerned here with colorings of ω^k, i.e. functions $P : \omega^k \to \{0, 1, 2, ..., \alpha\}$. We say that a set $A \subseteq \omega$ is **homogeneous** for a partition P if there is a particular color i such that, for any collection of k elements chosen from P, the k-tuple consisting of those elements is colored i. I.e. for any $\vec{x} = (x_1, x_2, ..., x_k)$ such that $x_1, ..., x_k \in P$, $P(\vec{x}) = i$.

We are principally concerned with just the subset of increasing functions κ^λ, and thus we distinguish this finer set with the notation $(\kappa)^\lambda$. Another notation we will employ is the Erdos-Rado partition notation. To write $\kappa \to (\rho)^\lambda_\alpha$ is to say that for any α-coloring $P : (\kappa)^\lambda \to \alpha$, there exists a homogeneous set of size ρ. Of particular interest is the case when $\alpha = 2$, and $\rho = \kappa$, i.e. cardinals κ and $\lambda < \kappa$ such that $\kappa^\lambda \to (\kappa)^\kappa_2$, and if the subscript is left out it will be assumed to be 2. If κ has the property that $\kappa \to (\kappa)^\kappa$, then we say that κ has the **strong partition property**. If instead we have that $\kappa \to (\kappa)^\lambda$ for all $\lambda < \kappa$, then we say that κ has the **weak partition property**, and write $\kappa \to (\kappa)^{<\kappa}$.

Theorem 3.6 (Ramsey's Theorem)**.** *ω has the weak partition property. I.e. $\omega \to (\omega)^{<\omega}$, i.e. there exists a homogeneous set for any 2-coloring of $(\omega)^k$, for any $k < \omega$.*

Proof. The proof is by construction. We have a 2-coloring of $(\omega)^k$, and we need an infinite set $A \subseteq \omega$ such that any increasing tuple of k elements in A maps to the same color. To begin with, pick a color, call it i_0. We assume without loss of generality that there are infinitely many tuples which map to this color. (If not, just pick the other color.) Now, pick a number, call it y_0. Now let $A_0 \subseteq \omega$ be the collection of all x such that for any $x_1, ..., x_{k-1} \in A_0$, $(y_0, x_1, ..., x_{k-1}) \mapsto i_0$, i.e. the same color. Assume WLOG that this set A_0 is finite. We can do this since if it is finite, then the same set but with respect to the other color must be infinite, in which case we just pick that one. Now, fix y_1 to be the smallest element of A_0. Again, define $A_1 \subseteq A_0$ to be the collection of all $x_1, ..., x_{k-1}$ such that for any $y_1 < x_1 < x_2 ... < x_{k+1} \in A_1$, $(y_1, x_1, ..., x_{k+1}) \mapsto i_1$. Thus we obtain the (binary) sequence $i_0, i_1, ...$ of colors, as well as the increasing sequence $A = \{y_0, y_1, ...\} \in (\omega)^\omega$. Note that for any y_l, any increasing set of $k-1$ integers $x_1 < x_2 < ... < x_{k-1}$ larger than $y_k \in A$ will have the property that $(y_k, x_1, ..., x_{k-1}) \mapsto i_k$. At least one of these colors has to appear infinitely often for each y_k. WLOG suppose it's i_0. If we then simply consider the subsequence of A which has the above property for this fixed color i_0, we will have finally found our homogeneous set. □

Note that the space $(\omega)^\omega$ is clearly a closed subspace of Baire space, and thus Polish.

Definition 3.2. A set $A \subseteq \omega^\omega$ is **completely Ramsey** if for any $H \in (\omega)^\omega$ and any $s \in \omega^{<\omega}$, there is a