Fourth Week

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

In order to prove dense hierarchies, we need newer techniques because the hierarchy theorems on time and space established so far cannot fathom examples such as that in Example 12.5 [HU79]. Also, our space hierarchy theorem was for deterministic case; for non-deterministic cases too, as in Example 12.6, one can show separations easily using a new technique. We develop the notion of translational lemmas as in Section 12.5 in [HU79]. Note that the tape compression theorem and the linear speedup theorem were extendible to non-deterministic computations as well, unlike what we saw for deterministic hierarchy theorems.

Now consider the proof of Lemma 12.2 [HU79]. The first issue is to show that $L_2 \in NSPACE(S_1(n))$ (and therefore by the premise, also in $NSPACE(S_2(n))$).

Exercise 1 *Show that* $L_2 \in NSPACE(S_1(n))$.

[Hint: Since $L_1 \in NSPACE(S_1(f(n)))$, M_1 would require no more than $S_1(f(n))$ space to decide the membership of input string x of length n in L_1. Let i be $f(n) - n$. Then, the space required is $S_1(n+i)$, where as, the input of M_2 is of length $n + i$.]

Next we have M_3 accepting L_2 and we require M_4 to accept L_1 finally in $S_2(f(n))$ space. We can use nondeterminism here to guess the length i required for padding dollars after input x so that M_3 can successfully be simulated for accepting L_2, thereby helping M_4 to accept x in $S_2(f(n))$ space if M_1 accepts x in $S_1(f(n))$ space.

Instead of guessing i, M_4 may also simply increase dollar padding length i upto a maximum of $f(n) - n$ (until M_3 accepts), accepting its own input x, and rejecting otherwise.

This translation lemma can be used to establish non-trivial hierarchies as in Example 12.6 and Theorem 12.12 [HU79]. Take these as study exercises.

Next we study how we can keep blowing up the resource and still not accept any new languages. It turns out that even $g(S(n))$ space does not add any languages that $S(n)$ space cannot accept, for any total recursive function $g(n) \geq n$; here, $S(n)$ depends on the choice of $g(n)$. See Theorem 12.13 [HU79]. This result is therefore called the *gap theorem*, where the resource gap can be increased as much as one wishes without accepting any new languages. The proof shows that given $g(n)$, we can find $S(n)$ for all $n \geq 1$. In order to construct $S(n)$ we consider only machines M_1, ..., M_n in the enumeration of TM's. Essentially, we choose a value for $S(n)$ so that for no i from 1 through n, $S_i(n)$ lies between $S(n)$ and $g(S(n))$. Before we see how to do such assignments to $S(n)$, we show that this condition establishes the equality of $DSPACE(S(n))$ and $DSPACE(g(S(n)))$. For the sake of contradiction, suppose the equality does not hold due to membership of some language L. Then, $L = L(M_k)$ for

some (fixed) k where $S_k(n) \leq g(S(n))$ space suffices for L_k, for all n. So, L is in the bigger space class but not in the smaller one. However, recall that we defined $S(n)$ such that for all $n \geq k$, $S(n) \geq S_k(n)$. So, $S_k(n) \leq S(n)$ almost everywhere. Therefore $L = L_k$ is in $DSPACE(S(n))$ by Lemma 12.3, a contradiction.

Now we get into the details of constructing $S(n)$). To keep all $S_i(n)$ below $S(n)$, we simply need to find the largest $S_i(n)$, $1 \leq i \leq n$ and set $S(n)$ to that largest value. However, some $S_i(n)$ may be undefined (infinite). So, maximizing is absurd. Therefore we do the following iteration:

(1) j=1
(2) if there is an $M_i, 1 \leq i \leq n$ so that $S_i(n)$ is in between $j+1$ and $g(j)$
(3) then $j = S_i(n)$; go to (2)
(4) else $S(n) = j$

Exercise 2 *Why and how can we do step (2)? [Hint: Use Lemma 12.4 [HU79])*

Exercise 3 *Do Exercise 12.7 and study Lemmas 12.3 and 12.4.*