Midterm Solutions
Computational Complexity

PROBLEM: A language L is called *unary* if $L \subseteq 1^*$. Show that if a unary language is NP-complete, then $P = NP$.

SOLUTION: Suppose a unary language L is NP-complete. That is, there exists a polynomial time reduction f that when given a SAT formula ϕ as input produces $f(x)$ as output where $\phi \in SAT$ iff $f(x) \in L$. We will show how we can solve SAT in polynomial time.

Suppose we want to determine if a SAT instance ϕ is satisfiable. We say ϕ' is restriction of ϕ if ϕ' can be obtained by setting some variables of ϕ to $True$ or $False$. Note that the restrictions ϕ' of ϕ map to at most $poly(|\phi|)$ strings under the mapping f (since the length of the mapped strings is bouded by a polynomial in $|\phi|$). The idea is to remember which of these mappings $f(\phi')$ we have already determined are not in L. For this we create a boolean array $NotInL[\,]$ with polynomially many entries, one for each possible $f(\phi')$, and initialize the array to $False$ (changing $NotInL[l]$ to $True$ will indicate we have determined $1^l \notin L$). We then call the $CheckSatisfiability(\phi)$.

FUNCTION $CheckSatisfiability(\phi')$
if *ϕ' has no variables in it* **then**
  Return $True$ if ϕ' is satisfiable. Otherwise set $NotInL[f(\phi')]$ to $True$ and return $False$.
else
  Pick a variable x in ϕ'.
  `/*Try both assignments to` x `and see if either leads to a satisfiable expression */`
  if $NotInL[f(\phi'_{x=True})]] = False$ **then**
    Return $True$ if $CheckSatisfiability(\phi'_{x=True})$ returns $True$.
  if $NotInL[f(\phi'_{x=False})]] = False$ **then**
    Return $True$ if $CheckSatisfiability(\phi'_{x=False})$ returns $True$.
  `/*Neither assignment worked */`
  Set $NotInL[f(\phi')]$ to $True$ and return $False$.

Make a "recursion tree" of calls of $CheckSatisfiability$ (The leaf is the first call $CheckSatisfiability(\phi)$. A node has leaves corrsopnding to each call to $CheckSatisfiability$ it makes.) It is easy to check that all calls to $CheckSatisfiability$ where the argument ϕ' maps to a given unary string 1^l are along one path from the root to a leaf (because, when the first call to $CheckSatisfiability$ with such a ϕ' returns, we have set $NotInL[l]$ to $True$). Therefore, we make only $O(|\phi|)$ calls for each l. This gives the a polynomial bound on the running time. $\square$

PROBLEM: For any positive integer k, show that there is a language in PH with circuit-complexity $\Omega(n^k)$. In fact you can exhibit such a language in some fixed finite level of PH, say Σ_5.

SOLUTION: First note that there are functions on $m = (k+1)\log n$ bits with circuit size $\frac{n^{k+1}}{(k+1)\log n} > n^k$ for large enough n. Such a function can be described by its truth table which is of size $n^{k+1} = poly(n)$. For each m we order the truth tables lexicographically and define f to be the first function in this order that has circuit size greater than n^k. We can think of f as a function on n bits where we ignore all except the first m bits.

To show that the language accepted by this function is in PH, we will do the following: On an input x of size n

- Guess the truth table for f. $(\exists f)$.
- Check it has large circuit size. This is done by

$$\forall \text{ circuits } C \text{ s.t. } |C| \leq n^k, \exists y \in \{0,1\}^m \text{ s.t. } C(y) \neq f(y)$$

- Check that every lex. smaller function g on $\{0,1\}^m$ does have small circuits.

$$\forall \text{ functions } g < f, \exists \text{ circuit } C' \text{ s.t. } |C'| \leq n^k, \forall y \in \{0,1\}^m\ C(y) = g(y)$$

This puts the language in Σ_4 (in fact Σ_3 since the set $\{0,1\}^m$ is of polynomial size). Using the Karp-Lipton theorem one can show that such a language exists in Σ_2. $\square$

PROBLEM: Let BPL be the class of languages accepted by a polynomial time randomized logspace machine with two sided error. Show that $BPL \subseteq P$.

SOLUTION: Define a configuration by contents of work tape, position of the input head and state of the machine. Observe that the total number of configurations is bounded by n^k for an input of size n. Define a transition matrix $P = p_{ij}$ where p_{ij} is the probability of going from config i to j. (This matrix depends on the input). Let π_0 be a vector with 1 for the start configuration. Then $P^t\pi_0$ is the distribution after t steps. To check if the machine accepts in time t, check that the prob of being in the accept state is at least 0.9. The matrix P^t can be computed efficiently by matrix multiplication. $\square$