Institute Name
CS1234 Course Course

Instructor: Jayalal Sarma
Scribe: Prasun Kumar
Date: Jan 12, 2012

LECTURE
5

FP vs #P question

We know $(\#\mathsf{P} = \mathsf{FP}) \implies (\mathsf{P} = \mathsf{NP})$. In the last lecture we left the question about the converse. i.e., is it true that, $(\mathsf{P} = \mathsf{NP}) \implies (\#\mathsf{P} = \mathsf{FP})$?

In this lecture, we will show a weaker version of the above containment. We will define a new class of languages PP (probabilistic polynomial time) in decision world, that contains NP, and will show that if we make a stronger assumption $\mathsf{PP} = \mathsf{P}$ (seemingly stronger than $\mathsf{NP} = \mathsf{P}$) we get the required implication. We will show in particular that $\mathsf{PP} = \mathsf{P} \iff \#\mathsf{P} = \mathsf{FP}$.

5.1 Complexity Class PP

For exploring the complexity class, we revisit the definition of NP. With respect to a "branching" Turing machine(NTM) (which can branch at each computation by using a guess bit), we have two different classes of languages as follows, which differs in their acceptance conditions. We denote for a machine M, by $\#acc_M(x)$ as the number of accepting paths. Let the machine run for $p(n)$ time on each path.

- $\mathsf{NP} = \{\text{L} : \exists \text{ branching TM } M_1, \text{ x} \in \text{L} \iff \#acc_{M_1}(x) \geq 1\}$.
- $\mathsf{coNP} = \{\text{L} : \exists \text{ branching TM } M_2, \text{ x} \in \text{L} \iff \#acc_{M_2}(x) \geq 2^{p(n)}\}$.

In terms of number of accepting paths on input x, the above classes represent the extremes. On one side, in NP we talk of atleast one accepting path, and on the other side, in coNP we talk of all accepting paths. To understand the structure of these classes we can ask the variant of this as : what different class of languages do we get if we set the accepting condition as the number of accepting paths being more than a fraction of the total number of paths on input x. A simpler situation is to consider the fraction to be half and the corresponding class of languages is called PP(probabilistic polynomial) which is defined as follows. We will come across other variants in the later lectures.

DEFINITION 5.1. $\mathsf{PP} = \{\text{L} : \exists \text{ NTM } M, \text{ x} \in \text{L} \iff \#acc_{M_1}(x) > 2^{p(n)-1}\}$

We first understand this complexity class with respect to the other ones we have already seen. We start with the following proposition.

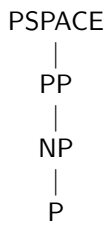
PROPOSITION 5.2. $\mathsf{NP} \subseteq \mathsf{PP}$.

Proof. Let $L \in \mathsf{NP}$ via a nondeterministic turing machine M, $x \in L \iff M$ has atleast one accepting path on x. Our aim is to give another turing machine M' for the same language L such that $x \in L \iff M'$ has more than half of the total number of paths as accepting paths on x.
Description of M' (on input x):

1. Simulate M on x.
2. If M accepts then choose one bit nondeterministically and accept in both branches.
3. If M rejects then choose one bit nondeterministically,accept in one branch and reject in other.

As we remarked in the last lecture, we can assume without loss of generality that the height of computation tree of M on x is exactly $p(n)$ for some polynomial p. Since the total number of paths for M' on input x is $2^{p(n)+1}$, it suffices to prove that $x \in L \iff \#acc_M(x) > 2^{p(n)}$. Let $x \in L$. Since M has at least one accepting path on x. Since M', by construction, creates an imbalance (in count) between the number of accepting and rejecting paths precisely when M accepts, the number of accepting paths in M' will be more than the number of rejecting paths. Thus $\#acc_M(x) > 2^{p(n)}$.
If $x \notin L$, all paths reject in M on input x, and thus number of accepting and rejecting paths in M' on input x are exactly equal and each of themsame is equal to $2^{p(n)}$. □

A similar proof will also show that coNP is contained in PP. Thus we have the following containment relationship among different classes of languages



Now we will prove the main theorem of the lecture which characterizes the FP vs $\#\mathsf{P}$ question in the decision world.

THEOREM 5.3. $(\#\mathsf{P} = \mathsf{FP}) \iff (\mathsf{PP} = \mathsf{P})$

Proof. ($\Rightarrow$) Assume $\#\mathsf{P} = \mathsf{FP}$, our aim is to prove $\mathsf{PP} = \mathsf{P}$. The reverse containment follows since we know $\mathsf{NP} \subseteq \mathsf{PP}$ by proposition5.2. Now we show that $\mathsf{PP} \subseteq \mathsf{P}$. Let $L \in \mathsf{PP}$ via a machine M running in time $p(n)$ for some polynomial p, such that $x \in L \iff \#acc_M(x) > 2^{p(n)-1}$. Define the function, $f(x) = \#acc_M(x)$. By definition $f \in \#\mathsf{P}$ and hence $f \in \mathsf{FP}$. There is a deterministic polynomial time Turing machine N which on input x outputs the value of $f(x)$. Given an x, to test whether it is in L or not, it suffices to test whether whether the MSB of the binary representation of $f(x)$ is 1 or not. This can be done by simply running the machine N on x and testing the MSB of the output. Hence $L \in P$.

($\Leftarrow$) Assume $\mathsf{PP} = \mathsf{P}$, our aim is to prove $\#\mathsf{P} = \mathsf{FP}$. The reverse direction is easy. $\mathsf{FP} \subseteq \#\mathsf{P}$ as we argued in the last lecture. To show the forward direction, let $f \in \#\mathsf{P}$ via M

such that $\forall x \in \Sigma^*$, $f(x) = \#accept_M(x)$. Note that the naive approach to compute $f(x)$ is to compute the number of accepting paths in computation tree of height $p(n)$ will take exponential time. But it suffices to find the minimum $0 \leq k \leq 2^{p(n)}$ such that :

$$k + \#acc_M(x) > 2^{p(n)} \tag{5.1}$$

Indeed, the minimum is achieved when $k = k_{min} = 2^{p(n)} - \#acc_M(x) + 1$. Thus $\#acc_M(x) = 2^{p(n)} - k_{min} + 1$. This is an indirect way of finding out $\#acc_M(x)$ and we are moving towards using our assumption that $\mathsf{PP} = \mathsf{P}$.

We have search problem in hand; to search for the minimum $k(k_{min})$ satisfying equation (5.1). We do this by binary search over the range $0 \leq k \leq 2^{p(n)}$. We solve the decision problem first. Given x and k, check if $k + \#acc_M(x) > 2^{p(n)}$.

For this, we construct an another Turing machine N such that the number of accepting paths is exactly $k + \#acc_M(x)$ and the total number of paths is $2^{p(n)+1}$. Assume such a construction exists. Define a language $A \subset \Sigma^*$ such that $x \in A \iff k + \#acc_M(x) > 2^{p(n)}$. Thus $x \in A \iff \#acc_N(x) > 2^{p(n)}$. This implies that $A \in PP$ (via the machine N !) and hence $A \in P$ by assumption. Given $x \in \Sigma^*$ we can test if $k + \#acc_M(x) > 2^{p(n)}$ by testing if $x \in A$ or not, which can be done in polynomial time. Now we can do this construction and simulation through a binary search in order to find the minimum value of k that satisfies our inequality.

To complete the proof, we give the construction of N (for a given k). We first construct a machine M_k that runs in time $p(n)$ and has exactly k accepting paths. We slightly modify the idea in the previous lecture to do this. Let $\ell = \lceil \log k \rceil$. The machine M_k guesses a $y \in \{0,1\}^{p(n)}$ and *accepts* if the first ℓ bits of y in binary represents a number less than k and the last $p(n) - \ell$ bits is all-zero (lexicographically first path) and *reject* otherwise.

We combine M_k and N (both using exactly $p(n)$ non-deterministic bits), to get the machine N. The machine N on input x guesses 1 bit and on the 0-branch it simulates M_k on x and on the 1-branch it simulates M on x. Clearly $\#acc_N(x) = \#acc_{M_k}(x) + \#acc_M(x) = k + \#acc_M(x)$. The length of each path is exactly $p(n) + 1$ and hence total number of paths is $2^{p(n)+1}$. □