Assume P = NP. SAT is in TIME(nk). Let L be in NP, we can reduce L to SAT, so NP = TIME(nk) for some k, but TIME(nk) is a strict subset of TIME(nk+1). What is wrong?

2) Show USAT = a boolean formula has exactly one satisfying assignment is in PSAT.

Solution: Take the input w. Create “w AND x1”. As the oracle if this can be satisfied. If so, we know that x1 can be set true.

Create “w AND (NOT x1)” and as the oracle if this can be satisfied. If so, then we can set x1 both true and false so there is more than one satisfying assignment. If only one these can be satisfied, we know how x1 must be set, so repeat for each variable.

3) Find an oracle A such NPA <> coNPA.

L = {x | for all a with |a| = |x| a is in A}

Idea, is the NP machine will accept if there is some branch in its choices that leads to an accept state. CoNP machine will accept if every branch leads to the accept state.

No NP oracle machine will be able to decide L. Order the NP oracle machines: M1, M2, ….

Build A iteratively. Consider machine Mk and strings of length n where n is larger than the strings we have used for previous machines.

Mk on an input of length n, every time it makes a query of A, A accepts (that string is in A).

Mk can only make a polynomial number of queries,

If Mk accepts, we find some string it did not query, and make that string not be in A.

If Mk rejects, there is no path in Mk where it asks a query of length n of A where Mk  accepts. We decide that all strings of length n are in A.

In either case, Mk cannot decide L.

The coNP machine can decide L. We have the nondeterministic choices (branches) cover all possible strings of length n.

Interactive Proofs

Generalizes the idea of an oracle machine, but now when we make queries of the machine, we can’t trust the answer. The machine we query might be lying.

We assume that our machine runs in polynomial time (or some other restriction like logspace). The machine we query we assume is more powerful.

Homework: If our machine is deterministic, polynomial time, the the best we can do with an interactive proof system is to decide languages in NP.

Our machine (called the verifier) will be random. We can think of the machine either as having the ability to flip a random coin. Or it can be a deterministic machine that takes the input the problem and a string of pre-flipped random bits. (like a certificate)

BPP. L is in BPP if a, for a random polynomial machine, if x is in L, the machine accepts with probability >= 2/3, If x is not in L, the machine accepts with probability <= 1/3.

RP. L is in RP if, for a random polynomial machine, if x is in L, the machine accepts with probability >= ½. If x is not in L, the machine accepts with probability 0.

RP is a subset of BPP.

Consider L in BPP, run the machine 3 times, what is the accept probability if x is in L? (Take the majority vote.)

20/27 = (3 yes + 2 yes) = (2/3)3  + 3 (22/ 33) > 2/3

Consider L in RP, run the machine 3 times, what is the accept probability?

7/8

We can make the accept probability exponentially high by re running the machine a polynomial number of times.

IP: We have a BPP verifier. The “prover” the machine we are querying is all powerful (some definitions restrict it to PSPACE).

If x is in L, then there exists a prover such that our verifier will accept with probability >= 2/3.

If x is not in L, then for all “provers” our verifier will accept with probability < 1/3.

Example: NONISO = Given two graphs, are the graphs non-isomorphic to each other. (We are assuming that we can’t determine if 2 graphs are isomorphic in polynomial time.)

The verifier chooses one of the two graphs at random, make an isomorphic copy of it (permuting the names of the vertices). It asks the prover which one it chose. If the prover always answers correctly, we increase the probability that the prover can correctly determine NONISO by ½.