CIS 511: Theory of Computation **Feb 16, 2017**

Lecture 11: Even More Time Complexity

Professor Sampath Kannan *Zach Schutzman*

NB: *These notes are from CIS511 at Penn. The course followed Michael Sipser's Introduction to the Theory of Computation (3ed) text.*

More About NP

Recall, a language is in NP if it can be decided by a non-deterministic Turing machine in polynomial time.

Recall the language $SAT = \{\phi | \phi \text{ is satisfiable}\}$. If ϕ is satisfiable, then there exists some satisfying assignment, and if an oracle gave you some satisfying assignment, you could very easily use this to verify that ϕ is satisfiable (just plug in).

Definition 11.1 *Alternatively, a language L is in NP if there is a polynomial time deterministic Turing machine V such that $L = \{x | \exists y : V \text{ accepts } (x, y)\}$. In English, V is a verifier and y a certificate. NP is the class of languages which, given a certificate, a verifier takes a language and a solution and decides whether x belongs to L. y is a 'proof' of x being in L.*

Some more example of languages in NP:

- *A clique is a set of vertices in a graph which are all pairwise adjacent.*

  The language $CLIQUE = \{\langle G, k\rangle | G \text{ has a } k\text{-clique}\}$ is in NP. We can see this alternatively by thinking about an NTM that guesses which nodes are in the clique or by considering a certificate, we can check whether the provided vertices actually form a clique by verifying that each pair of vertices are actually pairwise adjacent.

Claim 11.2 *The two definitions for NP are equivalent.*

Proof: *Let M be an NTM that recognizes L in polynomial time. Therefore, there is some accepting path in the branching tree, with path length polynomial in the input length. Specifying this path is an encoding of the address of the leaf at the end of the path. Since the height of the path is at most polynomial in the length of the input, we can encode this address in polynomial length. A verifier can then simulate M only on that branch, which decides in polynomial time.*

We can see the other direction by having the NTM non-deterministically guess the certificate. Since this is polynomial in length, it can be done in polynomial depth. ■

NP-Complete Languages

NP contains many problems that are important in practical optimization and decision problems. We would love to be able to solve problems in NP in polynomial time. The big question in computer science is "IS $P = NP$?"

NP-Complete languages are in a sense the hardest languages in NP. If a language NP-Complete and there is a polynomial time algorithm for L, then $P = NP$.

Definition 11.3 *A language is called NP-Complete if it is in NP and there exists a polynomial time mapping reduction from any language in NP to it.*

To find NP-Complete languages, we need to identify one first. SAT was the first proven NP-Complete language (Cook-Levin Theorem). Then, if A is NP-Complete and A mapping reduces to B in polynomial time, then B is called NP-Hard. If we also have that B is in NP, then B is NP-Complete.

Definition 11.4 *A language B is **NP-Hard** if there is a polynomial time mapping reduction from a language $A \in NP$ to B*

Proof: *A sketch of the proof of Cook-Levin ($SAT \in NP$-Complete):*

We've already seen that SAT is in NP.

Let L be any language in NP and V be the polynomial time verifier for L. View the computation of V as a sequence of configurations (snapshots of tape and state after each transition). We want to construct a mapping reduction that transforms an input x to V with into a boolean formula ϕ that is satisfiable if and only if x is in L.

Call the certificate of L y. We don't know y, but we can think about its bits as unknown boolean variables $y_1, y_2 \ldots$. The initial configuration of V looks like $q_o x \# y_1 y_2 \ldots$. We can say that x is in L if we end up in q_{accept}.

We need to make sure that V can't be fooled. We need to be sure that w is a valid input and that if we are given a sequence of configurations, that sequence must be legal according to the transition function of V.

V runs in polynomial time, and its input is polynomial in length and for polynomially many steps. Bound the length of the input by l and the number of steps taken by l (one of these can be a loose upper bound). Let's make boolean variables $x_{ij\sigma}$ where $i, j \leq l$ and $\sigma \in \Gamma \cup Q$. Set $x_{ij\sigma}$ equal to 1 if cell i at step j contains σ and 0 otherwise.

We can say that $x \in L$ if and only if there exists an accepting table of V on w, where an accepting table refers to the legal sequence of tape configurations ending in an accept state. We want to create a formula which is satisfiable if and only if there is an accepting table of V on w.

We will build the corresponding ϕ:

These conditions define a legal beginning and end configuration:

$\bigwedge_{i,j} \bigvee_{\sigma} x_{ij\sigma}$ *- at least one symbol in each cell*

$\bigwedge_{\sigma_1,\sigma_2} \neg x_{ij\sigma_1} \vee \neg x_{ij\sigma_2}$ *- at most one symbol in each cell*

$\bigwedge(x_{11q_0}) \bigwedge(x_{12w_1}) \ldots$ *- the input row contains q_0 and the correct value for the input w*

$\bigwedge(x_{l1q_{accept}})$ *- V takes exactly l steps and brings head to leftmost cell*

We need to make sure the steps are consistent with legal transitions:

We also AND some variables constructed from the transition function to ensure the value at cell i, j is either the same, or influenced by the ones just left or right at the previous step

If this formula is satisfiable, then we have a legal table, hence w is in L. If there is no table, then the formula *cannot be satisfiable.* ■