Q2) Show that a language $L \in ZPP$ if and only if $L$ has average polynomial time algorithm that gives right answer.

Solution:

To prove: $L \in ZPP$ iff $L$ has an average polynomial time algorithm that always give right answer.

Proof: first let us define ZPP class.

ZPP (zero error probabilistic polynomial time) is complexity class of problem for which probabilistic turing machine exist with these properties:

(1) It always return correct YES or NO answer.

(2) Answer is always do not know or correct answer.

- We define what we mean by average time. We take average time of $A(x,r)$ over all random sequence $r$ and input $x$.
- For size $|x|=n$ we take worst time taken for all inputs.
- We construct algorithm that always return right answer. To do so we run ZPP algorithm. If ZPP algorithm outputs '?' (do not know) we run algorithm again.
- Let the running time of ZPP algorithm be $T$. Then average running time of new algorithm will be

$$T_{avg} = \frac{1}{2}(T) + 2\left(\frac{1}{4}(2T) + \cdots + \frac{1}{2^k}(kT)\right) = O(T)$$

Now, we want to prove that if language L has an algorithm that runs in polynomial average $t(|x|)$ then this is ZPP!

We run algorithm for $2t(|x|)$ and output a '?' (do not know) if algorithm has not yet stopped. hence this belongs to ZPP!!

- We notice that worst running time of algorithm is polynomial ie $2t(|x|)$.
- Probability that algorithm output '?' is less than $\frac{1}{2}$ since original algorithm has running time $t(|x|)$ so it must stop before time $2t(|x|)$ at least half of times.

Hence proved that a $L \in ZPP$ if and only if L has an average polynomial time algorithm that always give right answer!

Q3) Show that $RP \subseteq BPP$.

To prove: $RP \subseteq BPP$

Proof: ~~We~~ ~~to~~ So Randomized probability time (RP) is complexity class of problem for which probabilistic turing machine exist such that

- it always run in polynomial time in input size
- if correct answer is NO, it always return NO
- if correct answer is YES, then it return YES with probability of atleast $\frac{1}{2}$ (otherwise return NO).

now we know BPP,

A language $L$ is in BPP if and only if there exist probabilistic turing machine M such that

- M run in polynomial time for all input
- for all x in $L$, M output 1 with probability greater than or equal to $\frac{2}{3}$
- for all x not in $L$, M output 1 with probability less than or equal to $\frac{1}{3}$.

So now we know about RP and BPP, we will convert an RP algorithm into BPP algorithm. Incase that input x does not belong to language RP algorithm always give right answer, so it satisfies BPP requirement of giving right answer with probability of atleast $\frac{2}{3}$.

Incase x does belong to language we need to improve probability of correct answer from atleast $\frac{1}{2}$ to atleast $\frac{2}{3}$.

If A is RP algorithm, then for fixed number k we define following algorithm:

$A^{(k)}$ :

input : x,

pick $H_1, H_2 \cdots H_K$.

if $A(x,H_1) = A(x,H_2) \cdots\cdots = A(x,H_K) = 0$ then Return 0

Else return 1

## Correctness of algorithm:

1) In case correct answer is 0, output is always right
2) In case right answer is 1, output is right except when all $A(x,H)=0$.

$$\text{if } x \notin L,\ Pr_{r_1 \ldots r_k}[A^R(x,r_1 \ldots r_k)=1]=0$$

$$\text{if } x \in L,\ Pr_{r_1 \ldots r_k}[A^R(x,r_1 \ldots r_k)=1] \geq 1-\left(\frac{1}{2}\right)^R$$

by right choice of $R$ $1-\left(\frac{1}{2}\right)^k$ would go close to 1. In particular choosing $k=2$ would suffice to have probability larger than $\frac{2}{3}$ that is required to make defination of BPP. By choosing $k$ to be polynomial in $|x|$ we can make probability exponentially close to 1.

hence defination of RP we gave above would be equivalent to defination in which instead of bound of $\frac{1}{2}$ for probability of correct answer when $x \in L$, we have bound of $1-\left(\frac{1}{2}\right)^{q(|x|)}$ for fixed polynomial $q$!.

hence proved, $RP \subseteq BPP$!

Question 4: if coNP=NP, then for every $i>1$, $\Sigma_i = NP$.

To prove: if coNP=NP, then for every $i>1$, $\Sigma_i = NP$.

Proof: We will first prove that under assumption of ~~algorithm~~ theorem that when $i=2$ $\Sigma_2 = NP$.

- Let $A \in \Sigma_2$ and let M be non deterministic oracle that decides A using oracle access to 3SAT.
- Let M' be the non deterministic polynomial time turing machine that decides complement of 3SAT.
- We can now describe non deterministic polynomial time turing machine M'' to decide A :

" On input x :

- M'' guesses accepting computation of M(x), along with oracle queries and answers
- for each oracle question $\phi$ for which a YES answer has been guessed M'' guesses satisfying assignment.
- for each oracle question $\psi$ for which a NO answer has been guessed M'' guesses accepting computation of $M'(\psi)$. "

It is hence easy to verify that $M''(x)$ has an accepting computation if and only if $M^{3SAT}(x)$ has accepting computation.

we can hence we can further by induction on $i$ that $\Sigma_i = NP$. We have proved for the base case where $\Sigma_2 = NP$.

Suppose $\Sigma_{i-1} = NP$, then $\Sigma_i = NP^{\Sigma_{i-1}} = NP^{NP} = \Sigma_2 = NP$ □.

hence proved that when $coNP = NP$, then for every $i \geq 2$, $\Sigma_i = NP$.

**Question 8** Let $A = \{(\langle M \rangle, w) : M$ rejects $w$ in less than $t(|\langle M \rangle|, w)$ time steps, $|\langle M \rangle| \leq \sqrt{\log t(|\langle M \rangle|, w)}$ and $M$'s tape alphabet has size 43.

(a) $A \notin TIME(O(t(n))$

(b) $A \in TIME(O(t(n) \cdot \log t(n)))$.

(a) $A \notin TIME(O(t(n))$

**Proof:** We will use principle of diagonalisation and time hierarchy theorem to solve the problem.

→ Assume that $A$ is decidable in $O(t(n))$, hence there exist some exist some deterministic turing machine $M_A$ that decides $A$ in time complexity smaller than $t(n)$ for every input of size $(n)$.

→ We will construct a turing machine $D$ that diagnoses all machine that run in $O(t(n))$. Machine $D$ takes on input $\langle M \rangle$: encoding of turing machine.

→ for any input $\langle M \rangle$, machine $D$ simulate $M$ running on input $\langle M \rangle$ for $t(|M|)$ steps. Here function $t(n)$ must be time constructible, meaning that there exist TM which

given an input n, compute value of t(n) in O(t(n)) time.

→ Machine D is designed to do opposite of what M does on input ⟨M⟩. If M rejects ⟨M⟩, then D accept and if M accept ⟨M⟩, D rejects

→ Contradiction for $M_L$: Since $M_L$ is assumed to decide L in O(t(n)), let's consider what happens when D is given as ⟨$M_L$⟩ as input. By definition, machine $M_L$ must reject ⟨$M_L$⟩ if it runs in less than $t(|\langle M_L \rangle|)$ steps. However D is constructed to accept ⟨$M_L$⟩ if $M_L$ rejects!!

Therefore $|\langle M_L \rangle, \langle M_L \rangle|$ would be in L if $M_L$ rejects it which is contradiction!

This shows that our initial assumption that L is decidable in O(t(n)) must be false. Therefore,

$$L \notin TIME(O(t(n)))$$

hence proved.

(b) $L \in TIME(O(t(n)) \log(t(n)))$

We will prove that $L \in TIME(O(t(n)) \cdot \log(t(n)))$ by constructing Turing machine for L that decides in time $TIME(t(n)) \log(t(n))$.

→ We construct turing machine R that decides language L. for input $(\langle M \rangle, w)$, R will simulate M on w and checks whether M rejects w in within $t(\langle m \rangle, w)$ steps.

→ Turing machine R simulates M step by step and count the number of steps taken by maintaining a counter. If M rejects before counter exceeds $t(\langle M \rangle, w)$ then R accept $(\langle M \rangle, w)$. If M does not reject w within this bound of time then R rejects.

→ We will now analyze complexity of simulation. The simulation of M on single ~~tape~~ step can be done in time proportional to some polynomial in $(\langle M \rangle)$ because time to simulate step of Turing machine depend upon size of machine description and (count state of tape, and head. Counter can be incremented in $O(\log(t(n))$ time since it is number of bits needed to represent counter upto $t(n)$.

→ now we will provide a bound on running time. The total running time of R is product of number of steps $t(n)$ that M is allowed to take and time taken to simulate each step and managing counter. This gives $O(t(n) \log t(n))$ as upper bound of running time of R.

Since R decides A ~~in~~ and operates within $O(t(n) \log t(n))$ we conclude that A is in $TIME(O(t(n)) \cdot \log t(n))$.

Hence proved $A \in TIME(O(t(n)) \cdot \log t(n))$

Q6:) Show that $ZPP \subseteq RP \cap coRP$.

Proof: Suppose $L \subseteq \{0,1\}^*$ is language in ZPP. We need to show that $L \in RP \cap coRP$.

So before going ahead let's define these classes,

ZPP: class of problem for which a probabilistic turing machine must with these properties.

- runs in polynomial time in input size
- return yes, no, or do not know
- answer is either correct or do not know.

RP: language $L$ is in RP if and only if there exist a probabilistic turing machine must with these properties

- runs in polynomial time in input size
- for all $x$ in $L$, M output 1 with probability greater than or equal to $\frac{1}{2}$
- for $x$ not in $L$, M output 0.

coRP: language $L$ would be in coRP if and only if there exist a probabilistic turing machine with these properties

- runs in polynomial time in input size
- if correct answer is 'YES', it always return 'YES'
- if correct answer is 'NO', it return 'NO' with probability greater than equal to $\frac{1}{2}$, else return yes

Since $L \in ZPP$, there is probabilistic TM $M$ with expected running time bounded by (a polynomial function) $q(|x|)$ that correctly decide membership for every string $x \in \{0,1\}^*$.

We describe an RP algorithm ($M'$) as follows:

"
(1) on input $x$, run machine $2q(|x|)$ many steps

(2) If $M$ stops by this time, output whatever $M$ outputs

(3) otherwise output 0."

Note that this algorithm never makes a mistake when $x \notin L$.

Let us analyze case when $x \in L$. Let $T(x)$ denote running time of $M$ on input $x$. Since $L \in ZPP$, we have

$$E(T(x)) \leq q(|x|)$$

↓ Expectation taken over random choices of $M$. By

markov inequality we have,

$$P(T(x) > 2q(x)) \leq \frac{E(T(x))}{2q(x)}$$

$$\leq \frac{q(x)}{2q(x)} = \frac{1}{2} \quad -(1)$$

Since from (1) probability that $M$ has running time more than $2q(x)$ is less than $\frac{1}{2}$.

hence it follows that,

$$x \in L \Rightarrow P(M'(x)=0) < \frac{1}{2} .$$

Thus M' is an RP machine for language L and $L \in RP$. ——— (*)

Let us now prove that $L \in coRP$.

We can define coRP algorithm as follows:

"on input x

1) run the machine M for $2q|x|$ steps

2) if M stops by this time, whatever M outputs

3) otherwise output 1 "

It's easy to see that algorithm never makes mistake when $x \in L$.

If $x \notin L$ then probability M' makes mistake is strictly less than $\frac{1}{2}$ (using some argument, using markov inequality) hence by definition $L \in coRP$. ——— (*)

hence $L \in RP \cap coRP$ since $L \in ZPP$ (given assumption) hence,

$$\boxed{ZPP \subseteq RP \cap coRP}$$

**Question 7** Show that the RP is in first level of polynomial hierarchy.

**Solution** Let us first define RP class.

A language $L$ is in RP if and only if there exist probabilistic TM $M$ such that,

- $M$ runs in polynomial time for all inputs
- for all $x$ in $L$, $M$ output 1 with probability greater than equal to $\frac{1}{2}$
- for all $x$ not in $L$, $M$ output 0.

We would now prove that $RP \subseteq NP$.

Let $L$ be any language in RP. By definition above there exist a probabilistic turing machine such that for any input $x$ in $L$, there is atleast one random string $r$ (of polynomial length in term of $x$) such that $M(x,r) = 1$. This means that machine $M$ accept input $x$ when random string $r$ is used.

Let's take a case when $x$ is not in $L$, then there would be no such random string $r$ that would make $M(x,r) = 1$ i.e $M$ reject for all random string $r$.

Hence '$r$' serves as certificate for membership of $x$ in $L$. Serves as evidence.

hence RP algorithm can be seen as verifier to show L is in NP.

hence we construct a deterministic turing machine that accept if and only if given x in L with corresponding certificate y.

This new machine operate in polynomial time (since ~~M also~~ M runs in polynomial time and checking $M(x, y) = 1$ also polynomial).

$\therefore L \in NP$ (by definition of NP)

$\Rightarrow \boxed{RP \subseteq NP}$ —①

we know that $NP = \Sigma_1$ (first level polynomial hierarchy)

hence using ① we say RP is in first level polynomial hierarchy.

Question: Suppose we have PP = {L : x ∈ L is decided by probabilistic polynomial time algorithm 3. By decidable we mean probability of algorithm being wrong ≤ 1/2. Why PP is less useful than BPP? Prove BQP ⊆ PP

Solution: Let us define PP, is a complexity class that consist of languages for which there exist a probabilistic polynomial turn algo that can decide membership with error probability ≤ 1/2.

However PP differs from other complexity classes such as BPP (bounded error probabilistic polynomial) and BQP (bounded error quantum polynomial) in way of handling error. In BPP and BQP error is explicitly bounded by atleast polynomial factor ensuring constraint on likelihood of error.

hence while BQP and BPP force error to be less than $\frac{1}{2}$ by atleast polynomial factor, PP merely requires error to be less than $\frac{1}{2}$!

lack of efficient error amplification in PP is notable drawback.

Error amplification: ability to increase probability of correctness by repeating algorithm

In case of PP since error is not bounded by polynomial factor hence it can take exponentially many run to achieve high success probability.

This characteristic make algorithm in PP potentially inefficient than BPP (which achieve higher probability of correctness in polynomial number of runs).

hence efficiency in error reduction contribute to greater utility of BPP as compared to PP class.

PART 2 $BQP \subseteq PP$.

So we can select {H, CNOT, CCNOT} gates. Our reduction to third gates allow us to write 'computation tree' and coefficient of final state can be written as sum of paths in tree.

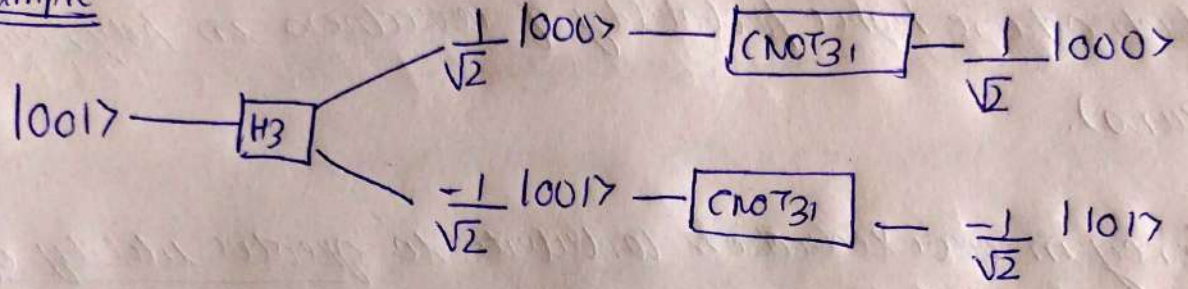
We need error probability to be less than $\frac{1}{2}$ and randomly picking two path in tree and evaluating them give us error strictly smaller than $\frac{1}{2}$.

Key observation:

When a state $|x\rangle$, then CNOT and CCNOT gates only flip one qubit ($|x\rangle \to |y\rangle$) while H split state into superposition with equal magnitude $\left(|x\rangle \to \frac{1}{2}(|y_1\rangle \pm |y_2\rangle)\right)$

hence starting from initial state $|\psi_0\rangle \in (\mathbb{C}^2)^{\otimes 2(n)}$ we can build a tree of depth $(h)$ and $2^h$ leaves ($h = poly(n)$ is number of H gates)

Example




At leaf of tree each leaf is $\pm \frac{1}{2^{h/2}} |y\rangle$ for some $y \in \{0,1\}^{2(n)}$.

Let $P$ denote path from root to a leaf. label(P): state $y$ and sign $P$) sign at that leaf.

Then for final state $|\psi\rangle = \sum_y \alpha_y |y\rangle$

$$|\psi\rangle = \frac{1}{2^{h/2}} \sum_P sign(P) |label(P)\rangle$$

$$= \frac{1}{2^{h/2}} \sum_{y \in \{0,1\}^{2(n)}} \left( \sum_{P: label(P) = y} sign(P) \right) |y\rangle$$

$$\Rightarrow \alpha_y^2 = \frac{1}{2^h} \sum_{\substack{P, P' \\ label(P) = label(P') = y}} sign(P) \cdot sign(P') \quad \text{—③}$$

now we know the coefficient $\alpha_y$, let us compute probability to output 1 by projection operator $\Pi_1 = |1\rangle\langle 1|$

$P_x[C_n(x)=1] = \langle\psi|\Pi_1|\psi\rangle = \sum_{y: y_1=1} \alpha_y^2$. Thus

$$P_x[C_n(x)=1] - P_x[C_n(x)=0] = \sum_y \alpha_y^2 \cdot (-1)^{1(y_1=0)}$$

$$= \frac{1}{2^h} \sum_{\rho,\rho':\, \text{label}(\rho)=\text{label}(\rho')} \text{sign}(\rho)\cdot\text{sign}(\rho')\cdot(-1)^{1(\text{label}(\rho_1)=0)} \qquad (4)$$

Given input $(x)$, we randomly simulate two paths $\rho, \rho'$ in computation tree and compute $\text{sign}(\rho)$, $\text{sign}(\rho')$ and $\text{label}(\rho)$, $\text{label}(\rho')$ This only takes $O(h) = \text{poly}(n)$ time. Then

- If $\text{label}(\rho) \neq \text{label}(\rho')$, then accept or reject with probability $\frac{1}{2}$
- If $\text{label}(\rho) = \text{label}(\rho')$, then accept iff $\text{sign}(\rho)\text{sign}(\rho')(-1)^{1(\text{label}(\rho_1)=0)} > 0$

This quantity is summand in (4), $q = P_r[\text{label}(\rho) = \text{label}(\rho')] > 0$

$$P_M[\text{Accept}] - P_M[\text{Reject}] = q \cdot E\left[\text{sign}(\rho)\cdot\text{sign}(\rho')(-1)^{1(\text{label}(\rho_1)=0)} \,\middle|\, \begin{matrix}\text{label}(\rho) \\ = \text{label}(\rho')\end{matrix}\right]$$

$$\alpha\, P_M[C_n(x)=1] - P_M[C_n(x)=0] \qquad -(5)$$

Thus by definition of BQP, $P_M[\text{Accept}] - P_M[\text{Reject}]$ is strictly positive (resp. negative) if $x \in A$ (resp. $x \notin A$). Thus we have classically achieved error less than $\frac{1}{2}$ in polynomial time

Thus proved $BQP \subseteq PP$.

Hence proved !!

ASSIGNMENT-4 PRISHA 2021101075

Q) Show that $P_{R}[P^{R} = NP^{R}] = 0$ where $P_{R}(\cdot)$ is taken over random oracle $R \subseteq \{0,1\}^{*}$.

Proof Idea: This is the same as showing $P_{R}[P^{R} \neq NP^{R}] = 1$. Instead of having access to a random oracle selected according to the uniform distribution, we have access to an oracle such that $\forall n$ w/p $1/2$ # string of length $n$ in $R$. Also w/p $1/2$ $\exists!$ (There exist exactly one string) string of length $n$ in $R$.

Given $n$ and access to oracle, we want to find out which case holds in poly($n$) time. This is like an NP problem because witness of the fact that there is a string of length $n$ in oracle is string itself.

It seems impossible however to solve the problem deterministically in time polynomial in $n$, whether there is a string of length $n$ in oracle or not, with high probability a polynomial time algorithm will only get 'No' answer from oracle, so conditioned on getting all 'No' answers, algorithm will give some answers whether a string of length $n$ is in oracle or not hence it will be incorrect approximately $\frac{1}{2}$.

To refine the argument, following adjustments

(1) Specify computational problem with string input of length $n$

(2) Base reasoning on uniform random oracle

(3) allow selection of random oracle before excluding polynomial time algorithm

(4) ensure zero probability of polynomial time algorithm existing for hard problem.

PROOF:

for every oracle R, define language $L_R$ such that bit string x of length n is in $L_R$ ($x \in L_R$) iff $\exists y$ s.t $|y|=n$ and $\forall z$ s.t $|z| \leq \log n$, $xyz \in R$.

basically for fixed x ($|x|=n$) if we pick random oracle R then to have $x \in L_R$, we need atleast one of $2^n$ condition (wrt choice of y) to be true.

Each has probability of $\frac{1}{2^n}$ of being true because it corresponds to event that a certain set of n string all belong to oracle

$\therefore$ x has probability $(1-(1-1/2^n)^{2^n} \simeq 1-1/e)$ of being in $L_R$.

and probability $\simeq \frac{1}{2}$ of not being in $L_R$. Furthermore we have:

Claim: for every oracle R, the language $L_R$ is in $NP_R$.

A witness that $x \in L_R$, is a string y such that $xyz \in L_R$ for all z and this can be tested with n oracle queries

note that $x \in L_R$ then we expect small number of witness of this fact, hence we are in setup fairly similar to one described above (with constant probability $x \in L_R$, with constant probability $x \notin L_R$, number of witness is very small).

Lemma 1: Let $M$ be a polytime oracle machine. For all sufficiently large $n$ and all $x$ of length $n$ we have $P_R[M^R(x) - L_R(x)] \leq 2/3$

Proof: Let $M'$ be a machine like $M$, except that if asks the query $(|z| = |y| = n, |z| = \log n)$ then it also ask queries $xyz'$ $\forall$ $z' \in \{0,1\}^{\log n}$. Since $M'$ is polynomial time, let $p(n)$ be a polynomial upperbound to the queries asked by $M'$ of length $n$.

Say $M'$ makes exactly $p(n)$ queries.

A 'transcript' $C$ of computation of $M'^R(x)$ is a description of all answers to the $p(n)$ oracle queries asked by $M'$.

Note however if $M'$ ask a question adaptively ($2^{p(n)}$ questions can be possibly asked & each would have 2 possible answers), then total number of valid transcript is only $2^{p(n)}$ each with probability $\frac{1}{2^{p(n)}}$.

Oracle $R$ is consistent with transcript $C$ if all the oracle queries and answers in $C$ agree with $R$.

If $M^R(x)$ has transcript $C$, $R'$ is consistent with $C$, then output of $M'^R(x)$ is the same as output of $M'^{R'}(x)$.

Hence given a transcript $C$, we can consider the conditional distribution of random oracle $R$ given $C$. This is just the distribution of an oracle that agrees with answer recorded in $C$ for $p(n)$ queries of $C$ and is random for all other inputs. Transcript $C$ is inconclusive for $x$ if it contain no query $xyz$ where $y$ is valid witness that $x \in L_R$.

Claim 2: $B_R[M'^R(x)$ has an inconclusive transcript $] \geq 1 - \frac{n^{O(1)}}{2^n}$

Proof Idea: Imagine the process of running the computation of $M'^R(x)$ and of picking $R$ randomly along the way by making decision for which queries of $M'$ are in $R$ and which are not and delaying every other decision!

Every time $M'$ makes block of queries of form $xyz$ there is probability $\frac{1}{2^n}$ that $y$ is witness of $x$.

$M'$ makes total of $\frac{p(n)}{r}$ query blocks, so by union bound, the probability that it queries a witness of $x$ [ending non inconclusive witness]

is almost $\frac{p(n)}{(n \cdot 2^n)} = \frac{n^{O(1)}}{2^n}$ □.

**Claim 3**: If C is inconclusive transcript of M' on x.

$$P_R[x \in L_R \mid R \text{ consistent with } C] = 1 - \frac{1}{e} + O(1)$$

Proof Idea: Let C be inconclusive transcript. Consider selection of random R consistent with C. Each of $2^n - n^{O(1)}$ string y for which no query xyz is in the transcript has probability of $\frac{1}{2^n}$ for becoming witness that $x \in L_R$, and string y for which xyz are in some transcript are not going to be witness.

$\therefore$ probability that $x \notin L_R$ is $\left(1 - \frac{1}{2^n}\right)^{2^n - n^{O(1)}} = \frac{1}{e} - O(1)$

□

Lemma follows from the claim 2,3. Pick random oracle R by picking random transcript $M'(x)$ and then random R consistent with C. When we pick C we have probability $1 - O(1)$ of picking an inconclusive one. Once transcript $M'(R)(x)$ is picked depending only on C, and is same for R consistent with C.

But for $1/e - O(1)$ fraction of R, the correct one is 'Reject' for $1 - 1/e + O(1)$ fraction the correct answer is Accept.

So, $M'^R(x)$ will be wrong w/p at least $1/e - O(1)$ which is more than $1/3$

The idea is simple. We want to show that what works $NP^R$ will not work for any polytime deterministic algorithm.

Lemma 2: Let $M$ be a polytime oracle machine. $\forall$ large enough $n$ and all $x_1, x_2 \ldots x_n$ each of length $n$ we have,

$$P_R[M^R(x_1) = L_R(x_1) \wedge \ldots M^R(x_n) = L(x_n)] \leq \left(\frac{2}{3}\right)^n.$$

Proof: Let $M'$ be an oracle such that $M'(R)$ simulate $M^R(x_1)$ $M^R(x_2) \ldots M^R(x_n)$. and then return output of $n$ simulation. Furthermore if $M'$ makes query of the form $xyz^0$, ~~$\forall z$~~ then it also make query of the form $xyz^1$ $\forall z^1 \in \{0,1\}^{\text{len}}$ Assume $M'$ make same number of queries to every oracle $= p(n)$

Transcript of $M'$ = record of all queries and answers.

There are $2^{p(n)}$ valid transcript and all are equally likely. A transcript would be ~~true~~ inconclusive if it contain no query $xyz$ where $y$ is valid witness for $x$.

Like we did lemma 1, pick a random transcript. It is ~~to~~ inconclusive with probability at least $1 - \frac{n^{\text{const}}}{2^n}$. So

transcript $C$ is inconclusive, we ~~not~~ sample random $R$ consistent with $C$, then $x_i$ has independently

probability $\frac{1}{e} - o(1)$ of not being in $L_R$ and probability $1 - \frac{1}{e} + o(1)$ being in $\mathcal{L}_R$.

$$P_{R_R}[M^R(x_1) - L_R(x_1) \wedge \cdots - M^R(x_n) = L(x_n)] \leq \left(1 - \frac{1}{e} + o(1)\right)^n + \frac{n^{o(1)}}{2^n}$$

$$\leq \left(\frac{2}{3}\right)^n$$

for large enough $n$!

Proof of theorem: Now let us fix a polynomial time machine $M$, then for sufficiently large $n$ we have,

$$P_R[M^R \text{ correctly decides } L_R] \leq \left(\frac{2}{3}\right)^n$$

and since this is true for very sufficiently large $n$, we have to conclude that

$$P_R[M^R \text{ correctly decide } L_R] = 0$$

finally let $\mathcal{M}$ be (countable) set of all polynomial time oracle machines then,

$$P_R[L_R \in P^R] = 0 P_R[\exists M \in \mathcal{M} \cdot M \text{ decides } L_R]$$

$$\leq \sum_{M \in \mathcal{M}} P[M \text{ decides } L_R] = 0$$

[using countable additivity of uniform distribution over oracles]

$$\therefore Pr_R[P^R = NP^R] = 0$$

□