Zero knowledge proofs (interactive proof).

We have polynomial time, random verifier that can ask questions of a prover.

We want the prover to convince the verifier but without revealing the proof (or the certificate).

Hamiltonian Cycle: (Given a graph, does it have a simple cycle that hits every vertex.)

Required assumption is that there exists “one-way functions”. A function that is easy to compute but hard to reverse.

Ex: assume that NP-complete problems are hard to compute (in the worst case)

Ex: assume that multiplication is a one-way function – factoring a large number is hard to do

Ex: determining if two graphs are isomorphic is hard to do

Verifier gives a graph G to the prover.

The prover finds a hamiltonian cycle on G.

The prover will do a random permutation on the vertices of G (call this graph H).

The prover sends H to the verifier.

The verifier can either ask for the Hamiltonian cycle on H or can ask for the function that maps the vertices of G to H. The verifier will choose at random.

We repeat this process k times. That gives us high confidence that the prover is right. O(2-k).

If the verifier can’t compute the isomorphism between G and H, the verifier can’t find the Hamiltonian cycle in G.

A bunch of classes of languages:

regular languages < c**ontext free languages**

**regular languages** <= *L <= NL* <= **P <= NP** <= Sigmak <= PH <= *PSPACE* = *NPSPACE* = IP <= EXPTIME <= NEXPTIME <= **Turing-decidable < Turing-recognizable** < All languages

L = (undirected PATH \*)

NL = (directed PATH)

NP = (3-SAT, Vertex Cover, Clique, Ham Cycle)

PSPACE = (TQBF)

Turing-recognizable = (ATM, halting problem)

coNL = NL

coNP (not known to equal NP) (complement of 3-SAT)

nondeterminism (we can guess, we guess correctly as long as there is an accept state to go toward)

Techniques:

pumping lemma (used as a contradiction proof for regular languages and context free languages)

reductions (computability, polynomial time, logspace)

Turing machines (enumerators, enhancements

push down automata = context free grammars (non-deterministic)

finite state automata = regular expressions

complete problems for a class of problems

Additional topics:

recursion theorem

Kolmogorov complexity

oracle machines

interactive proofs

zero knowledge proofs

randomized computation (BPP, RP)

finite state automata = nondeterministic FSA.

→ nondeterminism always tries to get to the accept state

We need to make sure when creating a nondeterministic machine, that it is impossible to accept if the input is not in the language.

A deterministic pushdown autamata != non-determinsitic one. We almost always write a non-deterministic machine, it is equal to the context free languages.

(Look ahead k languages were deterministic ones)

What about quantum computers?

A normal bit is a 0 or a 1.

A quantum bit is both 0 and 1, it “collapses” to one of those values with a certain “probability” when its observed.

We believe that quantum computers can solve more than the class P. But we believe that NP-complete problems still require exponential time on a quantum computer.

How do we prove P != NP?

It appears that we can’t use Turing machines. (We have oracles that show PA = NPA and PB != NPB.)

One idea is the use “circuits” for computation. Input gates, and/or/not/xor gates, and output gates. No cycle in the circuit. Instead of running time, we ask how many layers does our circuit have? How many inputs to each gate to we need?

(Nonuniform complexity: we need a different circuit for each input size.)