Third Week

Instructor: Prof. S. P. Pal
TA: Rahul Gokhale

We know that languages accepted by space bounded machines can be accepted by space bounded machines that halt on all inputs. Now we wish to return to nondeterministic space bounded machines and see how we can eliminate nondeterminism– naturally the space complexity would go up. In this context, we consider the proof technique as in the proof of Savitch's theorem.

Exercise 1 *Show that the problem of deciding the acceptance of an input string w on an $S(n)$-space bounded machine M is equivalent to deciding whether a directed graph $G_{w,M}(V,E)$ has a path from vertex $I_0 \in V$ to a vertex $I_A \in V$; here, I_0 and I_A correspond to starting and accepting configurations of machine M respectively and V is the set of all possible configurations of M on input w. Also, E is the set of all configuration transitions possible on M due to input w. Show that M is deterministic if and only if the outdegree of every vertex in V in the graph $G_{w,M}$ is 1.*

Exercise 2 *How is space reused in the two recursive calls (in the proof of Savitch's theorem) at the same level of recursion? What is the additional space required on a recursive call and what is the maximum depth of recursion?*

Exercise 3 *Analyze the running time complexity of the deterministic simulation as in Savitch's theorem. Is the whole graph $G_{w,M}$ stored in the simulating machine? If not, how is the set E of transitions simulated?*

Exercise 4 *If L is the language accepted by a nondeterministic $S(n)$-space bounded machine then how much space is required to accept the complement of L with a deterministic space bounded machine? Why?*

After studying the tape compression theorem, we now wish to determine how much more space is required so that newer languages can be accepted. In this context we consider the space hierarchy theorem as in [**HU79**].

Exercise 5 *Study the proof of the halting problem's undecibability and partial decibability and its diagonalization proof technique. What does it mean to say that a certain language is recursive, recursively enumerable and not recursively enumerable. Study section 8.3 thoroughly from [**HU79**]. Study Lemma 8.1, Theorem 8.4 and Theorem 8.5. How are these results related to the halting problem?*

Exercise 6 *What is the encoding of TMs in the proof Theorem 12.8? Why is it that an arbitrary length prefix of 1's is attached to encodings of TMs as defined in Chapter 8? How is the simulating machine forced to use only $S2(n)$ space?*

Exercise 7 *Argue that M simulates machine M_w on input w in* ***DSPACE(S2(n))****. Why would M_w's tape symbol set cardinality t determine the amount of space $\lceil logt \rceil$ times $S1(n)$ required for M to simulate M_w on input w.*

Exercise 8 *Assume (for the sake of contradiction) that $L(M) = L(\hat{M})$ where $\hat{M}$ is an $S1(n)$-space bounded TM with t tape symbols. Show that there is a w of such length n that $\lceil logt \rceil$ times $S1(n)$ is dominated by $S2(n)$, and M_w is $\hat{M}$. Show that the way M acts (in terms of accepting or rejecting, in its simulation of M_w on input w), it follows that $L(M)$ is not equal to $L(M_w)$ (and therefore to $L(\hat{M})$) for such a w as in the previous sentence. In particular, show that $L(M)$ and $L(M_w)$ differ on how they act on w of sufficient length n as required above. Conclude therefore that $\hat{M}$ being in* ***DSPACE(S1(n))*** *is impossible.*

For the case of time hierarchy, we may refer to the similar diagonalization result of Theorem 12.9 in [HU79]. Here, the ratio $\frac{T_1(n)\log_2 T_1(n)}{T_2(n)}$ can exceed any chosen value $c > 0$, for sufficiently large $n > n_c$ where n_c depends on c. In fact, the simulation of a multitape $T_1(n)$ time bounded machine is well nigh possible in $cT_1(n)\log_2 T_1(n)$ time (see Theorem 12.6 [HU79]) for some constant $c > 0$. So, a sufficiently long string w of length n can always be found out to satisfy a suitable c, so that $T_2(n)$ suffices in simulating the $T_1(n)$ time bounded machine M_w on input w of such bloated length, thereby helping realization of the diagonalization conveniently. The contradiction resulting from the simulation establishes the absurdity of the assumption that the simulating machine's language $L(M)$ is also $L(\hat{M})$, where $\hat{M}$ is a $T_1(n)$ time bounded machine.

We have also seen a different result, very reminiscent of the halting problem for Turing machines where we showed that a certain language H_f=$\{M; x — M$ accepts input string x in less than or equal to $f(|x|)$ steps$\}$, is in $DTIME(f^3(n))$, but not in $DTIME(f(\lfloor \frac{n}{2} \rfloor))$. For the former claim see [Papa94].

Exercise 9 *Show that $H_f \in DTIME(f^3(n))$.*

We outline only the latter claim sketching the diagonalization argument. We assume for the sake of contradiction that $H_f \in DTIME(f(\lfloor \frac{n}{2} \rfloor))$. So, there is a deterministic machine M_{H_f} that decides H_f in time $\lfloor \frac{n}{2} \rfloor$. Define machine D_f such that D_f on input M says 'yes' if and only if M_{H_f} syas no on input $M; M$.

Exercise 10 *Show that D_f decides in $f(n)$ time. Also show that $D_f(D_f)$ is 'yes' if and only if it is 'no' as well ! In other words, establish a contradiction.*

Now consider problem of non-deterministically computing the cardinality of the set of vertices reachable from a source vertex s in a directed graph $G(V, E)$ in space proportional to $\log_2 |V|$. Here $|V| = n$. We only elaborate on the subtle steps exploiting space reuse and nondeterminism. Only a constant number of index variables of length $\log_2 n$ are used in the computation. Also note that only the last step uses nondeterminism. The crucial step is the penultimate level where a conjunction over an iterator is computed for determining whether a certain vertex u is in $S(k)$; this is done by checking for all vertices v whether (i) $v \in S(k-1)$, and whether (ii) $u = v$, or u is directly reachable from v. If $u \in S(k)$ then for some v this must hold. However, if we are trying to determine whether this $v \in S(k-1)$, using only a nondeterminsitic guessing method, then the path of computation making a wrong guess would fail to verify that v is indeed in $S(k-1)$. So, the way we resolve this problem is by running the for loop over all v and keeping the count of successes where we actually get certified that a vertex v is indeed discovered to be in $S(k-1)$. If this count matches $|S(k-1)|$, which is alreadycomputed and stored in a counter, then we succeed. Otherwise, we reject the entire computation in the for loop. The correctness follows from the fact that there is always a correct guessing path for the for loop iterating over vertices v. No wonder this method is called 'inductive counting', using $S(k-1)$, to compute $S(k)$.

Exercise 11 *Show that* $NSPACE(S(n)) = co - NSPACE(S(n))$ *for* $S(n) \geq \log_2 n$, *where* $S(n)$ *is a fully space constructible function.*