CS 810: Introduction to Complexity Theory 02/20/03

Lecture 10: The Karp-Lipton Theorem

Instructor: Jin-Yi Cai Scribe: Christopher Hudzik, Sarah Knoop, Louis Kruger

We continue to probe the question of P=NP using boolean circuits. The following theorem (Theorem 1) presents some evidence that NP does not have polynomial size circuits. The proof is based a combination of techniques that we have seen: self-reduction of SAT (NP-complete languages) and a "guess and validation".

1 Karp-Lipton Theorem

Theorem 1 (Karp-Lipton Theorem). *If NP is contained in P/poly, then the polynomial hierarchy collapses to the second level, i.e.* $\Sigma_2^P = \Pi_2^P$.

Proof. We'll show that if SAT is contained in P/*poly*, then Π_2^p is contained in Σ_2^p. Note that, $\Pi_2^p \subseteq \Sigma_2^p$ implies PH $= \Sigma_2^p = \Pi_2^p$ (see Theorem 1 of Lecture 6).
Suppose SAT is in P/*poly*. Let L be any language in of Π_2^p. Then there exists a deterministic, polynomial-time predicate D and a polynomial p s.t.

$$x \in L \Longleftrightarrow \forall^p y, \exists^p z[D(x, y, z) = 1] \tag{1}$$

"$\exists^p z$ s.t. $D(x, y, z) = 1$" is an NP question. Thus, it can be reduced to an instance of SAT, $\Psi_{x,y}$ (a boolean formula dependent on x and y), using Cook's reduction. We can then reformulate as follows:

$$(\exists^{\mathrm{P}} z)[D(x, y, z) = 1] \Longleftrightarrow \Psi_{x,y} \text{is satisfiable.}$$

Now equation (1) can be restated in terms of this SAT question:

$$x \in L \Longleftrightarrow \forall^P y, \Psi_{x,y} \in SAT$$

where $|\Psi_{x,y}|$ is polynomially bounded in $|x|$.

We construct a Σ_2^{P} computation that simulates the Π_2^{P} computation of L. Let x be the input into our simulation. By assumption, there exists a polynomial-sized circuit that accurately decides SAT; the problem is, it'd take to long to search for it. The idea is that our Σ_2^{P} simulation will instead guess a polynomial-sized circuit, C, that may or may not accurately decide SAT. Now we set the "trap". We transform C into another circuit, C', that uses the circuit C as an "oracle" which not only decides if a boolean formula is satisfiable, but produces a satisfying assignment if it is so. In reality, our Σ_2^{P} simulation will guess C' initially. This is possible because SAT in P/*poly* implies that the SAT searching problem is also in P/*poly*. Thus, the SAT searching problem has a polynomial-sized circuit that can be guessed in polynomial time. Now, our simulation enters a universal state and constructs $\Psi_{x,y}$ such that $\forall^{\mathrm{P}} y, \Psi_{x,y}$ is polynomially bounded in $|x|$. The simulation then evaluates $C'(\Psi_{x,y})$. If C' decides that $\Psi_{x,y}$ is not satisfiable, our simulation rejects. If C' decides that $\Psi_{x,y}$ is satisfiable it goes ahead and computes (via self-reduction) a satisfying assignment, σ, for $\Psi_{x,y}$. The simulation, in this case, then checks the produced assignment. If $\Psi_{x,y}(\sigma) = F$, we had a bad guess of C, so reject. Otherwise, we accept.

Now if the simulation made an incorrect choice of C, then C' would also be incorrect. However, C' may incorrectly conclude that a satisfiable formula is unsatisfiable, but it will never give a false positive; saying an unsatisfiable formula is satisfiable. Consequently, if $x \notin L$, then there is some y for which $\Psi_{x,y}$ is not satisfiable. So for any guess C, the corresponding C' would always say "no" since it never gives false positives.
Conversely, if $x \in L$, then there is a circuit C of polynomial size that decides SAT (i.e. gives a C' that produces a satisfying assignment for $\Psi_{x,y}, \forall^{\mathrm{P}} y$). Our simulation will always guess this circuit and, thus, have an accepting path.

Hence, $\Pi_2^{\mathrm{P}} \subseteq \Sigma_2^{\mathrm{P}}$. □

2 Reduction Types

We have discussed two different type of reductions: Turing reductions and many-to-one reductions. As a review, we will define these again.

Definition 1 (Polynomial-Time Turing Reducible). *For any two sets A and B, A is* polynomial-time Turing reducible *to B, written $A \leq_T^P B$, if and only if there exists an polynomial-time oracle TM M, such that $A = L(M^B)$. A polynomial-time Turing reduction is a reduction via some polynomial-time oracle TM. This type of reducibility is also called Cook reducibility.*

Definition 2 (Polynomial-Time Many-To-One Reducible). *For any two sets A and B, A is* polynomial-time many-to-one reducible *to B, written $A \leq_m^P$, if there exists a polynomial-time computable function f such that $x \in A$ if and only if $f(x) \in B$. A polynomial-time many-to-one reduction is a reduction via some polynomial-time computable function. This type of reducibility is also called Karp reducibility.*

Turing reductions are "stronger" reductions than many-to-one reductions. They are stronger in the sense that if A is polynomial time many-one reducible to B, then A is also polynomial time Turing reducible to B. But the converse is believed to be false. Thus, Turing reductions yield weaker hypotheses than do many-to-one reductions. In the Karp-Lipton Theorem (Theorem 1), the hypothesis alludes to the question, "What happens $\mathrm{SAT} \leq_T^{\mathrm{P}} S$, a sparse set." This results in the polynomial time hierarchy collapsing to the second level. In the Mahaney Theorem below (Theorem 2), we see the results of SAT being many-to-one reducible to a sparse set.

Theorem 2 (Mahaney Theorem). *$SAT \leq_m^P S$, a sparse set if and only if NP=P.*