Chapter 7

Interactive Proof Systems

In Chapter 1, we defined a prover-verifier model for languages in NP. Let us take SAT as an example language in NP. Suppose the prover wants to prove that a given formula is satisfiable. He can do so by giving a satisfying truth assignment as a certificate. The verify can verify in polynomial time whether the assignmnet indeed satisifies the formula. Suppose the formula is actually unsasfiable. How can the prover provide a concise certificate that the formula is unsasfiable? In other words, is there a proof system for unsatisfiability? Next two chapters develop proof systems for languages beyond NP. In these proof systems, the prover and verifier have many rounds of interaction. We start with an example for an Interactive Proof system (**IP**).

7.1 Interactive Proofs – An Example

We explain IP with an example. We say that two graphs $G = (V, E)$ and $H = (V', E')$ are isomorphic, if relabeling G would give H. Formally, G and H are isomorphic, if there is a one-one and onto function $\sigma : V \to V'$, such that $(u, v) \in E$ iff $(\sigma(u), \sigma(v)) \in E'$. If G and H are isomorphic, we write it as $G \cong H$. For a permutation σ on V, we denote $\sigma(G) = (V, E_\sigma)$, where, $E_\sigma = \{(\sigma(u), \sigma(v))|(u, v) \in E\}$. Define the following two languages:-

$$GI = \{(G, H)|G \cong H\}$$

$$GNI = \{(G, H)|G \ncong H\}$$

It's easy to see that GI is in NP. We can simply guess a σ and verify whether $\sigma(G) = H$. What about GNI? It's in co-NP, by definition. But what about a proof system for GNI? No concise proof for GNI is known. Here, we describe what's called an interactive proof for GNI.

An Interactive Proof system (IP) for L consists of two players, prover and the verifier. The prover claims that a string x is in L and the verifier has to verify it. The prover

has no computational limitations, he is all powerful, whereas, the verifier is a probabilistic polynomial time player. They can interact with each other. The verifier can toss some coins and ask the prover some question and prover answers it. This process of interaction can go on for some polynomial number of rounds. At the end, the verifier accepts or rejects the input. If $x \in L$, then we require that the verifier accept with probability $\geq 3/4$ and if $x \notin L$, then he accepts with probability $\leq 1/4$. As usual, the values $3/4$ and $1/4$ are arbitrary and can be amplified.

We now describe an IP for GNI. So, let G and H be the two input graphs and the prover claims that $G \ncong H$. The verifier tosses a coin. If HEAD, he chooses G else chooses H. Let the chosen graph be S. Then, he chooses a random permutation σ and sends $S_\sigma = \sigma(S)$ to the prover. The prover has to send the original coin toss of the verifier. If G and H were really non-isomorphic, then, S_σ will be isomorphic to exactly one of G or H and the prover can find it easily and correctly guess the coin toss. On the other hand, if he were cheating and $G \cong H$, S_σ be isomorphic to both G and H. So the prover, can at most make a wild guess at what the coin toss was. He will have probability only $1/2$ to succeed. We can run this process a polynomial number of times to make the probability of cheating to be exponentially small.

It's easy to see that $NP \subseteq IP$. The verifier keeps quiet! On input x, the prover sends him the certificate that $x \in L$. The verifier verifies whether it's a valid certificate.

7.2 Arthur-Merlin Games

Arthur-Merlin games are restricted versions of IP. In AM we allow only one round of interaction betwen the prover and verifier. Here we call the prover as Merlin and the verfier as Arthur. Merlin is an all powerful magician, whereas, Arthur is a humble probabilistic polynomial time monarch! Merlin wants to prove that a string x is in L. As in IP, he does this by interacting with Arthur.

Definition (AM): $L \in AM$ iff $\exists$ a polynomial time predicate P such that

$$x \in L \Rightarrow Pr_y[\exists z, P(x, y, z) = 1] \geq \frac{3}{4}$$

$$x \notin L \Rightarrow Pr_y[\exists z, P(x, y, z) = 1] \leq \frac{1}{4}$$

The strings y and z are polynomially bounded in length of x.

Remarks: We can see that $AM \subseteq IP$. AM is a restricted version of IP. Arthur simply tosses coins and chooses a random string y and sends it to Merlin. Merlin sends back a certificate z. Arthur, evaluates the polynomial predicate $P(x, y, z)$. If $x \in L$, for most coin tosses y of Arthur, Merlin will have some valid proof z. On the other hand, if $x \notin L$, for most y, Merlin will not have a valid proof z.

As usual, we can amplify the success probabilities. Arthur, chooses many random strings, $y_1, y_2, \ldots y_r$ and sends them all to Merlin. Merlin replies back with $z_1, z_2, \ldots z_r$. The predicate P is modified to P' which would simulate $P(x, y_1, z_1), \ldots P(x, y_r, z_r)$. Then take the majority. We can use Chernoff bounds to show that the success probability is amplified exponentially in r. Thus, we can replace $\frac{3}{4}$ by $1 - \frac{1}{2^n}$, and $\frac{1}{4}$ by $\frac{1}{2^{poly(n)}}$.

Now suppose we have a university president who wants no error. He wants probabilities 1 and 0, respectively. For example, if $x \notin L$, then $\forall y \forall z$ it is false. Indeed, we can consider, four possibilities here.

(1.) $\begin{cases} x \in L & \text{then for most } y, \text{ there exists } z \text{ that makes it true} \\ x \notin L & \text{then for most } y, \text{ for all } z \text{ it is untrue} \end{cases}$

(2.) $\begin{cases} x \in L & \text{then for all } y, \text{ there exists } z \text{ that makes it true} \\ x \notin L & \text{then for most } y, \text{ for all } z \text{ it is untrue} \end{cases}$

(3.) $\begin{cases} x \in L & \text{then for all } y, \text{ there exists } z \text{ that makes it true} \\ x \notin L & \text{then for all } y, \text{ for all } z \text{ it is untrue} \end{cases}$

(4.) $\begin{cases} x \in L & \text{then for most } y, \text{ there exists } z \text{ that makes it true} \\ x \notin L & \text{then for all } y, \text{ for all } z \text{ it is untrue} \end{cases}$

Statement (1.) is simply the definition of AM. Statements (3.) and (4.) are both cases which fall into NP. Statement (2) defines, what we call AM_1, one sided error version of AM. We discuss it shortly.

We note that we can always transform $\exists z_{|z| \leq n^{O(1)}} P(x, y, z) = 1$ to a satisfiability issue by using Cook's theorem. Therefore, the general form of AM is:

For $x \in L$, then for most y, we get $\varphi_{x,y} \in$ SAT.

For $x \notin L$, then for most y, we get $\varphi_{x,y} \notin$ SAT.
where the map $x, y \mapsto \varphi_{x,y}(z)$ is computable in polynomial time.

We can think of AM as follows: We think of AM by writing $\text{AM} = \text{BP} \cdot NP$.

We define AM_1 to be the statement of (2.). Formally,

Definition (AM_1): $L \in AM_1$ iff $\exists$ a polynomial time predicate P such that

$$x \in L \Rightarrow Pr_y[\exists z, P(x, y, z) = 1] = 1$$

$$x \notin L \Rightarrow Pr_y[\exists z, P(x, y, z) = 1] \leq \frac{1}{4}$$

AM_1 can be thought of as $co-RP.NP$. It is immediate that $\text{AM}_1 \subseteq \text{AM}$. We have the following surprising theorem. We now prove that $AM = AM_1$. Equivalently, BP.NP=coRP.NP.

Theorem: $AM = AM_1$.
Proof: Let L be in AM and A be the underlying polynomial predicate. We amplify the success probability such that,

$$x \in L \Longrightarrow P_{y \in \{0,1\}^m}[\exists z \in \{0,1\}^r, A(x, y, z)] \geq 1 - \frac{1}{m}$$

$$x \notin L \Longrightarrow P_y[\exists z, A(x,y,z)] \leq \frac{1}{m}$$

Recall that a set $E \in \{0,1\}^m$ is called thin if $|E| \leq 2^m/m$. E is called fat if E^c is thin. A set may be neither thin nor fat. Let $k \sim \log m$.

We already proved the following lemma about fat and thin sets:-
Lemma: If E is fat then,

$$Pr_{s_1,s_2,\ldots s_m \in \{0,1\}^m}[\bigcap_{i=1}^{m}(E \oplus s_i) \neq \phi] = 1$$

If E is thin, then,

$$Pr_{s_1,s_2,\ldots s_m}[\bigcap_{i=1}^{m} E \oplus s_i \neq \phi] \leq \frac{1}{2^{m(k-1)}}$$

Fix an input x. Let $E_x = \{y|\exists z, A(x,y,z) = 1\}$. If $x \in L$, then, E is fat. If $x \notin L$, E is thin. We shall construct a new one sided error protocol for L as follows:- Suppose Merlin claims $x \in L$. He will prove this claim by showing that E_x is fat. To do this, Arthur will give him set of random shift strings $s_1, s_2, \ldots s_m$. Merlin will produce a $y \in \bigcap_{i=1}^{m} E \oplus s_i$. By above lemma, for any set of random strings, such a y exists. But, Merlin has to prove that y, indeed, has this property. Equivalently, for $1 \leq i \leq m$, $y \oplus s_i \in E$. To prove this, he produces, $z_1, z_2, \ldots z_m$ such that, $A(x, y \oplus s_i, z_i) = 1$. On the other hand, if $x \notin L$, by the second part of the above lemma, the probability that such a y exists is very small.

Formally, we now present a polynomial predicate A' as required by the definition of AM_1. $A'(x,s,z')$ takes three arguments, where x is the input string, $s =< s_1, s_2, \ldots s_m >$ will be a string of length m^2; $z' =< y', z_1, z_2, \ldots z_m >$ will be a string of length $m + mr$, where r is the length of z in the original AM protocol.

$$A'(x, < s_1, s_2, \ldots s_m >, < y', z_1, z_2, \ldots z_m >) = \bigwedge_{i=1}^{m}(A(x, y' \oplus s_i, z_i)$$

As explained already, with the help of the "fatness" lemma, we can easily verify that A' has the required properties. ♣

We now compare IP with AM. One main difference is that, in IP, the coin tosses of Arthur are *private*, whereas, in AM, they are *public*. In IP, Merlin cannot see the coin tosses of Arthur, whereas, in AM he *can* see them. (In our definition of AM, Merlin can see the "y" and then say his proof "z".) Nothing is hidden from Merlin. To explain this better, recall our IP protocol for Graph non-isomorphism.

Suppose $G_1 \ncong G_2$. The original interactive proof system worked as follows:- the verifier takes $b \in_R \{0,1\}$ and $\sigma \in_R S_n$. Then, sends over $\sigma(G_b)$ to the prover (and hides his coin toss). Then the prover sends back b' and claims that $G_{b'} \cong \sigma(G_b)$. The verifier then verifies

this. We can't use this protocol straightaway in the Arthur-Merlin setup. In AM, Merlin invites Arthur to send the bits, i.e., σ and b. Now Merlin can easily cheat Arthur.

But, we can design a clever AM protocol for GNI. Play this game. $\mathrm{Aut}(G)$ is defined to be the set of all graphs isomorphic to G. Look at $\mathrm{Aut}(G_0) \cup \mathrm{Aut}(G_1)$. This set is large, in fact much larger than one of $\mathrm{Aut}(G_0)$ or $\mathrm{Aut}(G_1)$. If $G_0 \cong G_1$ then these sets are the same. Otherwise, we have twice the size for this larger set. Merlin tells us which one of the sets is larger. Merlin engages in a proof of the size of these.

7.3 Merlin–Arthur Games

MA is similar to AM, except that Merlin moves first. He presents a proof to Arthur(a probabilistic polynomial time player), who verfies whether the proof is valid.

Definition (MA): L is in MA if, there exists a polynomial time computable predicate $A(.,.,.)$ such that,

$$x \in L \Longrightarrow \exists z Pr_y[A(x,y,z) = 1] \geq \frac{3}{4}$$

$$x \in L \Longrightarrow \forall z Pr_y[A(x,y,z) = 1] \leq \frac{1}{4}$$

where $|y|$ and $|z|$ are polynomially bounded in $|x|$.

Remarks: Here, if $x \in L$, a single proof "z" should work for most of the coin tosses "y" of Arthur. If $x \notin L$, no matter what proof Merlin presents, Arthur is unlikely to accept it. MA is like N.BPP. Similar to NP, but the verfication procedure is in in BPP. The proof "z" is a *publishable proof*. You can write it down and then check it. It's like a relaxed way of proving.

Again, we can amplify this from $\frac{3}{4}$ to $1 - 2^{-q(n)}$, for any polynomial $q(n)$.

Proposition: $BPP \subseteq MA$.
Proof: Arthur simply ignores Merlin, and just tosses some coins and simulats the BPP algorithm. As we remarked, MA is like $N.BPP$, so it's quite obvious that $BPP \subseteq N.BPP = MA$. ♣

It's easy to see that,
Proposition: $NP \subseteq MA$. ♣

As in AM, we can define the one sided error version of MA, namely MA_1:-
Definition (MA_1): L is in MA if, there exists a polynomial time computable predicate $A(.,.,.)$ such that,

$$x \in L \Longrightarrow \exists z Pr_y[A(x,y,z) = 1] = 1$$

$$x \in L \Longrightarrow \forall z Pr_y[A(x,y,z) = 1] \leq \frac{1}{4}$$

where $|y|$ and $|z|$ are polynomially bounded in $|x|$.

We saw that $AM = AM_1$. We can prove a similar result for MA:-
Theorem: $MA = MA_1$. Proof is left as a homework. ♣

MA is like $N.BPP$ and MA_1 is like $N.coRP$. We have shown above that $N.BPP = N.coRP$. Can we employ similar techniques to show that BPP $\subseteq$ coRP? Unfortunately, we can't. Because we do not allow the intervening of the prover. In the proof that MA $\subseteq$ MA_1 we allowed the prover to give the proof and the shifts, so we did have the intervening of the prover.

We now establish a link between MA and AM. We show that $MA \subseteq AM$. Let $L \in MA$ via a polynomial predicate $A(.,.,.)$ that meets the requirements mentioned in the definition of MA. One naive attempt to prove $L \in AM$ would be to use the same predicate A for the AM protocol. This would work, when $x \in L$. As per MA, there is some proof z_0 that works for most of Arthur's coin tosses y. In AM, irrespective of Arthur's coin tosses, Merlin can give this same proof z_0. This would satisfy, $Pr_y[A(x, y, z_0) = 1] \geq 3/4$. But, this idea fails in case $x \notin L$. A satisfies the condition that, for any z, only a small fraction y would work. But, for each y, some z may work. So, Merlin can cheat Arthur when $x \notin L$. Our proof will make use of this naive idea, but, by amplifying sufficiently, the success probability of MA, we avoid the pitfall described above.

Theorem: $MA \subseteq AM$.
Proof: Let $\in MA$. As we know that $MA = MA_1$, $L \in MA_1$. Furthermore, we can amplify the success probability so that, we have a polynomial predicate $A(.,.,.)$ that satifies,

$$x \in L \iff \exists z \left[Pr_y[A(x, y, z) = 1] = 1\right]$$

$$x \notin L \iff \forall z \left[Pr_y[A(x, y, z) = 1] \leq \frac{1}{2^{m+1}}\right]$$

where, $m = |z|$, and m is polynomial in $|x|$.

Now we can use the same predicate A for our AM protocol. The main point is that, when $x \notin L$, for any z, most of the y's do not work. By a counting argument, we see that, for half of the y's, no z works:-

$$Pr_y[\exists z, A(x, y, z) = 1] \leq \sum_{z \in \{0,1\}^m} [Pr_y[A(x, y, z) = 1] \leq \sum_{z \in \{0,1\}^m} \frac{1}{2^{m+1}} = \frac{1}{2}$$

On the other hand, when $x \in L$, there is some z_0, that satisfies, $Pr_y[A(x, y, z_0) = 1] = 1$. This z_0 can serve as a proof for Merlin, for all the coin tosses of Arthur. We have shown that,

$$x \in L \iff Pr_y[\exists z, A(x, y, z) = 1] = 1$$

$$x \notin L \iff Pr_y[\exists z, A(x, y, z) = 1] \leq \frac{1}{2}$$

We can reduce the probability of getting cheated, from 1/2 to smaller value, by repeatedly running, A. ♣

7.4 AM With Multiple Rounds

In AM and MA, we had two rounds of interaction, one by Arthur, and one by Merlin. We can generalize to include more rounds of interactions. For example, we have MAM. Merlin moves first, then Arthur moves, then Merlin.

Definition (MAM):

$$x \in L \Longleftrightarrow \exists z_1 \left[Pr_y[\exists z_2, A(x, y, z_1, z_2)] \geq \frac{3}{4} \right]$$

$$x \in L \Longleftrightarrow \forall z_1 \left[Pr_y[\exists z_2, A(x, y, z_1, z_2)] \geq \frac{3}{4} \right]$$

We can think of MAM in two ways. One way is MAM=N.AM, just like NP, but the verfication procedure is an AM predicate (instead of a polynomial predicate. To get the second way, first think of MA as MA(P). The undelying predicate is a polynomial predicate. Then, MAM is MA(NP). In other words, take the original definition of MA, and replace the phrase, "where A is a polynomial predicate" by "where A is a NP predicate".

As with MA and AM, we can also have a one sided error version of MAM, named MAM_1. We can prove the following theorem, using, essentially, the same technique we used to prove that $AM = AM_1$.

Theorem : $MAM = MAM_1$. ♣

Then, we proceed as in the proof of $MA \subseteq AM$, to show that $MAM \subseteq AMM$. But, what is AMM? This means, that Arthur picks the random string y_1. Then, Merlin gives a proof z_1. Then, again Merlin gives a proof z_2. But, Merlin can combine these two moves into a single move and give $z_1 \cdot z_2$. Thus, AMM is same as AM. In other words, we extend the proof $MA \subseteq AM$ to show that $MAM = MA(NP) \subseteq AM(NP)$. But what is AM(NP)? Observe that it is same as AM. We have proved that:-

Theorem: $MAM = AM$. ♣

We can generalize further to involve more rounds mixing M's and A's. For example, MAAAMMAAAMM. But we can certainly, replace a streak of A's by a single A and a streak of M's by a single M. In the above example, we have MAAAMMAAAMM⊆ MAMAM. Just like MAM is same as AMM, we can make MAMAM to be AMMAM, by changing the starting MA to be AM. Now again compress the "MM" into "M" to get AMAM. We can continue this process of turning "MAM" into "AM" then, compressing streaks of M's into a single M (and streak of A's into single A) to get AM. Define, AM[k] to be AM with k rounds of interaction. We have "shown" that,

Theorem: AM[k]=AM. ♣

A formal proof can be given by by using induction. The above theorem is valid only when k is a constant. What if the number of rounds is dependent on the input length? Meaning, if

the input is x, we allow $poly(|x|)$ rounds of intreaction. Define AM[poly] to be such a class. Then, it's known that

Theorem: AM[poly]=PSPACE. ♣

We have proved that

$$MA_1 = MA \subseteq AM_1 = AM = MAM$$

In fact, we showed that, any finite (fixed number of) iterations of M and A is $\subseteq$ AM.

7.5 AM and Other Complexity Classes

We now study the relationship of AM with respect to other classical complexity classes. Recall that, L is said to be in Σ_2^p, if there exists a polynomial predicate $D(.,.,.)$ such that,

$$x \in L \Longleftrightarrow \exists y \forall z[D(x,y,z) = 1]$$

where $|y|$ and $|z|$ are polynomially bounded in $|x|$. A language L is said to be in Π_2^P, if its compliment L^c is in Σ_2^P.

We know that AM is contained in Π_2^P. This is easy to see that $\text{AM}_1 \in \Pi_2^P$. L is in Π_2^P, if there is a polynomial predicate A such that,

$$x \in L \Longleftrightarrow \forall y \exists z[A(x,y,z) = 0]$$

$$x \notin L \Longleftrightarrow \exists y \forall z[A(x,y,z) = 1]$$

Verify that, $\text{AM}_1 \subseteq \Pi_2^P$. If $x \in L$, for all coin tosses y of Arthur, Merlin has some proof z. Whereas, if $x \notin L$, we have some y (in fact, most of the y), for which, all proofs of Merlin are invalid. As $\text{AM}_1 = \text{AM}$, we have proved the theorem,

Theorem: $\text{AM} \subseteq \Pi_2^p$ ♣

We know that $NP \subseteq$ AM. We also know that graph non-isomorphism is in AM. Can we generalize this to prove that, $coNP \subseteq$ AM? But it is unlikely, as shown by the following theorem.

Theorem: If $coNP \subseteq$ AM, the polynomial hierarchy (PH) collapses to AM.
Proof: The proof is a simple application of our theorem that $MAM =$ AM. From the definition of Σ_2^P, we see that, $\Sigma_2^P = N.(coNP)$. As we assumed that, $coNP \subseteq$ AM, we have $\Sigma_2^P \subseteq N.(\text{AM})$. We already observed that $N.\text{AM} = MAM$ and that $MAM =$ AM. Thus, $\Sigma_2^P \subseteq$ AM. As, AM $\subseteq \Pi_2^P$, we have that $\Sigma_2^P = \Pi_2^P =$ AM, which implies that $PH = AM$. ♣

We now prove that $MA \subseteq ZPP^{NP}$. We use ideas from our proof that $BPP \subseteq ZPP^{NP}$.

Theorem: $MA \subseteq ZPP^{NP}$.
Proof: Let $L \in MA$. There exists a polynomial predicate D, such that,

$$x \in L \iff \exists y \, [Pr_z[D(x,y,z) = 1]] = 1$$

$$x \notin L \iff \forall y \, [Pr_z[D(x,y,z) = 1]] \leq \frac{1}{m}$$

where, $|y| = |z| = m$, and m is a polynomial in $n = |x|$. Such an amplification can easily be achieved.

Recall that, a set $E \subseteq \{0,1\}^m$ is called thin, if $|E| < 2^m/m$. E is called fat, if E^c is thin. That is, $|E| \geq (1 - 1/m)2^m$. Call E to be perfect if E consists of all strings in $\{0,1\}^m$.

Recall our proof of $BPP \subseteq ZPP^NP$. There given input x, we tested, whether the witness set E_x, is fat or thin, in a zero error manner, using a SAT oracle. We are going to do the same, with little more sophistication.

Fix an input x. We say that y beats z if $D(x,y,z) = 1$. For a y, let E_y be the set of z's that y beats. The main observation is that, if $x \in L$, there there is some y, such that, E_y is perfect. If $x \notin L$, then for all y, E_y is thin. These are the two conditions that our ZPP machine will test (with the help of an oracle.)

We shall first try to find whether there is a y with perfect E_y. We pick, at random some polynomial z_i's. Then ask the oracle, if there is some y that beats all these z_i. If the answer is 'no', we can safely reject x. But, if the answer is 'yes' we can't be sure and we proceed. We then use self-reducibility to find y_0 that beats all these z_i. We then ask the oracle, if this y_0 beats all z's. If so we can safely accept x. If not, we next try to check whether E_y is fat or thin. To do that wee pick m random strings $s_1, \ldots s_m$. And ask the oracle, if E_{y_0}, when shifted by these strings will cover the all the z's. Verify that this is an NP question. If E_{y_0} is fat we have a high probability that the answer would be 'yes'. On the other hand, if E_{y_0} is thin, the probability is zero. We also know that, if x is not in L, for all y_0, E_{y_0} is thin. Thus, if we get the answer 'yes' from the oracle, we can safely accept. If we get the answer 'no', we are in trouble. Three cases are possible here:-

- Case 1: $x \in L$, but the oracle gave us a y_0 with a thin E_{y_0};
- Case 2: $x \in L$, the oracle gave us a a good y_0, but our shift strings were not good enough to cover (a low probability event).
- Case 3: $x \notin L$, and so the y_0 is thin.

So, if we get the answer 'no', we give up. We see that case 2 is a low probability event. Cases 1 and 2, can happen only when there is a thin y_0, that can beat our random strings $\{z_i\}$. But E_{y_0} being thin, how can y_0 beat so many z_i? We argue this formally later.

Formally, the ZPP machine M, given SAT oracle works as follows:- Let x be the input string.

1. Pick m^2 strings $z_1 \ldots z_{m^2}$ at random from $\{0,1\}^m$.

2. Ask the oracle if there is a y such that for all i, $D(x,y,z) = 1$. If the answer is 'no', then REJECT. If 'yes', use self-reducibility to get a y_0 that beats all these z_i.

3. Choose m strings $s_1, \ldots s_m$, at random from $\{0,1\}^m$.

4. Ask the oracle, whether E_{y_0} when shifted by these random strings will cover all the z's. That is, ask if the following predicate is true or not

$$\forall z \left[z \in \bigcup_{i=1}^{m} E_{y_0} \oplus s_i \right]$$

   If the above predicate is true, then ACCEPT. If not, output "?".

Correctness: In step 2, when we reject, we have that for any y, it fails to beat at least one z_i. So, there cannot be any perfect y. We are correct in rejecting x. In step 4, suppose we ACCEPT. We claim that we are again correct in accepting x. By contradiction, suppose $x \notin L$. Then every y is bad. For a bad y, $E_y < 2^m/m$. So shifting by m strings cannot cover all of $\{0,1\}^m$.

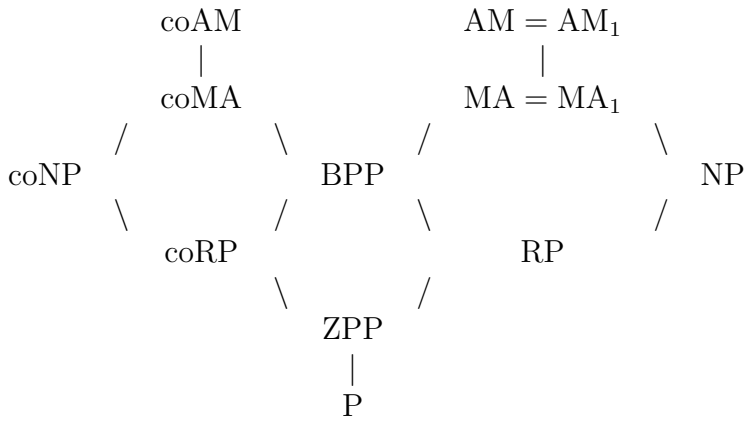
Now we argue that the probability that we output "?" is low. First let's bound the probability that we will have a non-good y_0 at the end of step 2. Fix a y_0 which is not good. That is $Pr_z[y_0 \text{ beats } z] \leq (1 - 1/m)$. We picked up m^2 strings at random. The probability that y_0 beats all these strings is $\leq (1-1/m)^{m^2} \sim e^{-m}$. There are at most 2^m such y's. So, the probability that there exists some y which is not good, but beats all these m^2 strings is $\leq 2^m/e^m$. This probability is exponentially small. So, with high probability, there are no non-good witnesses that can beat all the m^2 strings. (So, if $x \notin L$, we would stop at step 2, with high probability).

Assume that we proceeded to step 4, and y_0 is good. Then, with high probability, the shifting by m strings would cover all of $\{0,1\}^m$. This we proved already in proving that $BPP \in ZPP^{NP}$. Recall the lemma that if $S \in \{0,1\}^m$ has cardinality at least $2^m(1-1/m)$, then,

$$Pr_{s_1 \ldots s_m \in \{0,1\}^m} \left[\bigcup_{i=1}^{m} S \oplus s_i = \{0,1\}^m \right] \geq 1 - \frac{1}{2^{m(k-1)}}$$

where $k \sim \log m$. So the probability that we output "?" is bounded by $(2^m/e^m) + (2^{-m(k-1)})$. ♣

The structure among the complexity classes, so far is:- /* A BETTER PICTURE WITH ZPP^{NP} TO BE DRAWN */



7.6 LFKN Protocol for Permanent

We start the discussion by reviewing the definition of permanent of a matrix. Let $M = (a_{ij})$ be a $n \times n$ matrix where each entry is n bits long. We define,

$$\text{Perm}(M) = \sum_{\sigma=(\sigma_1,\ldots,\sigma_n)\in S_n} a_{1\sigma_1} \cdots a_{n\sigma_n}$$

where S_n is the set of all permutations over the set $\{1, \ldots n\}$.

There is alternative way to express the permanent. It is similar to the way we usually compute the determinent, except that we do not have those minus signs:

$$\text{Perm}(M) = \sum_{i=1}^{n} a_{1i} \cdot \text{Perm}(M_{1i})$$

where M_{1i} is the $(n-1) \times (n-1)$ submatrix obtained by deleting row 1, and column i. The proof of equivalence of the two definitions is that every term appears exactly once in both expressions.

We are now ready to discuss the interactive proof system, The input is a $n \times n$ matrix M, with each entry n-bits. Since permanent has $n!$ terms, $|\text{Perm}M| \leq 2^{n^{O(1)}}$. The prover P first chooses a huge prime $p > 2^{n^{O(1)}}$ and sends it to the verifier V. Using the recent polynomial time AKS algorithm for primality testing, the verifier can check that p is indeed a prime number. The prover next sends the $\text{Perm}(M)$ modulo p. As p is large enough, taking modulo p is fine. Hereafter, all the actions take place modulo p.

So the prover has produced a number α with the claim that $\text{Perm}(M) = \alpha \bmod p$. If prover's claim is true, we want the verifier to get convinced with probability 1. If not, the verifier has to be convinced with only a small probability.

The protocol revolves around the following polynomial. Define

$$F(x) = \sum_{i=1}^{n} \frac{(x-1)(x-2)\cdots\widehat{(x-i)}\cdots(x-n)}{(i-1)(i-2)\cdots\widehat{(i-i)}\cdots(i-n)} M_{1i}.$$

The above notation means that in the ith term of the summation, $(x-i)$ is missing in the numerator and $(i-i)$ is missing in the denominator. The coffiecients of the polynomial are $(n-1)\times(n-1)$ matrices. The polynomial has the following important property.

If we set $x=i$, of the n terms in the summation, except the ith term, all the other terms evaluate to 0. The ith term evaluates to M_{1i}. Thus we have $F(1)=M_{11}$, and in general, for $1\leq i\leq n$, $F(i)=M_{1i}$. $F(x)$ has been constructed using the matrix analog of the well-known Lagrange interpolation The polynomial can be viewed in two ways. It is a polynomial whose coefficients are $(n-1)\times(n-1)$ matrices. It is also a $(n-1)\times(n-1)$ matrix whose entries are polynomials (in x) of degree $n-1$. We will view $F(x)$ mainly using the second method.

Think of $F(x)$ as a $(n-1)\times(n-1)$ matrix whose entries are polynomials of degree $(n-1)$. What is the permanent of $F(x)$? It is a polynomial of degree $\leq(n-1)^2$. Denote this polynomial as $g(x)=\mathrm{Perm}(F(x))$. The polynomial $g(x)$ has many useful properties. First of all it can used to calculate the permanent of the minors M_{1i} easily:

$$\mathrm{Perm}[M_{1i}]=\mathrm{Perm}[F(i)]=Perm[F(x)]|_{x=i}=g(i).$$

Secondly, we can calculate the permanent of M:

$$\mathrm{Perm}[M]=\sum_{i=1}^{n}a_{1i}\cdot\mathrm{Perm}[M_{1i}]=\sum_{i=1}^{n}a_{1i}\cdot g(i).$$

The beauty of the polynomial $g(x)$ is that it can do more! We saw that, for any $1\leq i\leq n$, $g(i)=\mathrm{Perm}(M_{1i})$. But, in fact, $g(x)$ can be used to compute the permanent of many other matrices! Choose any $r\in Z/p$. Evaluate F at r to obtain the $(n-1)\times(n-1)$ matrix $N_r=F(r)$. If $1\leq r\leq n$, N_r is nothing but M_{ir} and for other values, N_r is some matrix! The main idea of the proof is that $g(r)=\mathrm{Perm}(N_r)$, for any $r\in Z/p$. Thus g can be used to compute permanent of some p many matrices (and $p>>n$). Still g is very short: it is only polynomially long in n. Now let us get back to the discussion of the interactive proof system and put our observations into use.

So the prover has sent a value α with claim that $\alpha=\mathrm{Perm}(M)$. Next he sends a polynomial $g'(x)$, with the implicit claim that $g'(x)=g(x)$ (of couse, he could be lying!). The verifier first checks whether these two claims are consistent: check if

$$\alpha=\sum_{i=1}^{n}a_{1i}\cdot g'(i).$$

If the above test fails, verifier halts and rejects the prover's claim. Suppose the above test passed.

We can make a simple observation:

$$\alpha\neq\mathrm{Perm}(M)\Longrightarrow g\neq g'.$$

Even if the prover is a liar, he has to be consistent with his lies! If he lies and gives $\alpha\neq\mathrm{Perm}[M]$, he has to lie again and give a g' different from g. Otherwise, the above consistency check will fail and verifier will catch him!

The next step is the heart of the proof. The verifier computes the polynomial $F(x)$. He can compute $F(x)$ easily, without the help of the prover. Next the verifier chooses a random number r from Z/p and computes $N_r = F(r)$. As we saw already, permanent of N_r can be computed

$$\text{Perm}(N_r) = \text{Perm}[F(r)] = g(r).$$

Of course, the verifier cannot compute the polynomial g. But, he has the g' given by the prover, who claimed that $g' = g$. So the verifier simply computes $g'(r)$ and assumes that $g'(r) = \text{Perm}(N_r)$.

There are two possible cases to analyze: $g' = g$ and $g' \neq g$. If $g' = g$, then clearly, $g'(r) = \text{Perm}(N_r)$. On the other hand, suppose $g' \neq g$. What is the probability (over the random choice of r) that $g'(r) = \text{Perm}(N_r)$? We know that g and g' are both polynomials of degree $(n-1)^2$. So they can agree on at most $(n-1)^2$ many choices of $x \in Z/p$. As r was a random value chosen from Z/p,

$$\Pr_{r \in Z/p}[g'(r) = g(r)] \leq \frac{(n-1)^2}{p}.$$

We chose p to be exponential in n. So except for a (exponentially) small probability, if $g' \neq g$, then $g'(r) \neq \text{Perm}(N_r)$. Let us summarize the discussion so far: if $\alpha \neq Perm[M]$, then $g' \neq g$ and hence, with probability exponentially close to 1, $g'(r) \neq \text{Perm}(N_r)$. But, by claiming that $g = g'$, the prover has, implicitely, claimed that $\text{Perm}(N_r) = g'(r)$.

The situation the same as the one with which the story started, but it is "smaller". Initially, the provr claimed that $\alpha = \text{Perm}(M)$, where M was a $n \times n$ matrix. Now, he has claimed that $g'(r) = \text{Perm}(N_r)$, where N is a $(n-1) \times (n-1)$ matrix. And if the prover' first claim was a lie, then the second claim is also a lie, with very high probability! How can the verifier validate the claim? Simply, by repeating the same process!

After $(n-1)$ rounds, the two players would be left with a matix of size 1×1, say (x), and the prover would have (implicitely) claimed its permanent to be some β. The verifier can esily check this: Is $\text{Perm}[(x)] = x = \beta$? If not he rejects the original claim of the prover that $\alpha = Perm(M)$. If $\alpha \neq \text{Perm}(M)$, then probability that $\beta \neq \text{Perm}[(x)]$ is at least $1 - n \cdot (n-1)^2/p$. (The factor n comes from the n rounds). We conclude that a honest prover will win with probability 1. And a dishonest prover will win with probability exponentially small.

Exact protocol: M is the $n \times n$ input matrix. The prover claims that $Perm(M) = \alpha$. The first thing prover does is to choose a prime number $p \sim 2^{n^3}$, (note: $p >> Perm(M)$). He sends it to the prover along with a proof that it's prime (using Pratt's theorem that Primes$\in$ NP). All the computation hence forth take place modulo p.

1. Prover's claim is that $Perm(M) = \alpha$, where $M = (a_{ij})$ is a $n \times n$ matrix.
2. If $M = (x)$ is a 1×1 matrix, the verifier checks $\alpha = x$. If they do not match he *rejects* else *accepts*.

3. The verifier computes

$$F(x) = \sum_{i=1}^{n} \frac{(x-1)(x-2)\cdots\widehat{(x-i)}\cdots(x-n)}{(i-1)(i-2)\cdots\widehat{(i-i)}\cdots(i-n)} M_{1i}.$$

4. Prover sends a polynomial $g'(x)$ degree $(n-1)^2$ (with the implicite claim that $g'(x) = \text{Perm}[F(x)]$).

5. Verifier checks

$$\alpha = \sum_{i=1}^{n} a_{1i} \cdot g'(i).$$

   If they do not match *reject*.

6. Verifier picks $r \in_R Z/p$ and sends it to prover.

7. Verifier computes $N_r = F(r)$, a $(n-1) \times (n-1)$ matrix.

8. Verifier sets $M \longleftarrow N_r$, $n \longleftarrow n-1$ and $\alpha \longleftarrow g'(r)$.

9. Go back to step 1.

If the $\alpha = Perm(M)$, the prover can provide correct polynomials g and the verifier will accept with probability 1. Suppose $\alpha \neq \text{Perm}(M)$. The verifier will reject in step 4 of some round. If that does not happen, to make verifier accept in the end in step 1, the verifier has to be lucky in at least one round, where the random r satisfies $g'(r) = g(r)$. Probability for this to happen in any particular round is $\leq (n-1)^2/p$. So probability for it to happen in one of the n rounds is $\leq n(n-1)^2/p$. As $p \sim 2^{n^3}$, this error probability is exponentially small. ♣