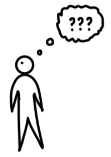
Randomization and Counting Classes

Styopa Zharkov

1 Introduction

In this text, we will explore some surprising results about randomization of complexity classes and the power of being able to count the number of accepting paths of a nondeterministic Turing machine. We assume knowledge of definitions of the standard complexity classes (NP, PH, PSPACE), oracles, and polynomial time many-one reductions. We will redefine the standard randomization complexity classes (BPP, PP), but will not dwell on them. Ultimately, the goal of this text is to understand the following diagram.

$$\begin{tikzcd}
 & & & \mathrm{PP} \arrow[drr] & & & \\
\mathrm{P} \arrow[r] & \mathrm{FewP} \arrow[r] \arrow[drr] & \mathrm{NP} \arrow[r] \arrow[ur] & \mathrm{PH} \arrow[r] & \mathrm{BP}\cdot\oplus\mathrm{P} \arrow[r] & \mathrm{P}^{\#\mathrm{P}} \arrow[r] & \mathrm{PSPACE} \\
 & & & \oplus\mathrm{P} \arrow[ur] & & &
\end{tikzcd}$$



Here, arrows denote inclusion.

Let us recall the two definitions for NP. This will give us some insight on the ideas we will cover.

Definition 1: NP is the set of all languages L such that there exists a poly-time nondeterministic Turing machine that decides L. ◁

Definition 1*: NP is the set of all languages L such that there exists a poly-time deterministic Turing machine V, and polynomial p where $x \in L \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}, V(\langle x, y\rangle) = 1$. In other words, V is the verifier and y is poly-length the certificate. ◁

The first definition implies the second because we can simulate the nondeterministic Turing machine with y as instructions for which guesses to make. The second implies the first because a nondeterministic machine can simply guess y. So, we have two perspectives on this complexity class—a nondeterministic perspective, and a certificate perspective. Similarly, we will have two perspectives on many definitions (although sometimes nondeterminism will be replaced with randomness). Keeping both in mind can help us understand some problems.

For those who aren't acquainted with the randomized classes, let us provide the definitions of the standard randomness classes PP and BPP.

Definition 2: The class PP is the set of all languages L such that there is a constant $\alpha \in [0,1]$ and poly-time probabilistic algorithm that accepts any $x \in L$ with probability more than α and accepts any $x \notin L$ with probability at most α. ◁

Definition 2*: The class PP is the set of all languages L such that there exist a poly-time Turing machine M, a constant $\alpha \in [0,1]$, and polynomial p where

$$x \in L \implies \Pr_{y,|y|=p(|x|)}[M \text{ accepts on } \langle x, y\rangle] > \alpha,$$

$$x \notin L \implies \Pr_{y,|y|=p(|x|)}[M \text{ accepts on } \langle x, y\rangle] \leq \alpha.$$

◁

We can see that the two definitions are equivalent by looking at y as the random choices that the machine in the first definition makes. Note that if we look at the special case of $\alpha = 0$ in definition 2*, we get definition 1* of NP because existence of y means non-zero probability. This means NP $\subseteq$ PP.

Similarly, we can define the class BPP. We only provide the certificate perspective this time, but it shouldn't be hard to convert it to a randomized algorithm definition.

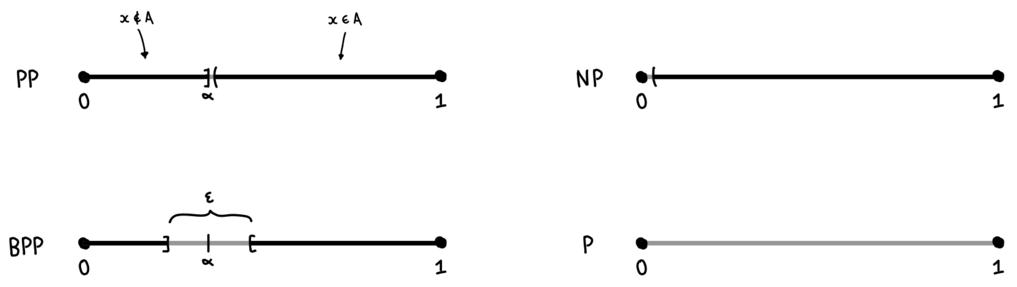
Definition 3: The class BPP is the set of all languages A such that there exist a poly-time Turing machine M, constants $\alpha \in (0, 1)$ and $\varepsilon > 0$, and a polynomial p where

$$x \in A \implies \Pr_{y,|y|=p(|x|)}[M \text{ accepts on } \langle x, y\rangle] \geq \alpha + \frac{\varepsilon}{2},$$

$$x \notin A \implies \Pr_{y,|y|=p(|x|)}[M \text{ accepts on } \langle x, y\rangle] \leq \alpha - \frac{\varepsilon}{2}.$$

◁

Intuitively, BPP is the same thing as PP, but there is a gap between the probabilities of acceptance in the two cases. This might seem like a useless gain, but we suspect that BPP = P, and we have seen that NP $\subseteq$ PP. To help us visualize these classes, we can represent this gap on a diagram.



For each diagram, the left segment is the possible probabilities of acceptance on an input not in A and the right is the possible probabilities for an input in A.

Now that we understand the basics, let's generalize this concept of randomness.

2 Operators on Classes

An operator on a class is essentially a function that transforms a class of languages. In this section, we will introduce some useful operators. We will also formally define what it means to randomize a class. Then, we examine some properties of our randomization operator.

Definition 4: We define the $\exists\cdot$ operator as follows. Let $\mathcal{C}$ be a class of languages. We say $\exists \cdot \mathcal{C}$ is the set of all languages A such that there exists a polynomial p and language $B \in \mathcal{C}$ where

$$x \in A \iff \exists y \text{ such that } |y| = p(|x|) \text{ and } \langle x, y\rangle \in B.$$

◁

Intuitively, the $\exists\cdot$ operator makes a class more nondeterministic. We can similarly define the $\forall\cdot$ operator. Note that, by definition, NP $= \exists \cdot$ P and coNP $= \forall \cdot$ P, and the levels of the polynomial hierarchy are simply P with several alternating $\exists\cdot$ and $\forall\cdot$ operators.

Now, let's define a randomization operator.

Definition 5: Let $\mathcal{C}$ be a class of languages. We define BP $\cdot\, \mathcal{C}$ to be the set of languages A such that there exists a language $B \in \mathcal{C}$, a polynomial p, and constants $\alpha \in (0, 1)$ and $\varepsilon > 0$ where

$$x \in A \implies \Pr_{y,|y|=p(|x|)}[\langle x, y\rangle \in B] \geq \alpha + \frac{\varepsilon}{2},$$

$$x \notin A \implies \Pr_{y,|y|=p(|x|)}[\langle x,y\rangle \in B] \le \alpha - \frac{\varepsilon}{2}.$$

◁

It's easy to see that using P as the class in this definition gives us that $\mathrm{BP}\cdot\mathrm{P} = \mathrm{BPP}$.

A fun activity for complexity theorists is to throw a bunch of these operators on some class and try to figure out what happens. Say, what is $\exists\cdot\forall\cdot\mathrm{BP}\cdot\mathrm{BP}\cdot\exists\cdot\mathrm{BP}\cdot\mathrm{coNP}$? The next few results will help us with some parts of this task. Similar to how we can think of $\exists\cdot$ as 'exists' statements on strings and $\forall\cdot$ as 'for all' statement, we can think of $\mathrm{BP}\cdot$ as a 'for most' statement.

We would like that allowing for randomness doesn't take away from what we already have, so it would be nice if $\mathcal{C} \subseteq \mathrm{BP}\cdot\mathcal{C}$. However, arbitrary classes (sets of sets of strings) can be very strange, so this isn't always true. Let's formulate a condition for which this is true.

Remark: Let $\mathcal{C}$ be a class of languages. If for any language $A \in \mathcal{C}$, we have $\{\langle x,y\rangle : x \in A, y \in \Sigma^*\} \in \mathcal{C}$, then $\mathcal{C} \subseteq \mathrm{BP}\cdot\mathcal{C}$. ◁

In essence, if we are allowed to add unnecessary parts to a language without leaving the class, then randomness doesn't hurt. We can see that for the case of $\mathcal{C} = \mathrm{P}$, this is obvious because any randomized Turing machine can simply ignore its ability for randomness, so $\mathrm{P} \subseteq \mathrm{BPP}$. It's not hard to check that this condition also holds for any other reasonable class that we will see, so from now on, we will assume $\mathcal{C} \subseteq \mathrm{BP}\cdot\mathcal{C}$.

Definition 6: We say that a class $\mathcal{C}$ has probability amplification if for every $A \in \mathrm{BP}\cdot\mathcal{C}$ and any polynomial q, there is a language $B \in \mathcal{C}$ such that

$$x \in A \implies \Pr_{y,|y|=p(|x|)}[\langle x,y\rangle \in B] \ge 1 - 2^{-q(|x|)},$$
$$x \notin A \implies \Pr_{y,|y|=p(|x|)}[\langle x,y\rangle \in B] \le 2^{-q(|x|)}.$$

◁

Intuitively, this means that we can choose the error in the definition of $\mathrm{BP}\cdot$ to be small, and everything still holds. In P, we can run any randomized machine multiple times and choose the outcome that appears more often to reduce the error probability. Similarly, for any reasonable class, it's not hard to show that it has probability amplification. For simplicity, we will assume that any class we see has probability amplification.

Lemma 1 (BP Merging): If $\mathcal{C}$ is a class that has probability amplification, then $\mathrm{BP}\cdot\mathrm{BP}\cdot\mathcal{C} = \mathrm{BP}\cdot\mathcal{C}$.

Essentially, for nice classes, we can merge two $\mathrm{BP}\cdot$ operators into one. This makes sense because if something is true for most of most things, then its usually true for most things.

Proof: We can assume that $\mathrm{BP}\cdot\mathcal{C} \subseteq \mathrm{BP}\cdot\mathrm{BP}\cdot\mathcal{C}$, so we only need to prove the other direction. Let A be a language in $\mathrm{BP}\cdot\mathrm{BP}\cdot\mathcal{C}$. By definition, there exist constants α, ε and a language $B \in \mathrm{BP}\cdot\mathcal{C}$ such that

$$x \in A \implies \Pr_{y,|y|=p(|x|)}[\langle x,y\rangle \in B] \ge \alpha + \frac{\varepsilon}{2},$$
$$x \notin A \implies \Pr_{y,|y|=p(|x|)}[\langle x,y\rangle \in B] \le \alpha - \frac{\varepsilon}{2}.$$

Since $B \in \mathrm{BP}\cdot\mathcal{C}$, and $\mathcal{C}$ has probability amplification, there is a language $D \in \mathcal{C}$ such that

$$\langle x,y\rangle \in B \implies \Pr_{z,|z|=p(|\langle x,y\rangle|)}[\langle x,y,z\rangle \in D] \ge 1 - \frac{\varepsilon}{4},$$
$$\langle x,y\rangle \notin B \implies \Pr_{z,|z|=p(|\langle x,y\rangle|)}[\langle x,y,z\rangle \in D] \le \frac{\varepsilon}{4}.$$

Using the union bound, we can see that

$$x \in A \implies \Pr_{\langle y,z\rangle,|\langle y,z\rangle|=p(|x|+p(|x|))}[\langle x,y,z\rangle \in D] \ge \alpha + \frac{\varepsilon}{2} - \frac{\varepsilon}{4} = \alpha + \frac{\varepsilon}{4},$$
$$x \notin A \implies \Pr_{\langle y,z\rangle,|\langle y,z\rangle|=p(|x|+p(|x|))}[\langle x,y,z\rangle \in D] \le \alpha - \frac{\varepsilon}{2} + \frac{\varepsilon}{4} = \alpha - \frac{\varepsilon}{4}.$$

So, by definition, $A \in \mathrm{BP}\cdot\mathcal{C}$. Thus, we conclude that $\mathrm{BP}\cdot\mathrm{BP}\cdot\mathcal{C} = \mathrm{BP}\cdot\mathcal{C}$. □

Lemma 2 (Swapping Lemma): If $\mathcal{C}$ is a class that has probability amplification, then $\exists\cdot\text{BP}\cdot\mathcal{C} \subseteq \text{BP}\cdot\exists\cdot\mathcal{C}$.

In fact, this is true for any reasonable operator, not just $\exists\cdot$, but we will only need this statement for $\exists\cdot$. You might have seen a special case where $\mathcal{C} = \text{P}$. In that case, this lemma becomes MA $\subseteq$ AM, but the proof is the exact same.

Proof: We will show that any language in $\exists \cdot \text{BP} \cdot \mathcal{C}$ is also a language in $\text{BP} \cdot \exists \cdot \mathcal{C}$.

Let A be a language in $\exists \cdot \text{BP} \cdot \mathcal{C}$. Intuitively this means that there is some $B \in \mathcal{C}$ where A is the set of strings x such that there exists a y such that for most z, we have $\langle x, y, z\rangle \in B$.

More formally, there exists some language B in $\mathcal{C}$, and polynomials p, q such that

$$x \in A \implies \exists y \text{ where } |y| = p(|x|) \text{ and } \Pr_{z,|z|=q(|x|)}[\langle x, y, z\rangle \in B] \geq 1 - 2^{-r(|x|)}$$
$$x \notin A \implies \forall y \text{ where } |y| = p(|x|) \text{ and } \Pr_{z,|z|=q(|x|)}[\langle x, y, z\rangle \in B] \leq 2^{-r(|x|)},$$

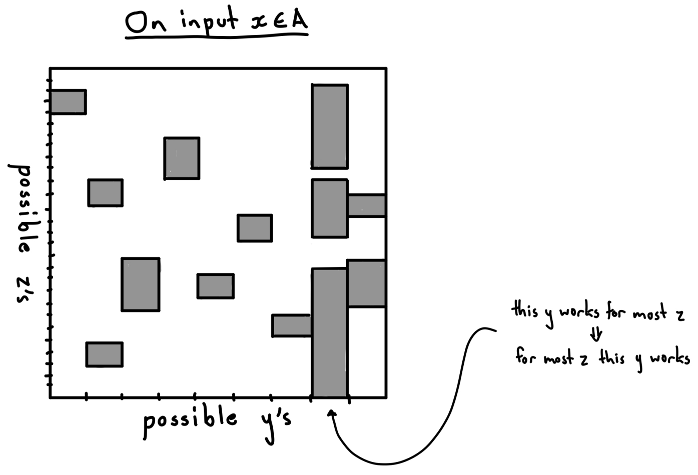
for any polynomial r. We will decide what r should actually be later. Note that we can achieve this small of an error because $\mathcal{C}$ has probability amplification.

Our goal is to show that $A \in \text{BP} \cdot \exists \cdot \mathcal{C}$. Intuitively, this means that there is some language B in $\mathcal{C}$ where A is the set of strings x such that for most z there exists a y such that $\langle x, y, z\rangle \in B$. In other words, we are just trying to show that we can swap the 'exists' and 'for most' here. More formally, we need to show that for some α and ε,

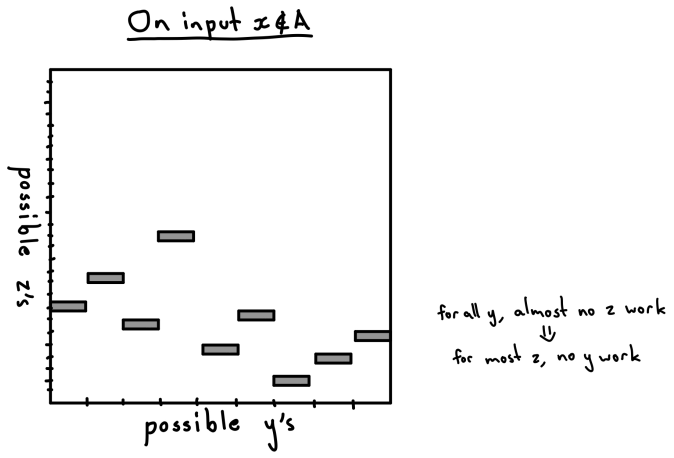
$$x \in A \implies \Pr_{z,|z|=q(|x|)}[\exists y \text{ where } |y| = p(|x|) \text{ and } \langle x, y, z\rangle \in B] \geq \alpha + \frac{\varepsilon}{2}$$
$$x \notin A \implies \Pr_{z,|z|=q(|x|)}[\exists y \text{ where } |y| = p(|x|) \text{ and } \langle x, y, z\rangle \in B] \leq \alpha - \frac{\varepsilon}{2}.$$

We can actually satisfy pretty much any α and ε, so for convenience, let's choose $\alpha = 1/2$ and $\varepsilon = 1/2$. Then, we must show that $x \in A$ implies a probability of at least $3/4$ and $x \notin A$ implies a probability of at most $1/4$ above.

Let's examine both cases. In the case that $x \in A$, there exists a y such that most z put $\langle x, y, z\rangle$ in B, so we can select that y for most z. Thus, the goal follows directly as long as $2^{-r(|x|)} \leq 1/4$. We did not need probability amplification here.



The case where $x \notin A$ is trickier. We must use probability amplification here. First, notice that there are $2^{p(|x|)}$ possible y. We can select $r > p + 2$. Then, using the union bound, we can see that the probability of any z having at least one y such that $\langle x, y, z\rangle \in B$ is at most $2^{-r(|x|)} \cdot 2^{p(|x|)} = 2^{-r(|x|)+p(|x|)} < 2^{-2} = 1/4$. If this is confusing, the following diagram could help. The shaded areas represent combinations of x and y that put $\langle x, y, z\rangle$ in B.



Thus, we have shown both cases, so we can conclude that $A \in \mathrm{BP} \cdot \exists \cdot \mathcal{C}$. We can notice that our choice of α and ε we arbitrary, and this argument actually works for an error as small as 2^{-t} where t is any polynomial.

3 The Power of Parity P

We know that nondeterministic machines accept if there is at least one accepting path, but what if we used a different condition? What if nondeterministic machines were to accept if there is an odd number of accepting paths? To explore this, we introduce the class $\oplus$P (read as 'parity P'). We will see that this class uncovers a surprising property about the BP$\cdot$ operator.

Definition 7: $\oplus$P is the set of all languages A such that there exists a poly-time nondeterministic Turing machine M with the property that $x \in A$ if and only if M has an odd number of accepting computation paths on input x. ◁

Definition 7*: $\oplus$P is the set of all languages A such that there exists a poly-time Turing machine M and polynomial p with the property that $x \in A$ if and only if there is an odd number of y of length $p(|x|)$ such that M accepts on $\langle x, y\rangle$. ◁

For any language $A \in$ P where M is the Turing machine that decides A, we can select $p(|x|) = 0$ in definition 7*, to see that $A \in \oplus$P. So, P $\subseteq \oplus$P. We can also see this by using definition 7 and defining a nondeterministic machine where one transition function simulates M and the other always leads to a reject state.

The next reasonable question to ask about a new complexity class is about its closure under complements.

Lemma 3: $\oplus$P = co$\oplus$P. ($\oplus$P is closed under complement).

Proof: For the proof, we can essentially add an extra dummy accept path to any Turing machine. This turns even numbers of accepting paths into odd ones and odd ones into even ones. Suppose $A \in \oplus$P. Let M and p be the Turing machine and polynomial such that

$$x \in A \iff |\{y \colon M(\langle x, y\rangle) = 1 \text{ and } |y| = p(|x|)\}| \text{ is odd.}$$

Consider $p'(n) = p(n) + 1$ and M' defined as follows:

```
On input $\langle x,y\rangle$,
If $y = 0^{p'(|x|)}$, accept and end.
Otherwise, if $y$ starts with a 0, reject and end.
Let $y'$ be $y$ without the first symbol.
Run $M$ on $\langle x,y\rangle$ and return its result.
```

We can see that the number of accepting y for M' is exactly one more than for M, so this machine shows that $\neg A \in \oplus$P. Since A was arbitrary, $\oplus$P = co$\oplus$P. ☐

Note that in a nondeterministic Turing machine, the number of accepting paths can be up to $2^{p(|x|)}$ where $p(|x|)$ is the runtime on input x. What if we don't want too many accepting paths? We can also define a class where the nondeterministic Turing machines are only allowed a polynomial number of accepting paths.

Definition 8: FewP is the set of all languages A such that there exists a nondeterministic Turing machine M and polynomial p that decides A and also has at most $p(|x|)$ accepting paths on any input x. $\triangleleft$

Definition 8*: FewP is the set of all languages A such that there exists a Turing machine M and polynomials p and q, with the property that if $x \in A$, the number of y such that $|y| = q(|x|)$ and M accepts on $\langle x, y\rangle$ is at least 1 and at most $p(|x|)$. If $x \notin A$, then there are no such y. $\triangleleft$

We do not know the relationship between $\oplus$P and NP (we suspect they are incomparable). However, adding this polynomial restriction allows us to put the class inside $\oplus$P.

Lemma 4: FewP $\subseteq \oplus$P.

Proof: For this proof, we will use a clever combinatorial argument. Let A be any language in FewP. By definition, there exists a Turing machine M and polynomials p, q such that

$$|\{y\colon M(\langle x, y\rangle) = 1 \text{ and } |y| = p(|x|)\}| = \begin{cases} \text{between 1 and } q(|x|) & \text{if } x \in A, \\ 0 & \text{if } x \notin A. \end{cases}$$

Consider the following machine M'.

```
On input $\langle x, z\rangle$ where $z$ is read as $\langle m, y_1, y_2, \ldots, y_m\rangle$,
Check that $1 \le m \le q(|x|)$ and all $y_i$ are strictly increasing. If not, reject.
For each $i \in \{1, \ldots, m\}$,
    If $|y_i| \ne p(|x|)$, reject and end.
    Run $M$ on $\langle x, y_i\rangle$
If all runs accept, then accept. Otherwise, reject.
```

If S is the set of y such that M accepts on $\langle x, y\rangle$, we can see that $\{y_1, \ldots, y_m\}$ can be any nonempty subset of S. Since S is polynomial in $|x|$, all subsets are polynomial in $|x|$. There are $2^{|S|} - 1$ nonempty subsets of S. So, the number of acceptable z is $2^{|S|} - 1$. Notice that this is even if and only if $|S| = 0$. So, M' shows that $A \in \oplus$P. Since A was arbitrary, FewP $\subseteq \oplus$P. $\square$

Interestingly, most examples of languages in $\oplus$P that we know of can be solved in $O(n^{\log n})$ time. We do not know the connection between $\oplus$P and that time complexity, but it makes $\oplus$P seem like it's not a giant class. It has been shown that applying the BP$\cdot$ operator to relatively small classes like P or NP doesn't significantly increase their size. However, at the end of this section we will see that BP $\cdot \oplus$P contains the entire polynomial hierarchy! First, let us prove a series of lemmas and provide some definitions that will lead up to that result.

Lemma 5: The following language is complete in $\oplus$P under poly-time many-one reductions.

$$\oplus\text{SAT} = \{\varphi\colon \varphi \text{ is a Boolean formula with an odd number of satisfying assignments}\}$$

Proof Idea: Examine the proof of completeness of SAT in NP (Cook-Levin theorem). Notice that when we create a formula from a nondeterministic Turing machine, the number of accepting paths is the same as the number of satisfying assignments. So, the same procedure would work to turn a nondeterministic turning machine for $A \in \oplus$P into a $\oplus$SAT formula. $\square$

Definition 9: We say a language A has AND functions if there is a poly-time computable function f such that

$$(x_1 \in A) \wedge (x_2 \in A) \wedge \cdots \wedge (x_k \in A) \iff f(\langle x_1, \ldots, x_k\rangle) \in A.$$

We can similarly define OR functions and NOT functions (where the NOT function will only take one x as an input). $\triangleleft$

Let's look at the language SAT. We can see that given formulas $\varphi_1, \ldots, \varphi_k$, we can write $\varphi_1 \wedge \cdots \wedge \varphi_k$ in polynomial time, so it has AND functions. Similarly we can write $\varphi_1 \vee \cdots \vee \varphi_k$, so SAT also has OR functions. However, it has NOT functions if and only if NP = coNP because a NOT function is essentially poly-time reduction to the complement. What is the case with $\oplus$SAT?

Lemma 6: $\oplus$SAT has AND, OR, and NOT functions.

Proof: Given as input formulas $\varphi_1, \ldots, \varphi_k$, we can output $\varphi_1 \wedge \cdots \wedge \varphi_k$ in polynomial time. We can see that the number of satisfying assignments for the output is the product of the number of satisfying assignments for each input. This is odd if and only if each of the input formulas has an odd number of satisfying assignments. So, the output is in $\oplus$SAT if and only if each of the inputs is in $\oplus$SAT. This shows that $\oplus$SAT has AND functions.

Given as input formula φ with variables $x_1, \ldots, x_n$, we can return

$$\varphi' = (\varphi \wedge y) \vee (\neg y \wedge x_1 \wedge \cdots \wedge x_n).$$

This essentially adds a dummy satisfying assignment to φ. If y is true, we have as many assignments as in φ, and if it's false, we have exactly one. So, $\varphi \in \oplus\text{SAT} \Leftrightarrow \varphi' \notin \oplus\text{SAT}$. This shows $\oplus$SAT has NOT functions.

To show that $\oplus$SAT has OR functions, we can simply use De Morgan's laws on the AND and NOT functions. If f is the AND function and g is the NOT funciton, then $h = g(f(g(\phi_1), \ldots, g(\phi_k)))$ is the OR funciton. $\square$

Before we talk about the polynomial hierarchy, let's show that NP $\in$ BP$\cdot\oplus$P. For that, we need the following lemma.

Lemma 7: SAT is in a sense probabilistically reducible to $\oplus$SAT. More formally, there is a randomized poly-time algorithm M and a polynomial p such that on input formula φ,

$$\varphi \in \text{SAT} \implies \Pr[M(\varphi) \in \oplus\text{SAT}] \geq \frac{1}{p(|\varphi|)}$$
$$\varphi \notin \text{SAT} \implies \Pr[M(\varphi) \in \oplus\text{SAT}] = 0.$$

Proof: For this proof, we will create the algorithm M that turns SAT formulas into $\oplus$SAT formulas. It will work by essentially filtering off random satisfying assignments We will see that there is a decent chance of exactly one assignment being left after the filtering.

If S is a subset of $\{1, \ldots, n\}$, we will use $\oplus S$ to denote the Boolean formula [1] $\bigoplus_{i \in S} x_i$. Essentially, $\oplus S$ asks if an odd number of some subset of variables is set to true. Let h be the OR function for $\oplus$SAT. Consider the following implementation of M.

```
On input formula $\varphi$ with variables $x_1, \ldots, x_n$,
Let $k$ be a random element of $\{0, \ldots, n-1\}$.
For each $i$ in $\{1, \ldots, k+1\}$,
    Let $S_i$ be a random subset of $\{1, \ldots, n\}$.
Output $h((\varphi \wedge \oplus S_1 \wedge \cdots \wedge \oplus S_{k+2}), (\varphi \wedge \neg x_1 \wedge \cdots \wedge \neg x_n))$ and end.
```

Note that if φ has no satisfying assignments, then the output also has no satisfying assignments, so the algorithm works in this case.

The assignment where all variables are set to false (call it the zero assignment) is a special case, so we must consider it separately. If the zero assignment is satisfying for φ, then $\varphi \wedge \neg x_1 \wedge \cdots \wedge \neg x_n$ has exactly one satisfying assignment, so it is in $\oplus$SAT, so the output of M is also in $\oplus$SAT and the algorithm works. So, from now on we will assume the zero assignment is not satisfying for φ. Also note that if the zero assignment is not satisfying, then

$$h((\varphi \wedge \oplus S_1 \wedge \cdots \wedge \oplus S_{k+2}), (\varphi \wedge \neg x_1 \wedge \cdots \wedge \neg x_n)) \in \oplus\text{SAT} \iff (\varphi \wedge \oplus S_1 \wedge \cdots \wedge \oplus S_{k+2}) \in \oplus\text{SAT},$$

so we can ignore the OR function and focus only on its first argument as the output. Intuitively, we considered the zero assignment separately because it is the only one that has no chance of satisfying any of the $\oplus S_i$ formulas.

[1] Verify for yourself that it is possible to create this formula in polynomial time.

Now, let's check the case when $\varphi \in$ SAT and the zero assignment is not satisfying. Let m be the number of satisfying assignments for φ. With probability at least $\frac{1}{n}$, the selected k is such, that $2^k \leq m \leq 2^{k+1}$ because m is at most 2^n, so at least one of the options for k fits this condition[2]. We will show that with this choice of k, the probability that the output has exactly one satisfying assignment is at least $\frac{1}{8}$. This would mean that the probability of the output being in $\oplus$SAT is at least $\frac{1}{8n}$, which is sufficient to complete the proof.

Let us show the $\frac{1}{8}$ bound by examining the probability of any of the satisfying assignments of φ to also be a satisfying assignment of $\varphi \wedge \oplus S_1 \wedge \cdots \wedge \oplus S_{k+2}$.

Let $\mathring{A}$ be a satisfying assignment for φ. Remember that $\mathring{A}$ is not the zero assignment, so there is a variable x_j that is set to true. For any S_i, the probability that $j \in S_i$ is $\frac{1}{2}$, and including a true variable in $\oplus S_i$ changes the parity of the number of true variables in $\oplus S_i$, so the probability that $\mathring{A}$ is a satisfying assignment for $\oplus S_i$ is $\frac{1}{2}$. This is true for all i, and S_i are chosen independently, so the probability that $\mathring{A}$ is a satisfying assignment for $\varphi \wedge \oplus S_1 \wedge \cdots \wedge \oplus S_{k+2}$ is $\frac{1}{2^{k+2}}$. Intuitively, each $\oplus S_i$ filters off about half of the assignments.

Since every two assignments have a variable that is differing, the probability that a different assignment $\mathring{A}'$ for φ is satisfying for $\oplus S_i$ given that $\mathring{A}$ is satisfying for $\oplus S_i$ is still $\frac{1}{2}$ because there is a $\frac{1}{2}$ chance of that variable of being included. So, the probability that $\mathring{A}'$ is satisfying for $\varphi \wedge \oplus S_1 \wedge \cdots \wedge \oplus S_{k+2}$ is also $\frac{1}{2^{k+2}}$ (in other words, this probability is pairwise independent). There are $m-1$ other assignments for φ besides $\mathring{A}$, so the union bound shows that the probability of any other assignment being satisfying for $\varphi \wedge \oplus S_1 \wedge \cdots \wedge \oplus S_{k+2}$ given that $\mathring{A}$ is must be at most $\frac{m-1}{2^{k+2}}$. This means the probability that $\mathring{A}$ is the only assignment for $\varphi \wedge \oplus S_1 \wedge \cdots \wedge \oplus S_{k+2}$ is at least

$$\frac{1}{2^{k+2}}\left(1-\frac{m-1}{2^{k+2}}\right) \geq \frac{1}{2^{k+2}}\left(1-\frac{2^{k+1}}{2^{k+2}}\right) = \frac{1}{2^{k+3}}.$$

Since there can't be 2 such assignments $\mathring{A}$ and there are m choices, the probability that there is exactly one satisfying assignment for $\varphi \wedge \oplus S_1 \wedge \cdots \wedge \oplus S_{k+2}$ is at least

$$\frac{m}{2^{k+3}} \geq \frac{2^k}{2^{k+3}} = \frac{1}{8}.$$

Thus, we can conclude that the probability that the output is in $\oplus$SAT is at least $\frac{1}{8n}$, which is large enough. Our algorithm works for all cases. □

Now, we can use the previous two lemmas and amplify the probabilities to prove that NP is contained within BP $\cdot \oplus$P.

Theorem 1: NP $\subseteq$ BP $\cdot \oplus$P.

Proof: It's easy to see that BP $\cdot \oplus$P is closed under poly-time many-one reductions (just run one algorithm after the other, as in NP). So, if we show that SAT $\in$ BP $\cdot \oplus$P, then we are done.

By our previous lemma, there exists a poly-time probabilistic algorithm M such that

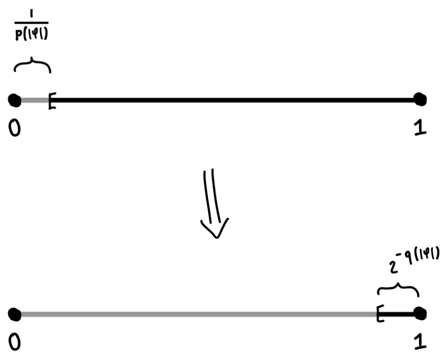
$$\begin{aligned} \varphi \in \text{SAT} &\implies \Pr[M(\varphi) \in \oplus\text{SAT}] \geq \frac{1}{p(|\varphi|)} \\ \varphi \notin \text{SAT} &\implies \Pr[M(\varphi) \in \oplus\text{SAT}] = 0. \end{aligned}$$

If we can show that there is a poly-time probabilistic algorithm M' such that

$$\begin{aligned} \varphi \in \text{SAT} &\implies \Pr[M(\varphi) \in \oplus\text{SAT}] \geq 1 - 2^{-q(|\varphi|)} \\ \varphi \notin \text{SAT} &\implies \Pr[M(\varphi) \in \oplus\text{SAT}] = 0 \end{aligned}$$

for any polynomial $q(|\varphi|)$, then we are done by the definition of BP $\cdot \oplus$P. Intuitively, we are trying to make a very small probability gap into a large one in polynomial time.

[2]Since our randomness is expressed in bits, it's not actually possible to do a uniform distribution unless n is a power of 2. However, it's definitely possible to do a distribution where each k has at least probability of $\frac{1}{2n}$ of being selected, which still works for the proof.



Note that we do not need the error to be as small as $2^{-q(|\varphi|)}$ for the theorem to hold (any constant would suffice), but we will show the stronger statement. Since this is a reduction, we can't simply run M many times, but we can combine many outputs of M into one formula using the OR function h that we know $\oplus$SAT has. Let t be the number of outputs that we combine. We will decide what this should be later. Consider the following implementation for M'.

```
On input formula $\varphi$,
For each $i \in \{1, \ldots, t\}$,
    Let $\varphi_i = M(\varphi)$.
Output $\varphi' = h(\varphi_1, \ldots, \varphi_t)$ and end.
```

If $\varphi \notin$ SAT, then the probability that $\varphi' \in \oplus$SAT is 0 because none of the φ_i are in $\oplus$SAT. If $\varphi \in$ SAT, then the probability that $\varphi' \notin \oplus$SAT is at most

$$\left(1 - \frac{1}{p(|\varphi|)}\right)^t.$$

In the interest of space, we do not include the math behind this bound, but if

$$t \geq 2 \cdot \log\left(2^{q(|\varphi|)}\right) \cdot p(|\varphi|) = 2 \cdot q(|\varphi|) \cdot p(|\varphi|),$$

then

$$\left(1 - \frac{1}{p(|\varphi|)}\right)^t \leq 2^{-q(|\varphi|)}.$$

So, with a polynomial t, we can achieve a small enough probability, showing that M works. We can conclude that this means SAT $\in$ BP $\cdot \oplus$P, so NP $\subseteq$ BP $\cdot \oplus$P. □

Unfortunately, this is not enough to prove the inclusion of the polynomial hierarchy. A stronger statement is needed. However, the proof for this statement follows the proof of NP $\subseteq$ BP $\cdot \oplus$P step for step.

Lemma 8: $\exists \cdot \oplus\text{P} \subseteq \text{BP} \cdot \oplus\text{P}$

Proof: In the proof for NP $\subseteq$ BP $\cdot \oplus$P, we showed a probabilistic reduction from a complete language in NP to $\oplus$SAT. For this proof, we can show a probabilistic reduction from a complete language in $\exists \cdot \oplus$P to $\oplus$SAT. The probability amplification step is the exact same as in theorem 1. Consider the following language. Let $\exists \oplus$SAT be the set of Boolean formulas φ with variables $x_1, \ldots x_{n_1}$ and $y_1, \ldots, y_{n_2}$ such that there is at least one assignment to $x_1, \ldots x_{n_1}$ where there is an odd number of satisfying assignments to $y_1, \ldots y_{n_2}$. We can again apply a variation of the Cook-Levin theorem to see that $\exists \oplus$ SAT is complete in $\exists \cdot \oplus$P.

Let us show a probabilistic reduction similar to the one in lemma 7. Consider the following algorithm M.

```
On input formula $\varphi$ with variables $x_1, \ldots, x_{n_1}$ and $y_1, \ldots y_{n_2}$,
Let $k$ be a random element of $\{0, \ldots, n_1 - 1\}$.
For each $i$ in $\{1, \ldots, k+1\}$,
    Let $S_i$ be a random subset of $\{1, \ldots, n_1\}$.
Output $h((\varphi \wedge \oplus S_1 \wedge \cdots \wedge \oplus S_{k+2}), (\varphi \wedge \neg x_1 \wedge \cdots \wedge \neg x_{n_1}))$ and end.
```

We are doing essentially the same filtering process, but only for the x variables. It's easy to see that if $\varphi \notin \exists \oplus \text{SAT}$, then $M(\varphi) \notin \oplus\text{SAT}$.

In the special case where setting all x variables in φ to false gives an odd number of satisfying assignments for the y variables, we have that $(\varphi \wedge \neg x_1 \wedge \cdots \wedge \neg x_{n_1})$ has an odd number of satisfying assignments, so $M(\varphi) \in \oplus\text{SAT}$.

In other cases, we can show that the probability that $M(\varphi) \in \oplus\text{SAT}$ is at least $\frac{1}{8n_1}$. So, the probabilistic reduction works in all cases. Using the same argument as in theorem 1, we can show that $\exists\oplus\text{SAT} \in \text{BP}\cdot\oplus\text{P}$. So, we conclude that $\exists \cdot \oplus\text{P} \subseteq \text{BP} \cdot \oplus\text{P}$. □

This result is interesting because we usually think of nondeterminism as more powerful than randomness. After all, $\exists\cdot\text{P} = \text{NP}$ is larger than BPP. But in this case of $\oplus$P, we saw that randomness is actually stronger.

Now, we can finally combine several of our results to show that $\text{BP} \cdot \oplus\text{P}$ contains the entire polynomial hierarchy.

Theorem 2 (Toda's theorem): $\text{PH} \subseteq \text{BP} \cdot \oplus\text{P}$.

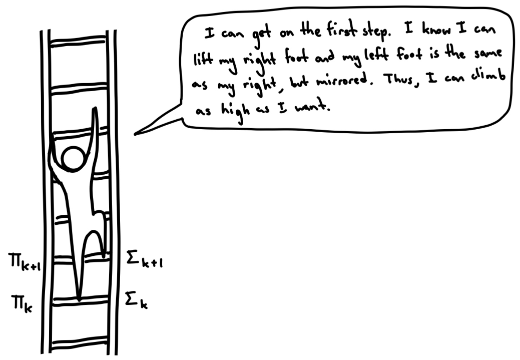
Proof: We will show by induction that for all $k \geq 0$, we have $\Sigma_k^{\text{P}} \cup \Pi_k^{\text{P}} \subseteq \text{BP} \cdot \oplus\text{P}$. We know $\oplus$P is closed under complement, so $\text{BP} \cdot \oplus\text{P}$ is also closed under complement, so showing that $\Sigma_k^{\text{P}} \subseteq \text{BP} \cdot \oplus\text{P}$ would be enough.

The base case of $k = 0$ is trivial because $\text{P} \subseteq \oplus\text{P} \subseteq \text{BP} \cdot \oplus\text{P}$.

Suppose we know that $\Sigma_k^{\text{P}} \cup \Pi_k^{\text{P}} \subseteq \text{BP} \cdot \oplus\text{P}$. Let us show the statement for $k+1$.

Let A be an arbitrary language in $\Sigma_{k+1}^{\text{P}} = \exists\cdot\Pi_k^{\text{P}}$. Since $\Pi_k^{\text{P}} \subseteq \text{BP}\cdot\oplus\text{P}$, we know $A \in \exists\cdot\Pi_k^{\text{P}} \subseteq \exists\cdot\text{BP}\cdot\oplus\text{P}$. Using the swapping lemma, we know $\exists\cdot\text{BP}\cdot\oplus\text{P} \subseteq \text{BP}\cdot\exists\cdot\oplus\text{P}$. Using lemma 8, we have $\text{BP}\cdot\exists\cdot\oplus\text{P} \subseteq \text{BP}\cdot\text{BP}\cdot\oplus\text{P}$. Using BP$\cdot$ merging, we know $\text{BP}\cdot\text{BP}\cdot\oplus\text{P} = \text{BP}\cdot\oplus\text{P}$. Thus, $A \in \text{BP}\cdot\oplus\text{P}$. Since A was arbitrary, we have shown that $\Sigma_{k+1}^{\text{P}} \subseteq \text{BP}\cdot\oplus\text{P}$. Since $\text{BP}\cdot\oplus\text{P}$ is closed under complement, we have $\Pi_{k+1}^{\text{P}} = \text{co}\Sigma_{k+1}^{\text{P}} \subseteq \text{co}(\text{BP}\cdot\oplus\text{P}) = \text{BP}\cdot\oplus\text{P}$. Thus, $\Sigma_{k+1}^{\text{P}} \cup \Pi_{k+1}^{\text{P}} \subseteq \text{BP} \cdot \oplus\text{P}$.

By induction, we conclude that all levels of the polynomial hierarchy are included in $\text{BP} \cdot \oplus\text{P}$, so we have shown that $\text{PH} \subseteq \text{BP} \cdot \oplus\text{P}$. □


I can get on the first step. I know I can lift my right foot and my left foot is the same as my right, but mirrored. Thus, I can climb as high as I want.
Π_{k+1}
Σ_{k+1}
Π_k
Σ_k

4 Counting Oracles

We have looked at $\oplus$P, where we cared about the parity of the number of accepting paths, but what if we knew more than just the parity? What if we knew exactly how many there are? To examine this, we will introduce counting functions and the #P counting class.

Definition 10: Let M be a poly-time Turing machine and p a polynomial. We denote by $\#_M^p$ the function $\{0,1\}^* \to \mathbb{N}$ where $\#_M^p(x)$ is the number of strings y of length $p(|x|)$ such that M accepts on $\langle x, y\rangle$. We call the set of all such functions #P.

Note that this is equivalent to the set of all functions computing the number of accepting paths of a poly-time nondeterministic Turing machine on an input x. We must also point out that #P is not a class of languages, but rather a class of functions, so we must be careful when comparing its power to other complexity functions. However, we can still consider the functions $\#_M^p$ as oracles that return a number instead of a bit. So, the following theorem makes sense.

Theorem 3: $\mathrm{BP} \cdot \oplus\mathrm{P} \subseteq \mathrm{P}^{\#\mathrm{P}}$.

An immediate result from this is that $\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$. Interestingly, this statement doesn't mention $\oplus$P or the BP$\cdot$ operator, but the best proof we know utilizes both.

Proof: Let A be an arbitrary language in $\mathrm{BP} \cdot \oplus\mathrm{P}$. We will show that $A \in \mathrm{P}^{\#\mathrm{P}}$ by creating a machine M' and polynomial p' such that from the value of $\#_{M'}^{p'}(x)$ we will be able to determine if x is in A or not. This proof uses some clever modular arithmetic to alter the $\mathrm{BP} \cdot \oplus\mathrm{P}$ machine for A.

Since $A \in \mathrm{BP} \cdot \oplus\mathrm{P}$, we know there exists some Turing machine M, polynomials p and q, and constants $\alpha \in (0,1)$ and $\varepsilon > 0$ such that

$$x \in A \implies \Pr_{y,|y|=p(|x|)}\left[\begin{array}{c}\text{The number of strings } z \text{ of length } q(|\langle x,y\rangle|)\\ \text{such that } M \text{ accepts on } \langle x,y,z\rangle \text{ is odd.}\end{array}\right] \geq \alpha+\varepsilon$$

$$x \notin A \implies \Pr_{y,|y|=p(|x|)}\left[\begin{array}{c}\text{The number of strings } z \text{ of length } q(|\langle x,y\rangle|)\\ \text{such that } M \text{ accepts on } \langle x,y,z\rangle \text{ is odd.}\end{array}\right] \leq \alpha-\varepsilon$$

We want to determine if x is in A from an oracle call. A simple guess for the approach is to just ask for the number of strings $\langle y,z\rangle$ such that M accepts on $\langle x,y,z\rangle$ (i.e. ask for $\#_M^p(x)$), but that doesn't work. It's not possible to tell how many strings y there are such that an odd number of strings z make M accept on $\langle x,y,z\rangle$ just from the total number of pairs $\langle y,z\rangle$.

We must take a more complicated approach and ask about a different machine. Let $p' = p+1$ (it will only matter that $p' > p$ here). Now, consider the following machine M'. For a cleaner proof, we will use p' to denote $p'(|x|)$.

```
On input $\langle x,y,z\rangle$ where $z$ is read as $\langle z_1^{(1)},\dots,z_{p'}^{(1)}\rangle,\dots,\langle z_1^{(p')},\dots,z_{p'}^{(p')}\rangle$,
If $|y|$ is not $p(|x|)$, then reject.
For each $i \in \{1,\dots,p'\}$,
    If $\langle z_1^{(i)},\dots,z_{p'}^{(i)}\rangle$ is not just a string of $q(|\langle x,y\rangle|)\cdot p'$ zeros,
        For each $j \in \{1,\dots,p'\}$,
            If $|z_j^{(i)}|$ is not $q(|\langle x,y\rangle|)$, then reject.
            Run $M$ on $\langle x,y,z_j^{(i)}\rangle$.
            If rejects, reject.
Accept and end.
```

Let's parse what this algorithm is doing. It essentially simulates M many times. The last certificate z is actually p' packages, each of which has p' certificates. The algorithm makes sure that y and each of the $z_j^{(i)}$ are the same length as the y and z in the machine for A. To be accepted, each package must either be a string of zeros or contain p' certificates that are accepted by M (we assume that no proper encoding is simply a string of zeros).

Now, let's look at the length of accepted strings. In an accepted string, each package has length $q(|\langle x,y\rangle|)\cdot p'$ because it is either that many zeros or contains p' certificates of length $q(|\langle x,y\rangle|)$. So, the total length of any accepted string is $p(|x|) + q(|\langle x,y\rangle|)\cdot p'^2$ because the y must have length $p(|x|)$ and there are p' packages. Note that $|\langle x,y\rangle| = |x| + p(|x|)$. So, the total length of any accepted string is $p(|x|) + q(|x| + p(|x|))\cdot p'^2$, which is a polynomial expressed only in terms of the length of x. To forget about how ugly it is, call this polynomial r.

In the end our strategy will be to ask the oracle for the value of $\#_{M'}^r(x)$, but to understand how to use it, we must examine the value of $\#_{M'}^{r-p}(\langle x, y\rangle)$. This is the number of possible z such that M' accepts on $\langle x, y, z\rangle$ (because y has length $p(|x|)$). Let $m = \#_M^q$ (the number of strings z such that M accepts on $\langle x, y, z\rangle$). We can see that each package must either be a string of zeros or a set of p' valid certificates. There are m options for each certificate, so there are $m^{p'} + 1$ options for each package. There are p' packages, so the value of $\#_{M'}^{r-p}(\langle x, y\rangle)$ is $(m^{p'} + 1)^{p'}$.

Why is this number useful? If m is odd, then $m^{p'} + 1 \equiv 0 \pmod 2$, so $(m^{p'} + 1)^{p'} \equiv 0 \pmod{2^{p'}}$. If m is even, then $m^{p'} \equiv 0 \pmod{2^{p'}}$, so $(m^{p'} + 1)^{p'} \equiv 1 \pmod{2^{p'}}$. Now, remember that we chose p' to be greater than p, so $2^{p'}$ is greater than the number of possible strings y.

Since $\#_{M'}^{r}(x)$ is just the sum of $\#_{M'}^{r-p}(\langle x, y\rangle)$ over all y of length p, we see that $\#_{M'}^{r}(x) \mod 2^{p'}$ is just the number of strings y such that the number of strings z where M accepts on $\langle x, y, z\rangle$ is odd. From this, we can calculate the probability for a random y and deduce if x is in A! If this number is more than $\alpha \cdot 2^p$, then $x \in A$. Otherwise, $x \notin A$.

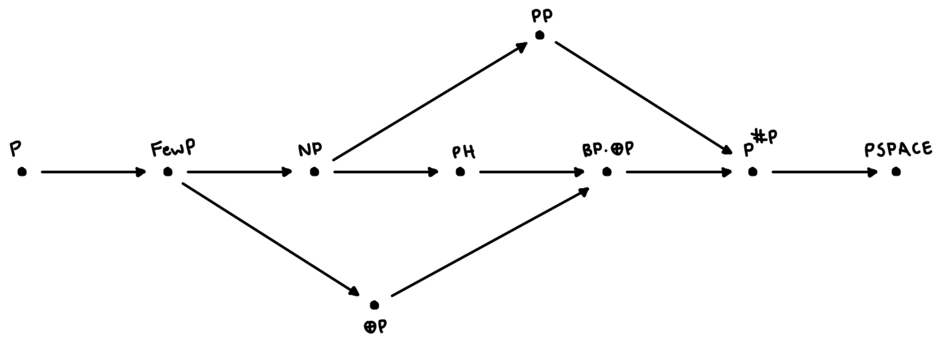
We have shown that one oracle call to $\#_{M'}^{r}(x)$ is enough to tell if x is in A or not. It's also worth noticing that our proof never used ε, so we did not need the probability gap in the definition of the BP$\cdot$ operator. $\square$

Corollary: From theorems 2 and 3, we get that PH $\subseteq$ P$^{\#\text{P}}$.

5 Finishing Notes

It's easy to show the final inclusion P$^{\#\text{P}}$ $\subseteq$ PSPACE because we can simulate any nondeterministic machine in poly-space and simple count the number of accepting paths, so we can simply brute force any function in #P instead of asking the oracle.

We can now return to the diagram and confirm that we have shown every inclusion.


$$\begin{tikzcd}
 & & & & \text{PP} \arrow[ddr] & & \\
 & & & & & & \\
\text{P} \arrow[r] & \text{FewP} \arrow[r] \arrow[ddrr] & \text{NP} \arrow[r] \arrow[uurr] & \text{PH} \arrow[r] & \text{BP}\cdot\oplus\text{P} \arrow[r] & \text{P}^{\#\text{P}} \arrow[r] & \text{PSPACE} \\
 & & & & & & \\
 & & & \oplus\text{P} \arrow[uur] & & &
\end{tikzcd}$$

We have seen the BP$\cdot$ operator, and we have seen the power of being able to count. However, complexity theory has not been solved, so we leave you with some exercises and some unsolved problems.

Exercise 1: For reasonable classes $\mathcal{C}$, show that BP $\cdot\, \mathcal{C} \subseteq \forall \cdot \exists \cdot \mathcal{C}$

Exercise 2: Show that $\oplus\text{P}^{\oplus\text{P}} = \oplus\text{P}$.

Exercise 3: Show that if $\oplus$P $\subseteq$ PH, then the polynomial hierarchy collapses.

Exercise 4: Show that P$^{\#\text{P}}$ = P$^{\text{PP}}$.

Problem 1: Is there a language in $\oplus$P that isn't in TIME($n^{\log n}$)?

Problem 2: Is FewP = $\oplus$P $\cap$ NP $\cap$ coNP?

Problem 3: What can we say about $\exists \cdot \forall \cdot$ BP $\cdot$ BP $\cdot \exists \cdot$ BP $\cdot$ coNP after all?

Sources:

This text roughly follows chapter 19 of Uwe Schöning's *Gems of Theoretical Computer Science.* Some information was taken from chapter 9 of *Computational Complexity: A Modern Approach* by Sanjeev Arora and Boaz Barak. A couple ideas were taken from Yuval Filmus' posts on the Computer Science Stack Exchange.

Algorithm for naming complexity classes

1. Randomly guess $k \in \{0, \dots, n\}$
2. Randomly guess a string of length k.
3. Append P to the end.
4. Output the resulting string.