CSE200 Lecture Notes
Diagonalization

Lecture by Russell Impagliazzo
Notes by William Matthews

Lecture April 12, 2010

1 Diagonalization

Claim 1 (Cantor)**.** *There are more sets of positive integers than positive integers.*

Proof. Both of these sets are infinite, how can we argue that one is bigger than the other? Two sets A and B are the same size if and only if there exists a bijection (a 1-to-1 function) between them. We will take A to be the set of positive integers, and B to be the set of sets of positive integers.

Assume, for the sake of contradiction, that there exists a bijection $f : A \to B$. We will construct the following matrix $M[i, j]$ where $M[i, j] = 1$ iff $i \in f(j)$.

$$M = \begin{array}{c|cccc} & f(1) & f(2) & f(3) & \dots \\ \hline 1 & 1 & 0 & 1 & \dots \\ 2 & 0 & 0 & 0 & \dots \\ 3 & 1 & 1 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

Consider the set $S = \{i \mid M[i, i] = 0\} = \{i \mid i \notin f(i)\}$. Since the function f is a bijection then there must exist a j such that $f(j) = S$. By the definition of S, $j \in S$ if and only if $j \notin f(j)$ however $S = f(j)$ which gives a contradiction. Thus f cannot exist.

We conclude by noting that the set of sets of positive integers must be larger than the set of integers since there exists a an injective mapping from integers to sets of integers by mapping each integer x to $\{x\}$. $\square$

1.1 From integers to Turing machines

Since each Turing machine has a finite description, we can map Turing machines to positive integers. Similarly since strings have finite lengths, we can also map strings to integers. A language is just a set of strings, which we may also view as a set of integers. By a similar argument, there are more languages than Turing machines. Thus, there exist languages for which no Turing machine decides them (languages which are not recursive).

Similarly to what we did before, construct the following matrix $M[i, j]$ where $M[i, j] = 1$ iff $i \in L(M_j)$ i.e. M_j halts and accepts on input i, where i is viewed as the string corresponding to integer i, and M_j is the Turing machine corresponding to integer j.

Define the language $\mathcal{L}_{diag} = \{i \mid M[i][i] = 0\} = \{i \mid i \notin L(M_i)\}$. Suppose for the sake of contradiction that there existed an integer j such that M_j decided $\mathcal{L}_{diag}$. Then M_j halts and accepts $j \iff j \in \mathcal{L}_{diag} \iff j \notin L(M_j)$ i.e. M_j does not accept j. Which is a contradiction, and therefore $\mathcal{L}_{diag} \notin REC$.

Since $\mathcal{L}_{diag}$ is somewhat contrived, can we come up with a more natural language that is not in REC? Define the language $HALT = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$. $HALT \notin REC$. Proof: Suppose, for the sake of contradiction that $HALT \in REC$. Thus, there exists a Turing machine M_{HALT}

which decides $HALT$. We may use M_{HALT} to construct a Turing machine which decides $\mathcal{L}_{diag}$: On input i, run M_{HALT} on $\langle M_i, i\rangle$. If M_{HALT} rejects then accept. Otherwise, run M_i on input i. If M_i accepts then reject, otherwise accept. First, we claim the this TM always halts. Since by assumption M_{HALT} decides $HALT$, we know that it always halts. We only run M_i on input i when $\langle M_i, i\rangle \in HALT$, so we know that M_i will halt. Second, we argue that our TM decides $\mathcal{L}_{diag}$. Each input i is accepted if and only if either M_i doesn't halt on i or M_i halts and rejects i. This corresponds exactly to $i \notin L(M_i)$, and therefore $i \in \mathcal{L}_{diag}$.

1.2 Time bounded Turing machines

For any "nice" time function $T(n)$ (non-decreasing, $T(n) \geq n$, $T(n)$ can be computed in time $T(n)$), there exists a language that is not decidable in time $T(n)$, but is decidable in time $poly(T(n))$. Define the language $\mathcal{LC}_M^{T(n)} = \{x \mid M \text{ is a TM and } M \text{ accepts } x \text{ in at most } T(|x|) \text{ steps}\}$. Using $\mathcal{LC}_M^{T(n)}$, define the language $\mathcal{L}_{diag}^{T(n)} = \{i \mid i \notin \mathcal{LC}_{M_i}^{T(n)}\}$. By a similar diagonalization argument, we can conclude that no Turing machine that runs in time at most $T(n)$ can decide $\mathcal{L}_{diag}^{T(n)}$. However we can construct a TM that runs in time $poly(T(n))$ which does decide $\mathcal{L}_{diag}^{T(n)}$, by simulating the input machine while keeping track of the number of steps. If we ever run for too many steps, then reject.