# The Descriptive Complexity and Ramsey Measure of Separating Classes of Oracles

PhD Defense for Alex Creiner, June 2022

# Overview of Results

- Complexity theory seeks to categorize fundamental difficulty classes for computational problems. Some informal examples are the following:
- $\boldsymbol{P}$ is the class of problems which are computable in a reasonable amount of time by an ordinary computing device or human being.
- $\boldsymbol{PSPACE}$ is the class of problems which are computable in a reasonable amount of space by an ordinary computing device or human being. (Think in terms of the amount of 'scratch paper' someone would need.)
- $\boldsymbol{NP}$ is the class of problems which are computable efficiently by machines with the added ability to quickly search over a set of strings of 'reasonable' length.
- $\boldsymbol{BQP}$ is the class of problems which are computable efficiently by a quantum computer.
- $\boldsymbol{PH}$ is the union of a hierarchy of classes similar to the arithmetic or Borel hierarchies, in which $\boldsymbol{P}$ stands in for $\Delta_1$ and $\boldsymbol{NP}$ stands in for $\Sigma_1$.
- Not much is known about the relationships between these classes aside from:

$$\boldsymbol{P \subseteq NP \subseteq PH \subseteq PSPACE}$$

$$\boldsymbol{P \subseteq BQP \subseteq PSPACE}$$

# Overview of Results continued

- Claims about the relationships between these classes tend to come down to whether specific problems are or aren't easy to solve.
- To suppose the existence of an **oracle** is to assume a particular problem (which the oracle magically solves for you in one step) is easy, and to consider the hypothetical reality in which that problem were easy.
- There are oracles relative to which $\boldsymbol{P \neq NP}$, and oracles relative to which $\boldsymbol{P = NP}$. People began to consider the size of sets of oracles relative to which one of these specifically is the case.
- A large body of striking results pertaining to this topic quickly presented themselves. For example, Bennett and Gill were able to show in 1981 that the set of oracles relative to which $\boldsymbol{P \neq NP}$ was Lebesgue measure 1, meaning that if you were to draw a random oracle out of a hat, the two classes would not be equal with probability 1.
- They pitched the *random oracle hypothesis,* the conjecture that if two classes had a certain relationship with probability 1, then that relationship held in our own reality. Within a couple of years this was demonstrated false.

# Overview of Results continued

- Our results consider the size and structure of classes of oracles forcing specific relationships. In particular:

$$\boldsymbol{O} = \{A: \boldsymbol{P^A \neq NP^A}\}$$

$$\boldsymbol{S} = \{A: \boldsymbol{PH^A \neq PSPACE^A}\}$$

$$\boldsymbol{Q} = \{A: \boldsymbol{NP^A \not\subseteq BQP^A}\}$$

- What we've shown is the following:
- The novelty of our research is that it attempts to do a bit of work bridging the gap between computability theory and descriptive set theory.
- In particular, we have that $\boldsymbol{O}, \boldsymbol{Q}$ and $\boldsymbol{S}$, seen from the perspective of descriptive set theory as pointsets, are all $\boldsymbol{\Pi^0_2}$-complete in the Borel hierarchy (and in fact belong in the refined lightface Kleene hierarchy.)
- We've also looked at an atypical measure of largeness (the standard two being category in Baire space and Lebesgue measure).
- We've shown $\boldsymbol{O}$ and $\boldsymbol{Q}$ in particular are 'not small' with respect to the Ramsey forcing measure, i.e. are both Ramsey positive.
- We've also shown that being 'large' with respect to the Ramsey forcing measure is sufficient for $\boldsymbol{P \neq NP}$ and for $\boldsymbol{NP \not\subseteq BQP}$ unrelativized, and that merely being Ramsey positive is sufficient for $\boldsymbol{PH \neq PSPACE}$ unrelativized.
- We thus have a stronger version of the 'random oracle hypothesis', which appears to be true at least in all three of the cases we considered. We have some suggestions for why that is.

# Syntactical Definitions

- A **language** or a **decision problem** is a problem in which the answer is 'yes' or 'no'. More formally, it is a set of strings over some alphabet $\Sigma$, corresponding to the 'yes' answers.
  - Given a fixed enumeration over the relevant alphabet, a language can also be seen as an infinite string of bits, in which the $n^{th}$ bit indicates whether or not the $n^{th}$ string is in the language.
- A **Turing machine** is a triplet $M = (\Sigma, Q, \delta)$, where $\Sigma$ is an alphabet, $Q$ is a finite set of states, and $\delta$ is a transition function $\delta$: $\Sigma \times Q \rightarrow \Sigma \times Q \times \{-1,0,1\}$.
  - At a given moment of computation, the machine is looking at a symbol $\sigma$ while in a state $q$. Based on this **context** it makes a decision according to the transition function. It replaces the symbol with a new one, enters into a new state, and moves it's focus somewhere else (forward a cell, backward a cell, or stays still depending on the third output).
  - The alphabet has a special blank symbol, and there is always at least one special halting state (usually two, one for acceptance and another for rejection), as well as a designated initial state
  - At the start of a computation, the machine 'tape' consists of only a finite set of nonblank symbols.
  - At any moment of the computation, the machine has a **configuration**, consisting of the current context as well as the entire nontrivial length of the tape (i.e. the part consisting of things other than blanks.

# Semantical Definition

- The transition function defines a mapping between configurations. If $c_1$ and $c_2$ are configurations of the Turing machine $M$, and the transition function takes $c_1$ to $c_2$, then we say that $c_1$ **yields $c_2$ in one step**. Inductively this defines what it means for $c_1$ to yield $c_2$ in k many steps for any $k \in \omega$.
- Input and output conventions: Given a string x with no blanks which is to be taken as the input to a machine M, we assume that the machine has an **initial configuration** determined by x.
- Configurations in which the state is a halting state are called **final configurations**. If in k steps, the initial configuration determined by x yields a final configuration, then we say that the **M halts in k steps on the input x**.
- If the halting configuration is in an accepting one, then we say the machine accepts the input x. If it is a rejecting one, we say the machine rejects x.

# Decidability and Time Complexity

- Let L be a language. Suppose M is a Turing machine such that for all strings x, M accepts x in some number of steps if $x \in L$, and M rejects x in some number of steps if $x \notin L$. Then we say that M **decides** the language L, and say that the language L is **decidable**, or **computable**.
- Our central concern however isn't simply problems which are computable, but rather problems which are **efficiently computable**.
- Say that a machine M **operates in time f(n)** if for any string x, the machine halts in a number of steps less than or equal to $f(|x|)$.
- Define the class $\boldsymbol{TIME}\big(f(n)\big)$ to be the collection of all languages which are decidable by Turing machines in time O(f(n)).
- We can now define the more robust time classes

$$\boldsymbol{P} = \bigcup_{k \in \omega} \boldsymbol{TIME}\big(n^k\big)$$

$$\boldsymbol{EXP} = \bigcup_{k \in \omega} \boldsymbol{TIME}\left(2^{n^k}\right)$$

# Space Complexity

- Complementary to the notion of time complexity is space complexity.
- The space used by a Turing machine in a computation is the maximum number of non-blank cells present at any point in the computation, *not* counting the input or output strings!
- What it means for a machine to **operate in space f(n),** and the basic classes $\boldsymbol{SPACE}(f(n))$ are defined identically to their time analogs.
- Two space classes are particularly noteworthy:

$$\boldsymbol{L} = \boldsymbol{SPACE}(log(n)) \qquad \boldsymbol{PSPACE} = \bigcup_{k \in \omega} \boldsymbol{SPACE}(n^k)$$

- Note that for any f(n), $\boldsymbol{TIME}(f(n)) \subseteq \boldsymbol{SPACE}(f(n))$ since a machine can only print as many characters as the number of steps it runs for before halting. Thus we immediately have that

$$\boldsymbol{P} \subseteq \boldsymbol{PSPACE}$$

# Nondeterministic Turing machines

- A **nondeterministic Turing machine** is defined identically to a normal Turing machine, i.e. as a triplet $N = (\Sigma, Q, \Delta)$. The only difference is that the transition function $\Delta$, is no longer required to be a function, but is rather simply a relation.
- This means that a single context can yield multiple actions of the machine, hence the nondeterminism. The machine can make ‘guesses’.
- Given a fixed input string x, the initial configuration given by x can yield one of finitely many different configurations. One can construct a graph in which vertices index configurations and in which edges are defined according to whether one configuration yields another. This is called the **configuration graph** of the machine.
- Say that a nondeterministic machine **accepts** an input x if there exists a *path in the configuration graph* from the initial configuration to an accepting configuration. The number of steps taken by the machine is the length of that accepting path.
- Say a nondeterministic machine **operates in time** f(n) if the maximum length of any path from any initial configuration to a halting configuration is length less than or equal to f(n). Space is defined identically.

# Nondeterministic Turing machines cont.

- Fix a language L. Suppose that a nondeterministic Turing machine N has the property that for any string x, $x \in L$ iff N accepts the input x. Then we say that N **accepts** the language L.
- Define the basic time class $\boldsymbol{NTIME}(f(n))$ to be the set of languages which are accepted by nondeterministic Turing machines operating in time O(f(n)). $\boldsymbol{NSPACE}(f(n))$ is defined similarly.
- Finally, define the class $\boldsymbol{NP} = \bigcup_{k \in \omega} \boldsymbol{NTIME}(n^k)$
- Also noteworthy is the class $\boldsymbol{NL} = \boldsymbol{NSPACE}(\log(n))$
- Nondeterministic Turing machines will be abbreviated NTMs, and regular Turing machines as DTMs.
- Since every regular Turing machine is also a nondeterministic Turing machine, and acceptance is a weaker condition than decidability, we immediately have that $\boldsymbol{TIME}(f(n)) \subseteq \boldsymbol{NTIME}(f(n))$ for and f(n), and therefore $\boldsymbol{P} \subseteq \boldsymbol{NP}$.

# Basic relationships

- Any NTM which accepts a language in time f(n) can be ‘simulated’ by a DTM, albeit in exponentially more time, by exploring the configuration graph.
- Additionally, a machine in space f(n) only needs to operate in time exponential in f(n). This is because there are only exponentially many different configurations which use that much space. If the machine operating in space f(n) isn’t halting in time approximately $2^{f(n)}$, then it has to be ‘looping’, and can be modified into one which does. By formalizing these observations, the following basic hierarchy can be shown.

$$\boldsymbol{L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP}$$

- It can be shown that *at least one* of these containments is proper. That, however, is the extent of what we know. It could be just one of them, or some of them, or all of them.

# Complementary Classes

- For a complexity class $\boldsymbol{C}$, define the complementary class $\boldsymbol{coC}$ to be the collection of all complements of problems in $\boldsymbol{C}$.
- By definition, a DTM computing a decision problem also computes it's dual, since it also gives no answers when the answer is no. Thus $\boldsymbol{P = coP}$, $\boldsymbol{PSPACE = coPSPACE}$, and so forth.
- For NTMs we have a nontrivial asymmetry. Just because there is a machine with short accepting paths for strings in a language doesn't mean that there is a machine with short accepting paths for strings not in a language.
- The question of whether $\boldsymbol{coNP = NP}$ is not clear, and in fact remains an open question.
- Clearly it is the case that $\boldsymbol{P \subseteq coNP}$, and thus $\boldsymbol{P \subseteq coNP \cap NP}$ but whether full equality holds is an open question. Thus $\boldsymbol{P = NP}$ is a stronger conjecture than $\boldsymbol{coNP = NP}$.

# Reductions, hardness, completeness

- Reducibility and completeness are important concept in both complexity theory and descriptive set theory.
- In descriptive set theory, a reduction from one set A to another B is a continuous function $f$ between the spaces which A and B belong to, for which a point $x$ is in the former set iff $f(x)$ is in the latter set.
- Since we can talk about membership in the former set in terms of the latter set up to a continuous function, the former set can be seen as simpler than the latter.
- Complexity theorists are concerned with the same things. One problem $L_1$ can be seen as easier than another $L_2$ in a computational sense if there is a polynomial time computable function f for which a string $x$ is in $L_1$ iff $f(x)$ is in $L_2$.
- In both cases, we have the notions of hardness and completeness. A problem is **hard** for a complexity class if every problem in the class is reducible to by a poly-time computable function. Analogously, a set is hard for a pointclass if every set in the pointclass is reducible to it by a continuous function.
- In both cases, if the hard set/problem belongs to the class itself (call it $\boldsymbol{C}$), then we say that the set/problem is $\boldsymbol{C}$**-complete**.
- Complete problems are to be seen as representative of the hardest problems in a class. They allow for the discussion of classes to be reduced to the discussion of particular pointsets or decision problems

# Oracles

- Any language O can be thought of as an **oracle**.
- A Turing machine with oracle $M^O$ has a special track of tape used to access the oracle. It can write a string on the tape, and magically receive a yes/no answer for whether that string is in O in a single step.
- The set of all Turing machines relativized to the oracle O which halt in polynomial time is denoted $\boldsymbol{P^O}$. All other relativized complexity classes can be defined similarly.
- Just as with reductions and hardness, the concept of relativizing a complexity class to an oracle has a direct analog in descriptive set theory, wherein a pointclass can be relativized to a real number.
- The class we are interested in is the class of oracles separating $\boldsymbol{P}$ from $\boldsymbol{NP}$:

$$\boldsymbol{O} = \{A : \boldsymbol{P}^A \neq \boldsymbol{NP}^A\}$$

- As mentioned, we can see this as a pointset in Cantor space or Baire space. Our first result is in demonstrating that, seen in this way, $\boldsymbol{O}$ is $\boldsymbol{\Pi}^0_2$-complete.

# Demonstrating (Lightface) Membership

- Define the following recursive functions.

$$\mathcal{N}_a^A(i,n,k) = \begin{cases} 1 & \text{if } N_i^A(x_n) \text{ halts in acceptance in time less than } |x_n|^k \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

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

$$\mathcal{M}_a^A(i,n,k) = \begin{cases} 1 & \text{if } M_i^A(x_n) \text{ halts in acceptance in time less than } |x_n|^k \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

$$\mathcal{M}_t^A(i,n,k) = \begin{cases} 1 & \text{if } M_i^A(x_n) \text{ halts in less than } |x_n|^k \text{ steps} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

# Demonstrating membership continued

- Idea: describe a ‘universal’ language for the more powerful class. If, relative to an oracle, that language is in the weaker class, then they must be the same.
- Consider the machine $U^A$ which takes inputs of the form $x, p^n, m$, where $x$ is a string, $p$ is a special symbol never occurring in $x$ or $m$, and $m$ is an integer index for a NTM.
- This machine ‘simulates’ the $m^{th}$ NTM on the input $x$ for $|x| + n$ many steps. Since NTM’s can be simulated by other nondeterminstic machines with linear simulation overhead, the language decided by this machine $L(U^A)$ is in $\boldsymbol{NP^A}$ for any oracle $A$.
- For a fixed enumeration of NTMs, let $u$ be the integer index of the machine $U^A$.

# Demonstrating membership continued

- It can be easily shown that $\boldsymbol{P^A = NP^A}$ iff for some oracle $A$, $L(U^A) \in \boldsymbol{P^A}$.
- This allows us to express that oracle is in $\boldsymbol{O^c}$ as a $\Sigma_2$ statement. In English, we simply need to say that there exists a polynomial time DTM which returns the same output on all inputs. In symbols, this is equivalent to the expression

$$A \in \boldsymbol{O}^c \iff \exists i, k \forall n[(\mathcal{M}_t^A(i,n,k) = 1) \wedge (\mathcal{N}_a^A(u,n,l) = 1 \iff \mathcal{M}_a^A(i,n,k) = 1)]$$

- Where u is the integer index of the NTM $U^A$, and $l$ is the polynomial degree of it's runtime.
- It follows that $\boldsymbol{O^c} \in \Sigma_2^0$, i.e. $\boldsymbol{O} \in \Pi_2^0$.
- This classification also demonstrates that $\boldsymbol{O}$ is Borel, which will be useful observation for later.
- Next we must show that $\boldsymbol{O}$ is $\Pi_2^0$ hard.
- We do this by exploiting Solovay's original construction of an oracle separating $\boldsymbol{P}$ from $\boldsymbol{NP}$. Let's look at that first.

# Solovay's Original Construction

- We want to construct an oracle ensuring that a particular language is always in $\boldsymbol{NP}$ but never in $\boldsymbol{P}$. For any oracle $A$, the language

$$L_A = \{1^n : A \text{ contains a string } x \text{ of length } n\}$$

- Is clearly always in $\boldsymbol{NP^A}$. The idea is to diagonalize across all DTMs halting in polynomial time to ensure none of them can decide the language.
- Fix an effective numbering of all DTMs. (This generally has every machine occurring infinitely often, and this will be essential.) We construct the oracle in stages. At stage n, run the $n^{th}$ DTM relativized to the partially constructed oracle $A_{n-1}$ for $n^{\log(n)}$ many steps on the input $1^n$.
- If the machine halts within this runtime, we want to make sure it gives the wrong answer to the question 'Is $x \in L_A$?'
- If the machine halts in acceptance, then we can simply do nothing and let $A_n = A_{n-1}$.
- If the machine halts in rejection, then we want to add a string, but need to make sure that the string we add does not change the computation of *any* machine already simulated up to that point.

# Solovay's Original Construction Continued

- We commit to only adding strings of length n to the oracle at the $n^{th}$ stage. The maximum number of strings queried by machines by this stage of the construction is

$$\sum_{j=1}^{n} j^{log(j)} < 2^n$$

- This means that there will always be a length n string $x$ which has never been queried by any machine already simulated, which we can add to the oracle without any possibility breaking the construction. Let $A_n = A_{n-1} \cup \{x\}$. This completes the description of the construction.

# Proving hardness

- It's well known that the set $H = \{b \in 2^\omega : b \text{ is infinitely often } 1\}$ is $\Pi_2^0$ hard.
- Thus it suffices to define a continuous function $f: 2^\omega \to 2^\omega$ such that $x \in H$ iff $f(x)$ is an oracle relative to which $\boldsymbol{P \neq NP}$.
- Given a fixed infinite bit string $b$, the task then is to create an oracle $A = f(b)$ which separates $\boldsymbol{P}$ from $\boldsymbol{NP}$ when $b$ has infinitely many ones, and equates them when it does not.
- Our method of doing this is to move through the bit string, performing a single step of the Solovay construction whenever we see a 1.
- Clearly this diagonalization process will only 'complete' when the bit string is in H. However, we also need a way to ensure that the classes are different whenever this doesn't happen.
- It can be shown that if $E$ is an $\boldsymbol{EXP}$-complete language, then $\boldsymbol{P^E = NP^E}$. In fact,

$$\boldsymbol{P^E = NP^E = BQP^E = PH^E = PSPACE^E = EXP}$$

- I.e. a sufficiently powerful language can smother any distinction between all of the complexity classes we are interested in. This will be useful for all of our constructions

# Proving hardness continued

- Fix an $\boldsymbol{EXP}$-complete language E. Given a fixed bit string b, we move through the string looking at each digit:
  - If the $n^{th}$ bit is a 0, we add to the oracle all strings from E which are of length n.
  - If the $n^{th}$ bit is a 1, then we do the next stage of the Solovay diagonalization. By this we mean that if $d(n)$ is the number of 1s encountered by stage n, then we simulate the $d(n)^{th}$ DTM for $n^{\log(n)}$ steps on the input $1^n$, adding a string of length n if the machine halts in rejection, and adding nothing otherwise.
    - The string we add, if we add one, must not be something queried at a previous stage corresponding to a 1. The same counting argument however confirms one will always be available.
    - Occasionally we might need to add strings from E early. If a machine being simulated queries a string x where $|x| > n$ and in which the $|x|^{th}$ bit of b is a 0, then we will add the string ahead of time to ensure consistency.
- If the bit string only has finitely many 1s, then the resulting oracle will be equal to the $\boldsymbol{EXP}$-complete language E up to a finite difference. These finite differences can be hardcoded into a machine and therefore ignored, and so the resulting oracle still forces equality between $\boldsymbol{P}$ and $\boldsymbol{NP}$.
- If the bit string has infinitely many 1s, then we will have completed a diagonalization out of every deterministic machine halting in polynomial time, ensuring inequality.
- The function defined by this construction is clearly continuous (in fact, it is uniformly continuous). Therefore $\boldsymbol{O}$ is $\boldsymbol{\Pi_2^0}$ complete.

# Quantum Turing Machines

- Quantum Turing machines (QTMs) are generalizations of DTMs which, instead of manipulating it's configurations directly, manipulates a probability distribution of possible configurations which might be measured at the end of a computation.
- Configurations of the machine in the ordinary DTM sense are seen basis vectors in a complex Hilbert space. The transition function, rather than taking configurations to other configurations, takes configurations to vectors of complex numbers (called **amplitudes**), whose squared magnitude represents the probability of seeing that configuration if the system is 'measured'. There is one complex number for every action that a normal Turing machine could take.
- At any given moment of computation we have, instead of a single configuration, a linear combination of configurations (called a **superposition**), of which a single one will be measured at the end.
- $\boldsymbol{BQP}$ is the set of problems which are decidable by QTMs in polynomial time.

# Quantum Turing machines continued

- It can be shown that $\boldsymbol{P \subseteq BQP \subset PSPACE}$, but not much is known beyond this. Of particular concern is the relationship between $\boldsymbol{BQP}$ and $\boldsymbol{NP}$. Consider the class

$$\boldsymbol{Q} = \{A: \boldsymbol{NP^A \nsubseteq BQP^A}\}$$

- By enumerating over the QTMs, one can show that $\boldsymbol{Q}$ is $\Pi_2^0$ in an identical way to how we did it for $\boldsymbol{O}$.
- Bennett, Bernstein, Brassard and Vazirani were able in 1997 to construct an oracle relative to which there existed a problem in $\boldsymbol{NP}$ but not in $\boldsymbol{BQP}$, and implying within that construction that the class $\boldsymbol{Q}$ has positive Lebesgue measure.
- A few key lemmas that they used to do this imply that for any particular oracle Turing machine computation, there will generally exist strings which aren't particularly significant to development of the wave function.
- Using this, we were also able to demonstrate hardness in a very similar way that we did it for $\boldsymbol{O}$. Their lemmas allow us to make a counting argument that in a diagonalization there will always be strings available to add to the oracle which don't overly 'perturb' the wave functions of any previous machines simulated.
- It thus follows that $\boldsymbol{Q}$ is $\boldsymbol{\Pi_2^0}$ complete just like $\boldsymbol{O}$.

# The Ellentuck Topology

- Next we turn to evaluating these oracle classes in terms of a different notion of measure from the typical ones (Lebesgue measure and comeagerness in Baire space).
- We can choose to see decision problems as increasing sequences of natural numbers. This set of sequences, which we will denote $[\omega]^{\omega}$, as a subspace of Baire space, is homeomorphic to Baire space itself.
- Our notion of largeness will be comeagerness in a refinement of this space. For a strictly finer topology, comeagerness becomes a stricter notion of largeness than that of the coarser topology.
- Basic open sets in Baire space are specified by finite initial segments.
- Basic open sets in Ellentuck space involve specifying not just a finite initial segment (called the **stem**), but also an infinite set of numbers from which the tail of the sequence has to come from (called the **body**).
- More formally, for a stem $a = \{n_1, n_2, \dots, n_{|a|}\},\ n_1 < n_2 < \cdots < n_{|a|}$, and a body $A$ such that $max(a) < min(A)$, let $[a, A]$ denote the set of all increasing sequences $s$ such that $s(i) = n_i$ for each $i < |a|$, and $s(i) \in A$ for all $i > |a|$.
- This defines the basic open sets of the **Ellentuck topology**.

# Ramsey theory and combinatorics

- Towards finding a way to demonstrate comeagerness, we need to discuss Ramsey theory. The following theorem can be seen as a generalization of the pigeonhole principle.
- *Ramsey's Theorem*: Let $P_0 \cup P_1 \cup \cdots \cup P_k$ be a partition of $\omega^m$. Then there exists a particular set in the partition $P_i$ and an infinite set $H \subseteq \omega$ such that every sequence of $m$ many numbers from $H$ is in $P_i$. That is to say, $[H]^m \subseteq P_i$.
- We call the set $H$ **homogeneous** for the partition (we also say that it is homogeneous for $P_i$ particularly).
- A further generalization would be to replace $\omega^m$ with $[\omega]^\omega$. This generalization is not true without extra conditions.
- *Galvin-Prikrey Theorem (Version 1)*: Let $P_0 \cup P_1 \cup \cdots \cup P_k$ be a partition of $[\omega]^\omega$, *and suppose that each* $P_i$ *is Borel*. Then the partition has a homogeneous set.
- A bit of extra vocabulary allows us to clean up this theorem and express a stronger version of it

# Ramsey and Completely Ramsey Sets

- Say a set $U \subseteq [\omega]^\omega$ is **Ramsey** if for every infinite set of numbers $A \in [\omega]^\omega$, there exists a 'thinned out' set of numbers $H \subseteq A$ which is homogeneous for the partition $U \cup U^c$. That is to say, either $[H]^\omega \subseteq U$ or $[H]^\omega \subseteq U^c$. (Note that the set $U$ here is going to end up being a complexity class, while $A$ will be a particular oracle.)
- Say a set $U \subseteq [\omega]^\omega$ is **completely Ramsey** if for every *basic open Ellentuck set* $[a, A]$ there exists a thinned out body $H \subseteq A$ such that either $[a, H] \subseteq U$ or $[a. H] \subseteq U^c$.
- Note that every completely Ramsey set is Ramsey, since for any infinite set $A$, $[A]^\omega = [\emptyset, A]$
- Completely Ramsey is the refinement of the property of being Ramsey says that even if we 'freeze out' some particular initial segment, we can still always thin out the remaining body to produce a homogeneous set.
- In terms of this vocabulary, we can restate Galvin-Prikrey Version 1 by simply saying: all Borel sets are Ramsey. The full Galvin-Prikrey theorem is stronger:
- *Galvin-Prikrey Theorem (Version 2)*: All Borel sets are *completely* Ramsey.

# Homogeneity and Comeagerness

- The full Galvin-Prikrey theorem and a bit more theory allows us to derive combinatorial conditions for collections being comeager in the Ellentuck topology.
- *Lemma 1:* A set X is nowhere dense in the Ellentuck topology iff for any basic Ellentuck set $[a, A]$, there is a thinned out body $B \subseteq A$ such that $[a, B] \subseteq X^c$.
- *Lemma 2*: Any meager set in the Ellentuck topology is nowhere dense.
- *Corollary*: Suppose there exists a homogeneous set for $U \subseteq [\omega]^\omega$. Then U is nonmeager in the Ellentuck topology. Furthermore, if for any basic Ellentuck set $[a, A]$ there exists a thinned body $B \subseteq A$ such that $[a, B] \subseteq U$, then U is comeager.
- Call a set which is comeager in the Ellentuck topology **Ramsey 1**, and one which is nonmeager **Ramsey positive**.
- We have from the corollary that to demonstrate that a class is Ramsey positive, it suffices to find a homogeneous set for it. To show that it is Ramsey 1, we need to fix an arbitrary basic Ellentuck set $[a, A]$ and show that we can always thin out the body to get a subset of the class.

# The Ramsey measure of $\boldsymbol{O}$

- We will now prove that $\boldsymbol{O}$ is Ramsey positive. In fact, we will actually prove that a slightly smaller collection contained in $\boldsymbol{O}$ is Ramsey positive. Namely

$$\boldsymbol{O'} = \{A : \boldsymbol{NP^A} \neq \boldsymbol{coNP^A}\}$$

- Since $\boldsymbol{P}$ is self dual, we have that $A \in \boldsymbol{O'}$ implies $A \in \boldsymbol{O}$, so $\boldsymbol{O'} \subseteq \boldsymbol{O}$. Showing that $\boldsymbol{O'}$ is large will therefore imply largeness for $\boldsymbol{O}$.
- As discussed, this amounts to proving the existence of an oracle which is homogeneous for $\boldsymbol{O'}$.
- We have from a similar argument to that made for $\boldsymbol{O}$ that this class is an element of $\Pi_2^0$ and thus Borel. Therefore by the Galvin-Prikrey theorem both sets are completely Ramsey.
- This means that any oracle must contain a suboracle which is homogeneous for either $\boldsymbol{O'}$ or it's complement.
- If we can demonstrate the existence of an oracle $A$ which can't possibly contain a homogeneous subset for the complement of $\boldsymbol{O'}$, then by virtue of being completely Ramsey it would follow that there must exist a subset $B \subseteq A$ which is homogeneous for $\boldsymbol{O'}$.
- It suffices then to construct an oracle $A$ such that for all subsets $B \subseteq A$, $B$ is not homogeneous for the complement $\boldsymbol{O'}$, i.e. that there exists a $C \subseteq B$ such that $\boldsymbol{NP^C} \neq \boldsymbol{coNP^C}$.

# The Ramsey Measure for $\boldsymbol{O}$, continued

- The idea is to construct a 'master set' A such that, no matter what the subset of A happens to be, it always contains what strings are needed to perform a Solovay-style diagonalization separating $\boldsymbol{NP}$ from $\boldsymbol{coNP}$.
- This time, we are going to be diagonalizing out of the set of NTMs rather than the set of DTMs.
- This adds complication in the sense that a polynomial length computation involves exponentially many computational paths, which could amount to exponentially many queries, leaving us with no strings to add.
- For this reason and others which will become clear, we avoid potential counting issues by simply running the next machine on an input which is so long that no machine simulated at a previous stage of the construction could have ever queried any strings of that length.
- In particular, we need $f(n)$ to be a fast enough growing function that for all $n$,

$$f(n+1) > f(n)^{\log(f(n))}$$

- That way, no strings we might want to add at a future stage of the construction (i.e. strings of length $f(n)$) will ever have been queried earlier.
- $2^{2^{2^{2^n}}}$ is a function growing fast enough to meet this criteria. $2^{2^{2^n}}$ is not.

# Constructing $A$

- Fix an effective numbering of all NTMs.
- At stage $n$, we run not just the $n^{th}$ machine for $f(n)^{\log(f(n))}$ many steps, but *also all of the previous machines*, on the input $1^{f(n)}$, and we do this *relativized to all subsets of the partially constructed oracle* $A_{n-1}$.
- We want to add a string $x$ at every step, and the string we add shouldn't change the result of any previous computation, for any of the $n$ machines, relativized to any of the $2^n$ possible subsets of $A_{n-1}$.
- None of the previous computations will be disrupted by definition of $f(n)$. The only one we need to worry about is the current batch of $n2^n$ many simulations.
- What we're going to be doing with $A$ is taking arbitrary subsets $B$ and diagonalizing out using only those strings. In this future construction, we will be running at the $m^{th}$ stage the $m^{th}$ machine for $g(n)^{\log(g(n))}$ many steps on $1^{g(n)}$, where $g(n)$ is $f$ evaluated at the $m^{th}$ string in $B$.
- Given the way we are constructing $A$, we want to ensure that there is always a string available to add in the case that the machine accepts relativized to the partial oracle. *Since we don't intend on adding a string when the machine rejects, we don't need to worry about flipping a rejection to acceptance in the construction of* $A$.

# Constructing A continued

- Thus for each of these $n2^n$ many simulations, we only need to worry about the machines with accepting paths, and of these machines we can fix a particular path to focus on not interrupting. As long as at least one path stays accepting, we won't disrupt the computation in a way that disrupts the overall construction.
- Thus in the worst case, each machine accepts relative to each subset $A' \subseteq A_{n-1}$, and queries a string of length $f(n)$ at every step. This amounts to $n2^n f(n)^{\log(f(n))}$ many strings which must be avoided. But clearly this amount is much smaller than $2^{f(n)}$, and so there will always be a string available to add to $A$.
- This defines a 'master set' A with the property that for any infinite $B \subseteq A$, we can always construct an oracle $C \subseteq B$ such that $\boldsymbol{P^C \neq NP^C}$, in the exact way that Solovay did, noting along the way that any strings we might need to add are always available in $B$.
- Upon demonstrating that this set $A$ behaves the way we intended, it follows that $\boldsymbol{O'}$ is Ramsey positive.
- A similar argument can be made for the class $\boldsymbol{Q}$ in order to show that this too is Ramsey positive.

# Sufficiency

- We can now demonstrate that the 'Ramsey measure hypothesis' is true, i.e. that $\boldsymbol{O}$ being Ramsey measure 1 is sufficient for deciding the $\boldsymbol{P \neq NP}$ question.
- Consider an oracle $A$ of the form $A = \{1^{f(n)} : n \in \omega\}$ where $f(n)$ is some increasing function which is 'reasonable' in the sense that, in time comparable to the rate of growth of the function itself, a DTM can print $1^{f(n)}$ on it's tape.
- We claim that if $\boldsymbol{P = NP}$, then $\boldsymbol{P^B = NP^B}$ for any $B \subseteq A$.
- Fix a $B$, and consider a polynomial time relativized NTM $N^B$. We will construct a relativized DTM $M^B$ which decides the same problem.
- On an input $x$, consider a relativized deterministic machine which repeatedly computes $f(n)$ until finding a result longer than the runtime of $N^B(x)$. For all of these, it queries the oracle $B$ on $1^{f(n)}$, constructing a lookup table with the answers to everything that $N^B(x)$ might want to ask $B$ about.
- Given this lookup table, a nonrelativized NTM $N'$ can easily simulate the relativized computation of $N^B$.
- By hypothesis $\boldsymbol{NP = P}$, so there exists a nonrelativized DTM M' which simulates M'.
- By chaining together the relativized DTM constructing the table with the nonrelativized DTM using the table to simulate $N^B$, we obtain an overall relativized DTM which simulates the relativized NTM. Thus $\boldsymbol{P^B = NP^B}$.

# Sufficiency continued

- If the set of oracles separating $\boldsymbol{P}$ from $\boldsymbol{NP}$ were Ramsey 1, then every nontrivial (i.e. infinite) oracle would have to have a homogeneous subset separating the two classes. We've shown that if $\boldsymbol{P} = \boldsymbol{NP}$ then there exist oracles such that this cannot be the case.
- Therefore if $\boldsymbol{O}$ were Ramsey 1, then it must follow that $\boldsymbol{P} \neq \boldsymbol{NP}$.
- Why is this happening? The oracle $A$ is, in a sense, too simple in it's structure to provide information which is significantly useful for computation. We will discuss the matter more when we've talked about the class separating $\boldsymbol{PSPACE}$ from $\boldsymbol{PH}$.

# The Polynomial Hierarchy

- It can be shown that a problem $L$ is in $\boldsymbol{NP}$ iff there exists a polynomial time computable binary relation $R$ such that

$$x \in L \iff \exists^p y R(x, y)$$

- Where $\exists^p$ denotes existential quantification over the set of strings which are polynomial in the length of $x$. (WLOG the polynomial degree can be taken as matching the runtime of $R$.)
- It follows that a problem $L \in \boldsymbol{coNP}$ iff the exact same condition holds, but with a universal quantifier $\forall^p$ in place of the existential one.
- These can be taken therefore as the first layer of an efficiency analog of the classical arithmetic hierarchy.
- Define a **PH-expression** to be a formula of the form

$$\phi(x) = \exists^p y_1 \forall^p y_2 \ldots Q^p y_l R(x, y_1, y_2, \ldots, y_l)$$

- Where $Q^p$ is either an existential or universal quantifier depending on the number of alternations, and where $R$ is a polynomial time computable relation.
- The collection of all languages with membership expressible via a PH-expression defines the class $\boldsymbol{PH}$, which can be seen as the union of all levels of the **polynomial hierarchy**.

# The polynomial hierarchy continued

- By simply considering relations which are polynomial time computable relative to an oracle A, we can easily relativize the class $\boldsymbol{PH}$ using expressions of the form

$$\phi(x) = \exists^p y_1 \forall^p y_2 \dots Q^p y_l R^A(x, y_1, y_2, \dots, y_l)$$

- It can be made clear that $\boldsymbol{PH} \subseteq \boldsymbol{PSPACE}$, but the question of equality remains open. Oracles can be constructed however which separate the classes.
- This construction exploits certain results about Boolean circuits, and in particular an association between PH-expressions and families of constant depth circuits.
- This construction can be utilized to show that the class

$$\boldsymbol{S} = \{A : \boldsymbol{PSPACE}^{\boldsymbol{A}} \neq \boldsymbol{PH}\}$$

  is $\Pi_2^0$-complete, just like $\boldsymbol{O}$ and $\boldsymbol{Q}$.
- However, there is a difference between this class and the other two.

# Ramsey measure and $\boldsymbol{S}$

- We do not have an analogous result for $\boldsymbol{S}$ about Ramsey measure, because it can be shown that *merely being Ramsey positive* is enough to decide the $\boldsymbol{PSPACE}$ vs $\boldsymbol{PH}$ question.
- Suppose that $\boldsymbol{PH} = \boldsymbol{PSPACE}$, and consider an arbitrary oracle $A$. We will show that $\boldsymbol{PSPACE^B} = \boldsymbol{PH^B}$ for all $B \subseteq A$.
  - It follows from this that there cannot exist a homogeneous set for $\boldsymbol{S}$, i.e. $\boldsymbol{S}$ cannot be Ramsey positive.
- Consider specifically a subset of $A$ which only has at most a single string of any given length. (We call an oracle with this property **tame**.) Let $L \in \boldsymbol{PSPACE}^B$. I.e. it is decidable by a DTM operating in time $n^k$ for some $k$.
- We can easily construct a $\boldsymbol{PH}^B$ machine which decides $L$. Fix an input x.
- First, for each length less than $|x|^k$, have an $\boldsymbol{NP}^B$ machine guess a random string of that length and consult the oracle. If it guesses correctly each time, it can construct a lookup table with all relevant information from $B$ which can substitute for B itself in a nonrelativized PSPACE computation.
- Since unrelativized $\boldsymbol{PSPACE} = \boldsymbol{PH}$, this means there is an unrelativied PH-expression which can decide L, given the lookup table. Since NP is always a subset of PH by definition, it follows that we have an implied relativized PH-expression equivalent to L.

# Why did this happen?

- An important difference between the constructions of oracles separating $\boldsymbol{P}$ from $\boldsymbol{NP}$, (or $\boldsymbol{NP}$ from $\boldsymbol{BQP}$) and the construction of oracles separating $\boldsymbol{PSPACE}$ from $\boldsymbol{PH}$ is that the oracles being constructed in the former cases are tame.
- Oracles constructed for the latter separation are *explicitly* not tame. (They involves adding even or odd number of strings of a given length to exploit the nonexistence of small parity circuits.)
- A tame oracle is called **very tame** if the potential candidate strings are efficiently computable.
- It is clear that problems which are very tame are severely limited in their ability to act as valuable oracle information, since, at best, they can only provide information that a DTM could provide itself.
- The same is true for the regular tame oracles as well, but with the letters DTM replaced with NTM, since without an explicit method of computing candidates, it would take an NTM to guess the right strings.
- In this sense it appears harder to separate $\boldsymbol{PSPACE}$ from $\boldsymbol{PH}$ than it is in the other cases. The oracle constructed to aid the computation tends to be difficult for a DTM, whereas in the other two cases it is not.
- In this sense, $\boldsymbol{PSPACE}$ and $\boldsymbol{PH}$ appear closer to being equal than the other two cases.

# Why did this happen? Continued

- There is an important structural difference between the tame oracles and the very tame ones: every non-trivial oracle contains a tame subset, but *not* necessarily a very tame subset.
- This fact is what allows us to confirm that Ramsey positivity is sufficient for $\boldsymbol{PSPACE \neq PH}$.
- The oracle being constructed to do this separation is, from the standpoint of computational difficulty, harder than those constructed in the other two cases. But at the same time, it is also generally easier to construct.
- From an information theory standpoint, it can be understood that *the easier a problem is to construct, the more information can be loaded into it, and the more difficult the problem can potentially be.*
- To have a homogeneous set separating two complexity classes is to effectively have a tame homogeneous set separating two complexity classes, since a non-tame homogeneous set can simply be thinned out to a tame one.
- To say then that a class of oracles is Ramsey positive is to have that a tame oracle is enough to do the separating, i.e. that an oracle providing information which would be obtainable without too much trouble for an NP-machine.
- This is believable in the case of $\boldsymbol{O}$ and $\boldsymbol{Q}$ since it is precisely $\boldsymbol{NP}$ which we are comparing to another class, but in the case of $\boldsymbol{S}$ both classes are at least as powerful as $\boldsymbol{NP}$ from the beginning.

# Conclusions

- From the above discussion we can see that there is a connection between the Ramsey measure of separating classes of oracles and the computational difficulty of oracles which do that separating.
- We can see also that the Ramsey measure of a class oracles separating two other classes says something about the information content of oracles which do the separating.
- This is likely why the ‘Ramsey oracle hypothesis’ appears to work as a strengthening of the random oracle hypothesis.