

# UCT

## Probabilistic Classes

KLAUS SUTNER

CARNEGIE MELLON UNIVERSITY

SPRING 2024



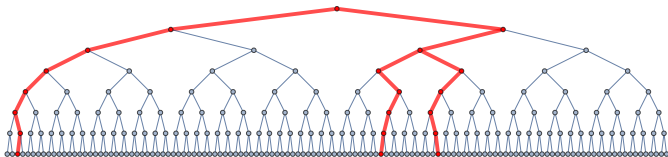
**1 Randomized Polynomial Time**

**2 ZPP, BPP, RP, PP**

- 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.

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  $\text{NP}$  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.

Technically, it is convenient to define a **probabilistic Turing machine (PTM)**  $\mathcal{M}$  to be a Turing machine acceptor with two transition functions  $\delta_0$  and  $\delta_1$ .

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  $O(\log |Q|)$  factor; that seems to be entirely irrelevant.

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

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

- $\mathcal{M}$  first generates a short sequence  $\alpha \in \mathbf{2}^*$  by nondeterministic guessing.
- $\mathcal{M}$  then runs entirely deterministically, using  $\alpha$  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).

In our setting, to check instance  $x$ , we need a witness whose length is polynomially bounded by  $n = |x|$ . We have a deterministic machine  $\mathcal{M}$  that requires a potential witness  $\mathbf{r} \in \mathbf{2}^{p(n)}$  as additional input and verifies some polynomially checkable property of  $\mathbf{r}$  and  $x$ . So we can write

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

We can think of the machine as performing a few (fictitious) coin tosses and can talk about the probability of success:

$$\Pr[\mathcal{M}(x) \text{ accepts}] = \frac{|\text{acc}_{\mathcal{M}}(x)|}{2^{p(|x|)}}$$

We need to make sure that the acceptance probability is high.

This should really be written more carefully as

$$\Pr_{\mathbf{r} \in \mathbf{2}^{p(|x|)}} [\mathcal{M}(x, \mathbf{r}) = 1]$$

with the understanding that the random bits  $\mathbf{r}$  are chosen uniformly at random. So this is a finite sample space, and we are really dealing with counting arguments, dressed up in a probabilistic manner.

Usually, one does not bother with this level of detail.



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.

For the definition of, say,  $\text{NP}$  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 mostly won't go there.

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

Again, to say that  $\mathcal{M}(x)$  is a random variable is incomplete information: we really should specify the probability space. By default, we use the uniform distribution: we flip a fair coin a polynomial number of times.

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

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  $L \subseteq \mathbf{2}^*$  (conflating  $L$  with its characteristic function, as usual). Alas . . .

- False Positives

We may have  $x \notin L$  but  $\mathcal{M}(x) = 1$ .

- False Negatives

We may have  $x \in L$  but  $\mathcal{M}(x) = 0$ .

The obvious remedy is to insist that these errors occur with small probability.

So we need to figure out what precisely we mean by “small” to obtain useful probabilistic classes.

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

### Definition (Bounded Error Probabilistic Polynomial Time)

A PTM  $\mathcal{M}$  **decides**  $L$  in time  $t(n)$  if for all  $x \in \mathbf{2}^*$ ,  $\mathcal{M}$  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  $O(\text{poly})$ .

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

Our assumption  $\Pr[\mathcal{M}(x) = L(x)] \geq 2/3$  really comes in two parts:

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

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

We will later discuss asymmetric cases where the two bounds do not agree.

The whole analysis of the algorithm is based on these probabilities and its internal logic.

It's only for actual execution of the algorithm that we need to worry about a source of sufficiently random bits.

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

V. Pratt showed that Primality is in  $\text{NP}$  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  $\text{co-RP}$ ), 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).

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

Probably (no pun intended) better than  $\mathbb{P}$ .



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  $\rho$  times (independently!) and get no negatives along the way, the error bound is down to  $2^{-\rho}$ .

In the general case we count answers:

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

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

Naturally there will be a trade-off between  $\rho$  and the error bound, we'd like to find a sweet spot.

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

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

For  $c = 1$  this would mean

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

which seems a like a rather weak assumption. Intuitively, something like  $\Pr[\mathcal{M}(x) = L(x)] \geq 1/2 + \epsilon$  for a constant  $\epsilon$  would seem to make more sense.

Incidentally, in the RealWorld<sup>TM</sup>, this would be the same.

## Theorem

Let  $\mathcal{M}$  be a PTM such that  $\Pr[\mathcal{M}(x) = L(x)] \geq 1/2 + |x|^{-c}$ .

By taking a majority vote with  $O(|x|^{2c+d})$  repetitions,  $d > 0$  a constant, we improve the performance guarantee

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

So the error is now exponentially small and the repetition factor is polynomial, as required.

For example, for  $c = d = 1$  we get an error of  $2^{-n}$  with a cubic number of repetitions.

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

### 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,  $0 < \delta$ :*

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

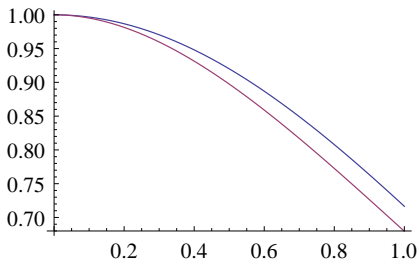
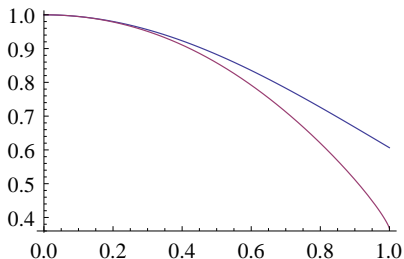
The expressions in the bounds are pretty messy, it is often more convenient and still sufficient to replace them by looser estimates. Chernoff-Hoeffding bounds are popular for example. Here is one, valid for  $0 < \varepsilon \leq 1$ :

$$\Pr[X \geq (1 + \varepsilon)\mu] \leq \exp(-\mu \varepsilon^2 / 3)$$

We will use this in the weaker form

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

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.

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

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

for  $|x|$  sufficiently large.



Here is one direction.

Suppose  $x \notin L$ ,  $|x| = n$ , and  $\Pr[X_i = 1] \leq 1/2 - \delta$  where  $\delta = n^{-c}$ . Since  $1/2 > (1/2 - \delta)(1 + 2\delta)$ , the likelihood of error in our majority vote is

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

The last step follows from  $\rho = 4n^{2c+d}$  provided that  $\log n \geq 2/c$ . □

So even for  $d = 1$  we have exponentially small error.

## 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.

**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  $5/6$ ).

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

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

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.

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

How about equality? From the error reduction result and the Boolean closure properties, it is entirely conceivable that  $\mathbb{P} = \text{BPP}$ . 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  $\text{NP}$  situation: the classes might be the same without any impact on practical algorithms.

## 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  $\mathbf{r}_n \in \mathbf{2}^{p(n)}$  such that

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

*Proof.*

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

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

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

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

Then

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

But then

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

□

Yup, that's totally non-constructive; we don't really know what the universal witness is.

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

1 Randomized Polynomial Time

2 ZPP, BPP, RP, PP

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  $\text{RP} \subseteq \text{NP}$ ; alas, (provable) closure under complementation vanishes.



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

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

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.

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  $\text{NP} \cap \text{co-NP}$ , but the analogous question here has a nice answer, see below.

Lemma

$$\text{ZPP} = \text{RP} \cap \text{co-RP}$$

## 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  $(2/3)^2$ , but we can amplify to get  $\mathcal{M}_\cap$  to accept with probability at least  $2/3$ .

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

□

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.

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).

## Lemma

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

$$\text{RP}, \text{co-RP} \subseteq \text{BPP}$$

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

The difference between this and the  $\mathbb{P} = \text{NP}$  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).

Alas, the relationship between BPP and 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  $\mathbb{P} = \text{NP}$  implies  $\mathbb{P} = \text{BPP}$ .

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  $1/2 + \epsilon$  for some sort of  $\epsilon$ .



So if  $\mathcal{M}$  uses random bitstrings of length  $p(|x|)$ , then  $x$  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.*

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 PP is overshooting a bit when it comes to capturing feasible computation:

Theorem

$$\text{NP} \subseteq \text{PP}$$

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

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

Define a new machine  $\mathcal{M}'$  that uses witnesses of length  $p(n) + 2$ :

$$\mathcal{M}'(x, 00r) = \mathcal{M}(x, r).$$

$$\mathcal{M}'(x, abr) = 1 \text{ for } ab \neq 00 \text{ unless } abr \in 1^*.$$

Now choose  $w \in \{0,1\}^{p(n)+2}$  uniformly at random. Then since  $\mathcal{M}$  is an NP machine

$$x \in L: \Pr[\mathcal{M}'(x, w) \text{ accepts}] \geq 3/4$$

$$x \notin L: \Pr[\mathcal{M}'(x, w) \text{ accepts}] < 3/4.$$

By our comments above, this establishes membership in PP.

□

## 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

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

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

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

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

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

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

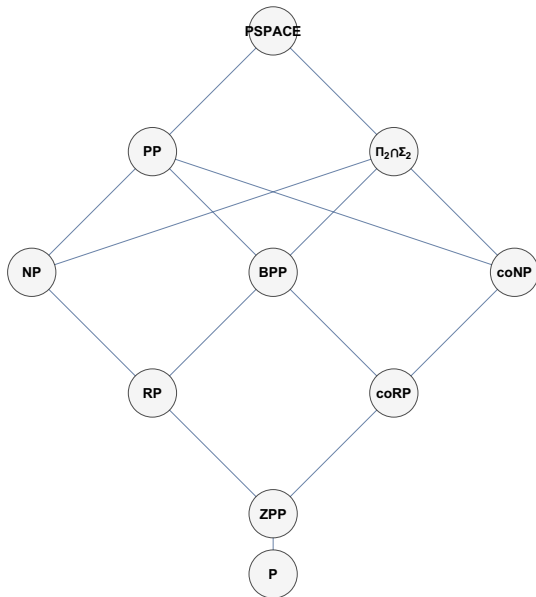
On the other hand, if  $x \notin L$  we have

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

Done.

□

Union/intersection is harder.



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

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

Examples:  $\text{NP} \cap \text{co-NP}$ ,  $\text{ZPP}$ ,  $\text{RP}$ ,  $\text{co-RP}$ ,  $\text{BPP}$ .

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



Both  $\text{NP}$  and  $\text{co-NP}$  are syntactic classes: for example, we can provide an enumeration  $M_e$  of the corresponding machines (more precisely, a nondeterministic Turing machine together with a polynomial time bound).

But for  $\text{NP} \cap \text{co-NP}$  we would need 2 machines, one for  $L$  and one for  $\bar{L}$ .

Alas, it is undecidable whether whether 2 machines have appropriate acceptance languages.

How could one systematically get rid of random bits?

One nice idea is to think about PRNGs that start with a short truly random string  $X$ .

Then they compute in an entirely deterministic manner a sequence of bits  $b_i$ ,  $i \in [n]$ , that are random enough to fool all probabilistic machines in a certain class.

Write  $\ell(n)$  for the length of the initial seed  $X$ .

If  $\ell(n)$  is small enough, we could simply try out all possible strings of length  $\ell(n)$  and do a majority count.

How could we go about squeezing  $n$  many well-behaved bits out of  $\ell = \ell(n)$  many?

Here is a sketch: suppose we had a function  $f : \mathbf{2}^k \rightarrow \mathbf{2}$  and subsets  $S_i \subseteq [\ell]$  of cardinality  $k$ .

The we could use

$$b_i = f_i(X_{j_1}, X_{j_2}, \dots, X_{j_k})$$

where  $S_i = \{j_1, j_2, \dots, j_k\}$  (in increasing order).

Since there are potentially exponentially many such sets  $S_i$ , this could produce a long output sequence.

Needless to say,  $f$  is also critical here. It needs to be difficult enough to compute.

When we replace true random bits by pseudorandom bits we should expect the error probability to increase.

So what we need is this:

$p$  = error probability with true random bits

$q$  = error probability with generated pseudorandom bits

Then  $|p - q|$  needs to be small. The technical details are messy, we won't go there.

One can prove that the existence of “nice” Nisan-Wigderson generators implies  $\mathbb{P} = \text{BPP}$ .

So far, we have considered real algorithms augmented by random bits. In terms of a formal model, this leads to probabilistic Turing machines.

What would happen if we added randomness to just plain old finite state machines? Ordinary finite state machines (deterministic or nondeterministic) only recognize regular languages, languages that are linear time and constant space.

## Theorem

*A probabilistic finite state machine can recognize  $\{ a^n b^n \mid n \geq 1 \}$ .*

For simplicity, assume we have a two-way machine.

- Check that the input is of the form  $a^n b^m$ .
- Check that  $n = m \pmod{1000}$ .
- Flip a fair coin for each  $a$  and  $b$ .  
Event  $A$ : all  $a$ -coins come up heads.  $B$  analogous.
- If  $A \wedge B$  or  $\neg A \wedge \neg B$  do nothing.
- If  $A \wedge \neg B$ : increment a counter  $c_a$ .
- If  $\neg A \wedge B$ : increment a counter  $c_b$ .
- If one counter (say,  $c_a$ ) reaches 100:  
check if  $c_b > 0$ . If so, accept; else reject.

This result is surprising, but not earth-shattering, the language is just slightly more complicated than regular. But the next theorem is utterly insane.

## Theorem

*A probabilistic finite state machine can recognize any semidecidable set.*

This can be handled by applying a similar technique to simulate 2-counter machines (finite state machines that have 2 unbounded counters, but no other memory), one of the most primitive systems known to be computationally universal by a famous theorem of Marvin Minsky.