COM S 6810 Theory of Computing January 22, 2009

Lecture 2: Diagonalization

Instructor: Rafael Pass *Scribe: Jesse Simons*

This lecture will cover three proofs using different forms of diagonalization. You most likely have seen diagonalization previously in proving that there are more reals than integers or that the halting problem is impossible to compute. Before we begin, we'll state, but not prove, a useful lemma and corollary to the lemma.

Lemma 1 (Universal Simulation) $\exists$ *Turing Machine, U such that* $\forall x, i \in \{0, L\}^*$, $U(x, i) = M_i(x)$ *where* M_i *denotes the Turing Machine represented by string* i*. Furthermore, if* $M_i(x)$ *halts within* t *steps, then* $U(x, i)$ *halts within* $O(t \log t)$ *steps.*

Corollary 1 *Given a description* $\langle M \rangle$*, of a Turing Machine* M*, a string* x*, and an integer* $t > n$ *where* $n = |\langle M \rangle| + |x| + |t|$*, the problem of deciding whether* $M(x) = 1$ *within* t *steps can be solved in* $O(t \log t)$ *steps.*

Next, we begin the first of our three diagonalization proofs.

Theorem 2 $DTIME(n \log^2 n) \not\leq DTIME(n)$

Proof.Consider the language, L, which is decided by the decision process D. Define $D(\langle M \rangle)$: if $M(\langle M \rangle)$ accepts within $n \log n$ steps, reject; otherwise accept.
Claim 1: $L \in DTIME(n \log^2 n)$. This follows directly from Corollary 1.
Now, assume for contradiction that there exists a machine M_i that runs in time cn that decides language L. Without loss of generality assume $|M_i| \log(|M_i|) > c|M_i|$. This is WLOG, for we can just pad M_i with zeros. Then by definition $M_i(\langle M_i \rangle)$ rejects iff $M_i(\langle M_i \rangle)$ accepts. $\rightarrow\leftarrow$ ■

Observe that the deterministic running times $n \log^2 n$ and n are not specific to the proof. We could have used most any other pair to derive a similar result. Therefore, this Theorem 2 implies that more deterministic time provides more power, or rephrased, the ability to use more deteterministic time allows for the ability to decide some problems that couldn't be decided with less.

Can we prove an analagous theorem for nondeterministic time using an analagous proof? No, Claim 1 no longer holds; we can not simulate a non-determistic Turing Machine in $NTIME(n \log^2 n)$ or even in polynomial $NTIME$. Instead, we'll utilize a modified technique called lazy diagonalization to prove the non-deterministic case. But first a lemma most likely proven to you in an undergraduate theory of compuation course:

Lemma 3 *If we have a nondeterministic Turing Machine M with decription $\langle M \rangle$ that runs wihtin $t > n$ steps in accepting some language L, we can simulate M with a deterministic machine that accepts the same language in $O(2^t)$ steps.*

Theorem 4 $NTIME(n \log^2 n) \not\leq NTIME(n)$

Proof. First consider the following function over the natural numbers:

$$f(1) = 2 \quad f(k+1) = 2^{kf(k)}$$

Next, consider the language L defined by the following decision process.
$D(x = \langle M \rangle, \mathbf{1}^t)$: For some natural number k
Case 1: If $f(k-1) < t < f(k)$, Accept iff $M(\langle M \rangle, \mathbf{1}^{t+1})$ accepts within $n \log n$ steps.
Case 2: If $t = f(k)$, reject iff determinsitic simulation of $M(\langle M \rangle, \mathbf{1}^{f(k-1)+1})$ accepts within $t = 2^{kf(k-1)}$ steps.

Notice that since we can easily solve for k, L can be decided in $NTIME(n \log^2 n)$.

Claim: $L \notin NTIME(n)$
Assume for contradiction that there exists a nondeterministic Turing Machine T that decides L in cn steps. For $f(k-1) < t < f(k)$ such that $t \log t > cn$ since T behaves like D we know:

$$T(\langle T \rangle, \mathbf{1}^t) = T(\langle T \rangle, \mathbf{1}^{t+1})$$
$$\Rightarrow T(\langle T \rangle, \mathbf{1}^{f(k-1)+1}) = T(\langle T \rangle, \mathbf{1}^{f(k)})$$

But since $2^{kf(k-1)} > 2^{O(f(k-1))}$ for large enough k, we know by Lemma 3 the deterministic simulation of $T(< T >, 1^{f(k)})$ does not get cut off in Case 2 of decision process D. Then:

$$T(\langle T \rangle, \mathbf{1}^{f(k)}) = 1 - T(\langle T \rangle, \mathbf{1}^{f(k-1)+1})$$

We've now shown that our assumption causes these two terms to be both equal and unequal.$\rightarrow\leftarrow$ ■

We will begin discussing a third proof that should be completed more formally in the next lecture. Note that for some common problems, it is still an open question as to whether they belong to P,NP-complete, or something inbetween. Both graph-isomorphism and factorization (and until recently primality) are prime examples of such problems. Now, consider the following theorem:

Theorem 5 (Ladner's Theorem) *If $P \neq NP$, $\exists$ a decision problem (and thus also a language) A such that $A \in NP$, A is not NP-Complete, and not $A \notin P$*

We can prove this by starting with a classic NP-complete problem and "shooting holes" in it. Consider the following language that uses a soon to be specified function f:

$$L = \{x | x \in SAT \wedge f(|x|) \text{ is even}\}$$

Observe that if f is a function such that $\exists_y \forall_{x \geq y} f(x) = c$, then if c is odd $L \in P$, and if c is even L is NP-complete. Obviously, if c is odd then only finitely many instances of SAT are included in L and each can be checked for in polynomial time. Otherwise, only finitely many instances are excluded from classic SAT and the problem remains NP-complete.

If we can construct a f such that:
$L \in P$: then f gets stuck on an even $c \Rightarrow NP = P$
L is NP-complete then f gets stuck on an odd $c \Rightarrow NP = P$
In both cases, we'd conclude the contradictory claim $NP = P$, and thus L must not be in P of NP-complete.

We can accomplish this by having the output of f descibe a machine M and subscribing to the following methodology. To show not in P, we only increase the output if M fails to decide L. On the other hand, to show not in NP, increase number if M is not a good reduction.

Let $M_1, M_2, ...$ be an enumeration of Turing Machines such that $M_i(x)$ runs for at most $|x|^i$ steps. Let $g_1, g_2, ...$ be an enumeration of all polytime reductions (which too can be represented as Turing Machines).

$f(n)$:
Case 1: If $f(n-1) = 2i$: If $\exists_x$ s.t. $|x| < \log n$ such that $M_i(x) \neq L(x)$ then $f(n) = f(n-1) + 1$, otherwise $f(n) = f(n-1)$
Case 2: If $f(n-1) = 2i + 1$: If $\exists_x$ s.t. $|x| < \log n$ such that $SAT(x) \neq L(g_i(x))$ then $f(n) = f(n-1) + 1$, otherwise $f(n) = f(n-1)$

If $L \in P$ then there is a polynomial Turing Machine that accepts exactly L, in which case f will get stuck in the else clause of Case 1, an even number, which makes L NP-complete. If instead L is NP-complete, there is a reduction g_i from SAT to L, in which case f will get stuck in the else clause of Case 2, an odd number, which means $L \in P$.