In the last lecture, we proved the following inclusions:

(i) $NSPACE(s(n)) \subset TIME(s^2(n))$
(ii) $TIME(t(n)) \subset SPACE(t^2(n))$

In the first part of this lecture, we will reduce $t^2(n)$ to $t(n)$ in (ii).

Last class, we gave the following algorithm that computes $A \in TIME(t(n))$ in a Deterministic TM $M$, in order to prove (ii):

\begin{verbatim}
Begin
  on input $x$
    let $C =$ initial configuration of $M$
    call $Eval(C)$
End

Eval($C$)
  if $C$ is halting
    then return T if $C$ is accepting
       F if $C$ is rejecting
  else
    if $C$ is an Existential node
      then let $C_1$ and $C_2$ be the successors of $C$
         if $Eval(C_1)$
           then return T
         else return $Eval(C_2)$
    if $C$ is a Universal node
      if negation of ($Eval(C_1)$)
        then return F
      else return $Eval(C_2)$
\end{verbatim}

Without loss of generality, be $M$ an ATM such that each non-halting configuration of $M$ has exactly two successors. In order to see this, notice that each node has a finite number of successors (since the machine is finite, it has a finite number of states). Let’s say, that the fan-out is $k$. Then we can transform our tree to an equivalent binary tree, just by introducing intermediate nodes. Note that this increases the depth of the tree by a constant factor of $\log k$. So, the machine built in this way, is just slower by a constant factor than the original machine.

Now, let’s introduce the following Global Variable: $PATH \in \{R, L\}^*$ (sequence of right and lefts moves). Define: $Eval(PATH)$ to be $Eval(C)$, where $C$ is the configuration reached from the initial configuration following the $R$ and $L$ moves from $PATH$.

We will still use the same algorithm as above, we just need to make the following modifications: substitute $C$ by $PATH$, and $C_1$ by $PATH L$ and $C_2$ by $PATH R$. 

The analysis of the space complexity of the new routine is straightforward. The routine can be computed in space $O(t(n))$, since in each recursive call to $EVAL(C)$, instead of having to store the entire $t(n)$-bit configuration $C$, we simply add one additional symbol to $PATH$. □

So, in this way we had finally achieved:

(iii) $ATIME(t(n)) \subseteq DSPACE(t(n))$

(i) and (iii) together shows us that: (Savitch’s theorem)

(iv) $NSPACE(s(n)) \subseteq DSPACE(s^2(n))$

Other results we want to prove here are:

(v) $ASPACE(s(n)) \subseteq DTIME(2^{O(s(n))})$

(vi) $DTIME(t(n)) \subseteq ASPACE(\log t(n))$

Proof of v:

Let $A \in ASPACE(s(n))$. $A = L(M)$, where $M$ is an ATM.

Algorithm:

on input $x$

write a list of all $2^{O(s(n))}$ configurations of $M$ that uses space $s(n)$

apply the labelling procedure until $C_i$ (the initial node) is labelled

(this last step takes polynomial time in the number of configurations.)

Labelling : Labels are 0 or 1. Halting and accepting: 1. Halting and rejecting: 0. Starting from $C_2$ see if $C_3,...,C_r$ ($r$ is a certain integer) are labelled. Then start from $C_3$ and so on. Until everything is labelled. When stop see the label of $C_i$. □

Proof of vi:

Let $A \in DTIME(t(n))$. Then, $A$ is accepted by a 1-tape TM $M$ in time $t^2(n)$. One can think of the computation done by $M$ as a Table. Each row of the table, represents the contents of the worktape, location of the head of the tape and state at a given time. Thus, each cell of the table is: or a state or a element of the tape alphabet or just a blank space.

The rows are indexed in this way : Starting from the lowest row at time $= 0$. Until reaching the top row at time $= t^2(n)$.

What is important to notice about this table, is that, the value at each cell is completely defined by the value of the nearest three cells below.

Algorithm:

on input $x$

call $Table(1, t^2(n), (q_{acc}, b))$

(note : $|x| = n$, $q_{acc}$ = accepting state, $b$ = blank space)
Table \((i, j, z)\) evaluates to:

T if at time \(j\) location \(i\) of the table contains \(z\)
F otherwise.

This is the alternating routine for Table:

\[
\text{Table } (i, j, z) \\
\text{if } j = 0 \\
\quad \text{then} \\
\quad \quad \text{if } i \leq n \\
\quad \quad \quad \text{return T iff } z = x_i \\
\quad \quad \quad \text{else} \quad \text{return T iff } z = b \\
\text{else} \\
\quad \text{Existentially guess } z_1, z_2, z_3, \ldots \text{ such that:} \\
\quad (\text{Table } (i-1, j-1, z_1) \ \text{AND} \ \text{Table } (i, j-1, z_2) \ \text{AND} \ \text{Table } (i+1, j-1, z_3)) \implies (\text{Table } (i, j, z)) \\
\quad \text{Universally check } \text{Table } (i-1, j-1, z_1) \\
\quad \quad \text{Table } (i, j-1, z_2) \\
\quad \quad \text{Table } (i+1, j-1, z_3).
\]

Let’s analyze space complexity of this last algorithm:

We essentially just need to store \(i\) and \(j\). These numbers are between 1 to \(t^2(n)\). Thus, they required \(O(\log t(n))\) space. □

(v) and (vi) imply:

\(P = \textit{ASPACE}(\log n)\).

Observations: Let \(A \in P\). Then, there is a family of circuits \(C_n : n \in N\), such that, each \(C_n\) is a circuit in \(n\)-inputs with a polynomial \(n^{O(1)}\) of gates iff:

\(x \in A\) iff \(C_{n}(x) = 1\),

and the function from \(n\) to \(C_n\) is easy to compute.

\(C_n\) is the circuit consisting of the configurations of the \(\text{ASPACE}(\log n)\) machine accepting \(A\).