UCT

Probabilistic Classes

Klaus Sutner
Carnegie Mellon University
Spring 2022

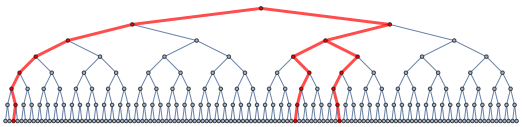
Where Are We? 2

- We have seen a number of examples of probabilistic algorithms, many of them workhorses that are in heavy use.
- In some cases, deterministic "fast" counterparts exist, but are much inferior in practice (order statistics, primality testing).
- In some cases, randomness seems inherently necessary (Monte Carlo simulations, cryptography).
- Reasonably good sources of pseudo-random and truly random bits are available, but PRNGs should not be used blindly.

Probabilistic Classes 3

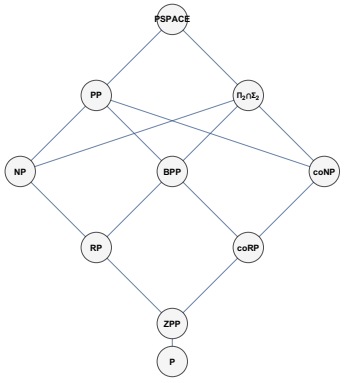
These examples and many more make it tempting to expand our complexity zoo a bit: there should be complexity classes corresponding to probabilistic algorithms.

For anyone familiar with nondeterministic classes like $\mathbb{NP}$ this is not much of a stretch: we already have computation trees



All we need to do is impose additional conditions on the number of accepting branches so we have error bounds.

Lots of Choicse 4



There are lots of different ways one can organize these bounds.

Probabilistic Turing Machines 5

Technically, it is convenient to define a probabilistic Turing machine (PTM) $\mathcal{M}$ to be a Turing machine acceptor with two transition functions δ_0 and δ_1.

At each step in the computation, $\mathcal{M}$ chooses $\delta_{0/1}$ with probability $1/2$, and, of course, independently of all other choices. These are the critical probabilistic properties we need for analysis; they are trivially satisfied if our bits are truly random.

It is preferable to use this specific type of nondeterminism since

- It simplifies the probabilistic analysis significantly.
- It may inflate the size of the corresponding Turing machine by a $\log|Q|$ factor; that seems to be entirely irrelevant.

Projections

As with $\mathbb{NP}$, one can avoid the funky probabilistic Turing machines by invoking witnesses and polynomially bounded projections.

Suppose we have a standard $\mathbb{NP}$ machine $\mathcal{M}$. We may safely assume that

- $\mathcal{M}$ first generates a short sequence $\alpha \in \mathbf{2}^\star$ by nondeterministic guessing.
- $\mathcal{M}$ then runs entirely deterministically, using α to remove nondeterminism from all transitions.

So all the nondeterminism is front-loaded, we first generate a choice sequence and then use it to disambiguate the rest of the computation.

Think about deterministic simulation, it's very similar (enumerate all choice sequences, check each one).

Probability via Projections

In our setting, to check instance x, we need a witness w whose length is polynomially bounded by $n = |x|$. The machine then randomly picks $w \in \mathbf{2}^{p(n)}$ and deterministically verifies some polynomially checkable property of $w\#x$. So we can write

$$\mathsf{acc}_{\mathcal{M}}(x) = \{\, u \in \mathbf{2}^{p(|x|)} \mid \mathcal{M} \text{ accepts } w\#x \,\}$$

If we think of a random experiment, where w is chosen uniformly at random, then the probability of acceptance is

$$\Pr[\mathcal{M}(x) \text{ accepts }] = \frac{\mathsf{acc}_{\mathcal{M}}(x)}{p(|x|)}$$

So the machine performs a few (fictitious) coin tosses and we would like the acceptance probability to be high.

The question is: exactly what conditions should we impose?

Recall: Counting

We have already encountered the idea of counting accepting computations in the class $\#\mathbb{P}$.

There we used accepting computations as a workaround to the problem of defining nondeterministic transducers that compute functions (which are single-valued by definition, the exact opposite of nondeterministic computation).

This leads to a useful and interesting complexity class, so we should expect good results in our probabilistic setting.

Running Time

For the definition of, say, $\mathbb{NP}$ it suffices to have at least one short witness, there is no harm if other computations are longer (and we could eliminate them if we wanted to).

In the probabilistic case, we say that $\mathcal{M}$ runs in time $t(n)$ if it halts in at most $t(|x|)$ steps regardless of the random choices made. Note that this is certainly motivated by examples from ordinary nondeterministic computation: a nondeterministic SAT algorithm first guesses a truth assignment, and then evaluates the formula.

Actually, since we are dealing with probabilities anyway, we might also consider expected running time. We won't go there.

Random Variables

Write $\mathcal{M}(x) \in \mathbf{2}$ for the random variable that indicates acceptance/rejection.

A word of warning: to say that $\mathcal{M}(x)$ is a random variable is rather incomplete information: we really should specify the probability space.

The default is to consider all witnesses $w \in \mathbf{2}^{p(n)}$ under uniform distribution: we flip a fair coin a polynomial number of times.

We could additionally consider distributions on the instances (as in ordinary average case running time), but we will ignore this.

Bounding Error

The critical part of any argument here will be a bound on errors. There are two types of errors that are a priori independent. We would like $\mathcal{M}(x) = L(x)$ for some language $L \subseteq \mathbf{2}^\star$ (conflating L with its characteristic function, as usual). Alas ...

- False Positives
  We may have $x \notin L$ but $\mathcal{M}(x) = 1$.
  Remedy: $x \notin L$ implies $\Pr[\mathcal{M}(x) = 1]$ small.
- False Negatives
  We may have $x \in L$ but $\mathcal{M}(x) = 0$.
  Remedy: $x \in L$ implies $\Pr[\mathcal{M}(x) = 0]$ small.

We get different probabilistic classes by imposing different constraints on false positives/negatives. So we need to figure out what we mean by "small."

BPP

Let $t : \mathbb{N} \to \mathbb{N}$ be a reasonable running time and $L \subseteq \mathbf{2}^\star$ a language.

Definition (Bounded Error Probabilistic Polynomial Time)

A PTM $\mathcal{M}$ decides L in time $t(n)$ if for all $x \in \mathbf{2}^\star$, $\mathcal{M}$ on x halts in at most $t(|x|)$ steps (regardless of random choices) and

$$\Pr[\mathcal{M}(x) = L(x)] \geq 2/3$$

BPP is the class of all languages decided by a PTM in time $O(\mathrm{poly})$.

Do not worry about the magic number $2/3$, we will get rid of it below.

BPP, Again

To be clear: by $\Pr[\mathcal{M}(x) = L(x)] \geq 2/3$ we mean

$$x \in L \Rightarrow \Pr[\mathcal{M}(x) = 1] \geq 2/3$$
$$x \notin L \Rightarrow \Pr[\mathcal{M}(x) = 0] \geq 2/3$$

Again: probability here is defined in terms of the random bits $\boldsymbol{r} \in \mathbf{2}^{p(n)}$ chosen in a particular run. We will always assume $\boldsymbol{r}$ is chosen uniformly at random.

So the machine can make errors in both directions, but the probability of false positives/negatives is at most $1/3$.

The Old Classic

Primality, the problem of determining whether a number is prime, is obviously in co-$\mathbb{NP}$.

V. Pratt showed that Primality is in $\mathbb{NP}$ by constructing short witnesses to primality, a clever use of basic number theory. Alas, his method does not yield a BPP algorithm: guessing prime decompositions is entirely hopeless.

Probabilistic primality testing algorithms in the 1970s showed Primality to be indeed in BPP (and actually in co-RP), but it was not known to be in $\mathbb{P}$ at the time.

Then in 2002 Agrawal, Kayena and Saxena ruined everything by showing that Primality is in fact in $\mathbb{P}$ (using no more than high school arithmetic in the process; the paper is a must-read).

Big Claim

BPP seems to capture the intuitive notion of a problem efficiently solvable by a feasible algorithm very well.

Possibly better than $\mathbb{P}$.

Truth Amplification (aka Error Reduction)

An error probability of $1/3$ or so is, of course, much too high for practical purposes. In the real world, given a probabilistic algorithm $\mathcal{A}$, we would run it repeatedly and tally the answers.

The critical assumption here is that the runs are independent, without independence there is not much one can say.

In some cases this is fairly straightforward: e.g. our probabilistic primality tests have no false negatives, and the error probability for a false positive is at most $1/2$.

So if we run the algorithm r times (independently!) and get no negatives along the way, the error bound is down to 2^{-r}.

General Case

In the general case we count answers:

- Run $\mathcal{A}$ some number r times, independently, and
- take a majority vote.

The repetition factor r should be reasonably small, certainly polynomial (otherwise we might as well construct the whole computation tree of $\mathcal{A}$).

The real question is how large r needs to be, and how much of an error reduction we can expect.

Destroying Error

Consider a poly time PTM $\mathcal{M}$ such that for some constant $c > 0$:

$$\Pr[\mathcal{M}(x) = L(x)] \geq 1/2 + |x|^{-c}$$

Theorem

By taking a majority vote with $O(|x|^{2c+d})$ repetitions, $d > 0$ a constant, we can reduce the error to

$$\Pr[\mathcal{M}'(x) = L(x)] \geq 1 - 2^{-|x|^d}$$

Note that the original machine is rather weak; a much stronger assumption would be $\Pr[\mathcal{M}(x) = L(x)] \geq 1/2 + \varepsilon$. And, the repetition factor r is polynomial, as required.

Recall: Chernoff Bound

Consider the Bernoulli trial $X = X_1 + X_2 + \ldots + X_n$ where the X_i are independent indicator variables with probability p, so $\mu = \mathsf{E}[X] = np$.

Theorem (Chernoff Bound)

Lower tail, $0 < \delta \leq 1$:

$$\Pr[X < (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu}$$

Upper tail, $0 < \delta$:

$$\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

Eyeballing It

The expressions in the bounds are pretty messy, it is often more convenient and still sufficient to replace them by looser estimates such as

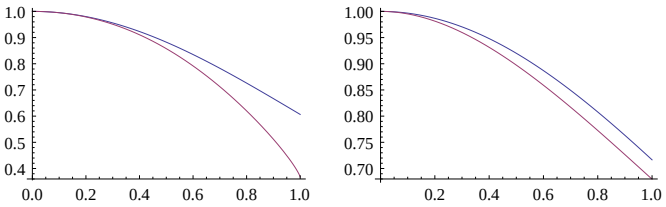
$$\Pr[|X - \mu| \geq c\mu] \leq 2e^{-\min(c^2/4, c/2)\mu}$$

Exercise

Look up a proof of Chernoff's bounds, and some simplified versions. Derive the particular estimate used in the proof below.

Plots

Below $\mu = 1$. For $0 < \delta \leq 1$ the picture shows lower tails on the left, upper tails on the right. Red is the "real" bound, blue a simplified one.



Not too bad.

Back to BPP

We are given a PTM $\mathcal{M}$ such that

$$\Pr[\mathcal{M}(x) = L(x)] \geq 1/2 + |x|^{-c}$$

We claim that, for any constant d, we can design a new PTM $\mathcal{M}'$, performing a majority vote, such that

$$\Pr[\mathcal{M}'(x) = L(x)] \geq 1 - 2^{-|x|^d}$$

The magic repetition number that makes this happen is

$$r = 4|x|^{2c+d}$$

for n sufficiently large.

Error Bound

It is convenient to use the following Chernoff bound, ε small:

$$\Pr[X \geq \mu(1+\varepsilon)] < e^{-\mu\,\varepsilon^2/4}$$

Suppose $x \notin L$, $|x| = n$, and $\Pr[X_i = 1] \leq 1/2 - \delta$. Then the likelihood of error in our majority vote is

$$\begin{aligned}\Pr[X > r/2] &\leq \Pr[X > r(1/2-\delta)(1+2\delta)] \\ &\leq 2^{-r(1/2-\delta)\delta^2} \\ &\leq 2^{-n^d}\end{aligned}$$

The last step follows from $\delta = n^{-c}$ and $r = 4\,n^{2c+d}$ for all n such that $\log n \geq 2/c$. This threshold can be lowered by increasing the constant in r.

□

Closure

Lemma

BPP *is closed under union, intersection and complement.*

Proof.

Closure under complementation follows directly from the definitions.

To show closure under intersection, suppose $\mathcal{M}_i$ decides L_i in BPP, $i = 1, 2$. Build a new machine $\mathcal{M}_\cap$ that does the following:

- Compute $b_i = \mathcal{M}_i(x)$.
- Return $\min(b_1, b_2)$.

The two runs use independent random bits.

Clearly, $\mathcal{M}_\cap$ runs in polynomial time, so we only have to show it obeys our error bounds (potentially after error reduction).

Proof

Case 1: $x \in L_1 \cap L_2$

Then $\Pr[\mathcal{M}_\cap(x) = 1] \geq 2/3 \cdot 2/3 = 4/9$, which can be fixed by amplifying $\mathcal{M}_1$ and $\mathcal{M}_2$ (say, to $7/8$).

Case 2: $x \notin L_1 \cap L_2$

$$\begin{aligned}\Pr[\mathcal{M}_\cap(x) = 0] &= \Pr[\mathcal{M}_1(x) = 0 \vee \mathcal{M}_2(x) = 0] \\ &= \Pr[\mathcal{M}_1(x) = 0] + \Pr[\mathcal{M}_2(x) = 0] - \Pr[\mathcal{M}_1(x) = 0 \wedge \mathcal{M}_2(x) = 0] \\ &\geq 2/3\end{aligned}$$

For the last step, consider the cases $x \notin L_1 \cup L_2$, $x \in L_i - L_{3-i}$ and find the minima.

And $\mathbb{P}$?

It is trivial from the definitions that $\mathbb{P} \subseteq$ BPP.

How about equality? From the error reduction result and the Boolean closure properties, it is entirely conceivable that $\mathbb{P} =$ BPP. Alas, that is an open problem.

As already mentioned, quite a few researchers think that equality holds. This means that we can derandomize all these algorithms without relinquishing a polynomial time bound.

But note that equality per se does not mean much: a linear probabilistic algorithm could go to n^{1000} deterministic. This is very similar to the $\mathbb{P}$ versus $\mathbb{NP}$ situation: the classes might be the same without any impact on practical algorithms.

Feeble Derandomization

Theorem

Suppose $\mathcal{M}$ decides a BPP *language L with witnesses of length $p(n)$. Then for all $n \in \mathbb{N}$ there is a special witness $u_n \in \mathbf{2}^{p(n)}$ such that*

$$\forall\, x \in \mathbf{2}^n \left(x \in L \iff \mathcal{M}(x, u_n) \text{ accepts} \right)$$

Proof.

Use error reduction to get an error bound of 4^{-n}.

Consider the set of bad witnesses for $x \in \mathbf{2}^n$:

$$B_x = \{\, u \in \mathbf{2}^{p(n)} \mid \mathcal{M}(x, u) = 1 - L(x) \,\}$$

Proof Contd

If we pick $u \in \mathbf{2}^{p(n)}$ uniformly at random we have $\Pr[u \in B_x] \leq 4^n$.

Then

$$\Pr[u \in \bigcup B_x] \leq 2^n \cdot 4^{-n} = 2^{-n}$$

But then

$$\Pr[u \in \bigcap \overline{B_x}] \geq 1 - 2^{-n} > 0$$

□

Note that a similar result for $\mathbb{NP}$ seems rather unlikely.

One-Sided Error

A BPP machine can make errors in both directions. Here is a more restricted version in the case where there are no false positives.

$$x \in L \Rightarrow \Pr[\mathcal{M}(x) = 1] \geq 2/3$$
$$x \notin L \Rightarrow \Pr[\mathcal{M}(x) = 0] = 1$$

Definition

RP is the class of all languages decided by a one-sided error PTM in polynomial time.

So RP produces no false positives, but may produce false negatives, with low probability.

It follows immediately that RP $\subseteq \mathbb{NP}$; alas, (provable) closure under complementation vanishes.

Error Reduction

If we run a RP machine r times on a yes-instance, the likelihood for at least one Yes answer is at least

$$1 - (1 - 2/3)^r = 1 - 3^{-r}$$

If we keep running till a Yes appears, we are dealing with a geometric distribution and the expected number of runs is $3/2$. Of course, on a no-instance we would go forever.

More generally, we could change the $2/3$ to any positive constant or even something like n^{-c} without changing the class.

The Other Side

Similarly one defines co-RP, the complements of all languages in RP.

So this means no false negatives, but potentially false positives.

Schwartz-Zippel, Solovay-Strassen and Miller-Rabin are all in co-RP.

We have little information about $\mathbb{NP} \cap$ co-$\mathbb{NP}$, but the analogous question here has a nice answer, see below.

Lemma

ZPP $=$ RP $\cap$ *co*-RP

Closure

Lemma

RP *is closed under union and intersection.*

Proof.

Here is intersection. We have two RP machines $\mathcal{M}_i$ accepting languages L_i.

A machine $\mathcal{M}_\cap$ for $L = L_1 \cap L_2$ simulates both machines and accepts iff both accepts.

Suppose $x \in L$. Then both machines accept with probability at least $(2/3)^2$, and $\mathcal{M}_\cap$ accepts with probability $4/9$, which is enough.

Suppose $x \notin L$, say, $x \notin L_1$. But then $\mathcal{M}_1$ never accepts x, and neither does $\mathcal{M}_\cap$.

□

Zero-Sided Error

Here is a wild idea: how about a PTM that never makes a mistake?

$$x \in L \Rightarrow \mathcal{M}(x) = 1$$
$$x \notin L \Rightarrow \mathcal{M}(x) = 0$$

Here is the glitch: for some computations the running time may not be polynomial; the machine is fast only on average.

ZPP is the class of all such PTM with expected polynomial running time.

These are also called Las Vegas algorithms, as opposed to the more civilized Monte Carlo algorithms.

The randomized version of quicksort can be construed as a Las Vegas type algorithm (though within poly time): with small probability it will have quadratic running time, on average the running time is log-linear.

Clocking a ZPP Algorithm

In reality, we would use a clock to halt the algorithm if it has not returned any (necessarily correct) answer after a polynomial amount of time.

In this case we can think of the output as "don't know."

Lemma

A problem is in ZPP *iff it has an always-correct algorithm that has polynomial average-case running time.*

Of course, in reality we rerun the algorithm whenever we get a "don't know" (meaning the algorithm is out of time).

Some Inclusions

Lemma

$\mathbb{P} \subseteq$ ZPP $=$ RP $\cap$ *co*-RP

RP, *co*-RP $\subseteq$ BPP

But note, as usual, that we may have collapse: it could be that $P =$ BPP, so these classes are all the same.

The difference between this and the $\mathbb{P} = \mathbb{NP}$ question is that a lot of people think that collapse is likely in the probabilistic case but not in the nondeterministic case (not everybody, though).

And $\mathbb{NP}$?

Alas, the relationship between BPP and $\mathbb{NP}$ is currently open.

It is known, though, that

$$\text{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$$

near the bottom of the polynomial time hierarchy.

It follows that $\mathbb{P} = \mathbb{NP}$ implies $\mathbb{P} =$ BPP.

Pushing Things: PP

All our classes BPP, ZPP, RP and co-RP are quite reasonable. The question arises whether more general probabilistic algorithms could also be of interest.

Could we possibly relax our conditions and still get an intuitively plausible randomized algorithm?

How about this: a string is in the language iff it is accepted by a majority of computations:

$$x \in L \Rightarrow \Pr[\mathcal{M}(x) = 1] > 1/2$$
$$x \notin L \Rightarrow \Pr[\mathcal{M}(x) = 1] \leq 1/2$$

Note that this is much weaker than our previous probabilistic classes, we are not specifying a lower bound of the sort $1/2 + \varepsilon$ for some sort of ε.

Robustness

So if $\mathcal{M}$ uses random bitstrings $u \in \mathbf{2}^{p(|x|)}$, x belongs to L iff

$$\frac{\text{acc}_{\mathcal{M}}(x)}{2^{p(|x|)}} > 1/2$$

It turns out that we can replace the $(>, \leq)$ conditions by $(\geq, <)$.

Also, we could change the cutoff to some other value $c > 1/2$.

Exercise

Think about how one might go about proving this.

Properties?

We have

$$\text{RP} \subseteq \text{PP} \subseteq \text{PSPACE}$$

since we can simply count the witnesses leading to acceptance in exponential time and linear space.

Alas, it looks like are overshooting a bit when it comes to capturing feasible computation:

Theorem

$\mathbb{NP} \subseteq$ PP

Note that this is not entirely clear, we have to amplify a potentially single accepting computation to a majority.

Proof

Let $\mathcal{M}$ be an $\mathbb{NP}$ machine accepting L with witnesses of length $p(n)$.

Define a new machine $\mathcal{M}'$ that uses witnesses of length $p(n) + 2$:
$\mathcal{M}'(x, 00u) = \mathcal{M}(x, u)$.
$\mathcal{M}'(x, 1au) = 1$ iff $1au \notin 1^\star$.

Now choose $w \in \mathbf{2}^{p(n)+2}$ uniformly at random. Then since $\mathcal{M}$ is an $\mathbb{NP}$ machine
$x \in L$: Pr[$\mathcal{M}'(x, w)$ accepts] $\geq 3/4 - 2^{-p(n)-2} + 2^{-p(n)-2} = 3/4$.
$x \notin L$: Pr[$\mathcal{M}'(x, w)$ accepts] $= 3/4 - 2^{-p(n)-2} < 3/4$.

By our comments above, this establishes membership in PP.

□

Closure

Lemma

PP *is closed under complement, union and intersection.*

Proof.

These results are tricky, we will only discuss complements here.

Note that it suffices to show that we can symmetrize our definition so that

$$x \in L \Rightarrow \Pr[\mathcal{M}(x) = 1] > 1/2$$
$$x \notin L \Rightarrow \Pr[\mathcal{M}(x) = 1] < 1/2$$

Proof Contd

Let $\mathcal{M}$ be a PP machine recognizing L, and let $p(n)$ be a polynomial bound on the witness length.

For any input $x \in \mathbf{2}^n$, machine $\mathcal{M}$ uses $p(n)$ random bits, so we have a bit of granularity:

$$x \in L \Rightarrow \Pr[\mathcal{M}(x) = 1] \geq 1/2 + 2^{-p(n)}$$

We build a new machine $\mathcal{M}'$ that returns $\mathcal{M}(x) \wedge X$ where X is a random indicator variable such that $\Pr[X = 0] = 2^{-p(n)}$. X is easy to realize in polynomial time. Hence

$$\Pr[\mathcal{M}'(x) = 1] = (1 - 2^{-p(n)}) \Pr[\mathcal{M}(x) = 1]$$

Proof Contd

Hence, if $x \in L$, we have

$$\Pr[\mathcal{M}'(x) = 1] \geq (1 - 2^{-p(n)})(1 + 2^{-p(n)}) = 1 - 2^{-2p(n)} > 1/2$$

On the other hand, if $x \notin L$ we have

$$\Pr[\mathcal{M}'(x) = 1] < \Pr[\mathcal{M}(x) = 1] \leq 1/2$$

Done.

□

Union/intersection is harder.

Semantic vs Syntactic Classes

A syntactic complexity class can be described by purely syntactical means, in particular by an effective enumeration of a class of machines corresponding to the class.

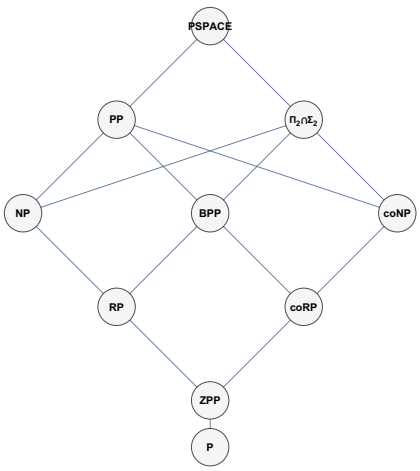
Examples: $\mathbb{P}$, $\mathbb{NP}$, co-$\mathbb{NP}$, PP.

By contrast, a semantic complexity class requires some additional definitional means; in particular filtering out machines according to some undecidable property.

Examples: ZPP, RP, co-RP, BPP.

Syntactic classes tend to have complete problems, semantic ones don't.

Random Classes



Polynomial Time Computable Functions

We have encountered PP once before briefly in the discussion of counting. Here is the result again that ties equality between $\mathbb{P}$ and PP to computing functions.

Theorem

$\mathbb{P} = \text{PP}$ *if, and only if,* $\mathbb{FP} = \#\mathbb{P}$.

Proof.

The har part is $\mathbb{P} = \text{PP}$ implies $\#\mathbb{P} \subseteq \mathbb{FP}$: we have translate the external counting of accepting computations into an actual computation.

Consider $f \in \#\mathbb{P}$, so there is a poly time TM $\mathcal{M}$ such that $f(x) = \#\text{acc}_{\mathcal{M}}(x)$, the number of witnesses $w \in \mathbf{2}^{p(|x|)}$ such that $\mathcal{M}(w\#x)$ accepts, $p(n)$ some polynomial.

Proof, Contd

$\#\mathbb{P}$ is closed under addition: for two TMs $\mathcal{N}_0$ and $\mathcal{N}_1$ define a new machine $\mathcal{N} = \mathcal{N}_0 \oplus \mathcal{N}_1$

$$\mathcal{N}(aw\#x) = \mathcal{N}_a(w\#x)$$

Hence $\#\mathsf{acc}_{\mathcal{N}}(x) = \#\mathsf{acc}_{\mathcal{N}_0}(x) + \#\mathsf{acc}_{\mathcal{N}_1}(x)$. Note the $\mathcal{N}$ requires one more witness bit.

It is trivial to construct a machine C_β such that $\#\mathsf{acc}_{C_\beta}(x) = \beta$.

Now assume $\mathbb{P} = \mathsf{PP}$. But then we can check in polynomial time whether

$$\mathsf{Pr}[(C_\beta \oplus \mathcal{M})(x) = 1] = \left(\beta + \#\mathsf{acc}_{\mathcal{M}}(x)\right) 2^{-p(|x|)-1} > 2^{-p(|x|)}$$

So we can compute $\#\mathsf{acc}_{\mathcal{M}}(x)$ by binary search: find the least $\beta \leq 2^{p(|x|)}$ so that the last inequality holds. This can be handled in polynomial time overall.

□