18.405J/6.841J Advanced Complexity Theory	February 12, 1998

Lecture 3

Lecturer: Daniel A. Spielman *Scribe: Stacy McGeever*

1 Relation Between ATIME And SPACE

Theorem 1 $\forall\ f(n) \geq n : ATIME(f(n)) \subseteq SPACE(f(n)) \subseteq ATIME(f^2(n))$

We proved the first half of this theorem in the last lecture. We now show $SPACE(f(n)) \subseteq ATIME(f^2(n))$.

Proof

The idea is to turn the computational tableau of a PSPACE machine M into an instance of ST Connectivity (but without actually constructing a full graph).

Note that M can have at most $c^{f(n)}$ distinct configurations, each of length $f(n)$. The constant c depends on the number of alphabet symbols $|\Sigma|$ and states $|Q|$ in the finite control of M. In particular:

$$\begin{aligned}
\#\ distinct\ configurations &= \#\ distinct\ strings\ of\ symbols \times \#\ states \times \#\ tape\ positions \\
&= |\Sigma|^{f(n)} \times |Q| \times f(n) \\
&\leq c^{f(n)}
\end{aligned}$$

We can order all possible configurations systematically, so an index into this ordering will suffice to specify a particular configuration, which we can then construct. We need $log(c^{f(n)}) = O(f(n))$ bits for this index.

Imagine a directed graph whose nodes are configurations of M. One of these nodes (S_w) represents the start configuration of M on an input w. Another node (T) will represent the accepting configuration (we can assume without loss of generality that there is only one such configuration). Two configuration nodes C_1 and C_2 have an edge between them iff the configuration represented by C_2 is reachable from the configuration represented by C_1 in one step. Since M's transition function is deterministic, each node has at most one output edge - hence if there's a path from S_w to T, it's unique.

The graph described above has $c^{f(n)}$ nodes, so we can't build the entire thing. But we can use recursion and alternation to solve the problem. First, we need an algorithm to determine whether a path of length k exists between two nodes C_i and C_j:

$FromTo\ (C_i, C_j, k)$

IF $k = 1$

return *TRUE* if C_j is reachable from C_i in one step, else return *FALSE*

ELSE

existentially guess an intermediate configuration, C_m

return $FromTo(C_i, C_m, ceiling(k/2))$ *AND* $FromTo(C_m, C_j, floor(k/2))$

This algorithm runs in $O(log(k))$ steps (not time!).

We now build an ATM N such that $L(N) = L(M)$. On input w, N calls $FromTo(S_w, T, c^{f(n)})$. N accepts w iff that call returns *TRUE*.

Since the existence of a path from S_w to T means that there is an accepting computation of M on w, it is clear that $w \in N$ implies $w \in M$. The reverse direction is also clear from the construction of the graph, hence $L(M) = L(N)$.

N uses alternation to universally *AND* together the recursive calls in the last step of $FromTo$ and to existentially guess the intermediate configurations. On any computational branch, there will be at most $O(log(c^{f(n)})) = O(f(n))$ calls to $FromTo$. Each call will result in writing a constant number of configurations or indices to configurations, so every step is $O(f(n))$. Determining if C_j is reachable from C_i in one step for the base case is also computable in polynomial time. Hence N runs in ATIME($f^2(n)$), and the proof is complete. ■

Corollary 2 *AP = PSPACE.*

2 Relation Between ASPACE and TIME

Theorem 3 $\forall\ f(n) \geq n : ASPACE(f(n)) = TIME(2^{O(f(n))})$

Proof
First, we prove containment in the forward direction.

$- > ASPACE(f(n)) \subseteq TIME(2^{O(f(n))})$

Main idea: we can write down the entire computational history of an ASPACE($f(n)$) machine in time exponential in $f(n)$.

Given an ATM M in ASPACE($f(n)$), the length of the longest computational step is $f(n)$, so M will have at most $c^{f(n)} = 2^{O(f(n))}$ distinct configurations (where c is a constant depending on M). The reasoning is exactly the same as in the proof of Theorem 1.

Imagine a directed graph whose nodes are configurations of M. Each of these nodes will have at most two children representing reachable configurations (as a result of the way we defined ATMs). We have enough time to write out the entire graph, and when building the nodes, we leave space for marking the node *accept* or *reject*. Acceptance of an input w by M corresponds to whether the starting configuration node S_w in this graph evaluates to *accept*.

Let x be a node in our graph. Starting from the leaves upward, evaluate the nodes according to the following algorithm:

$MarkNode$ (x)

> If x is $\exists$:
> mark it as *accept* if any child is marked *accept*, else mark *reject*
> If x is $\forall$:
> mark it as *accept* if all children are marked *accept*, else mark *reject*
> If x is deterministic:
> use the child's value to compute *accept* or *reject*

$MarkNode$ marks at least one more node as *accept* or *reject* every time the graph is scanned, so there will be at most $2^{O(f(n))}$ scans. Each scan takes time $2^{O(f(n))}$, so the running time of this algorithm is at most $(2^{O(f(n))})^2$, which is still $2^{O(f(n))}$, giving $ASPACE(f(n)) \subseteq TIME(2^{O(f(n))})$.

We now show containment in the other direction.

$< -TIME(2^{O(f(n))}) \subseteq ASPACE(f(n))$.

Note that an ASPACE($f(n)$) machine cannot write out even a single configuration of a TIME($2^{O(f(n))}$) machine M. However, we can keep an index into the computational tableau. The idea is that any cell $C_{i,j}$ in the computational tableau is determined completely by the three cells $C_{i-1,j-1}$, $C_{i-1,j}$, $C_{i-1,j+1}$ on the row directly above it.

We assume without loss of generality that M is a one-tape TM with a unique accepting configuration. The computational tableau of M has $2^{O(f(n))}$ columns (representing space), and $2^{O(f(n))}$ rows (representing time). Each row or column index can be represented by $log(2^{O(f(n))}) = O(f(n))$ bits. Assume that when M accepts, it enters a special state (q_{accept}), so acceptance by M can be determined by checking the contents of the cell on the bottom left-hand corner of the tableau. In particular, for some constant c, we will check to see whether the state of cell $(c^{f(n)}, 1) = q_{accept}$.

The circuits we will construct to hook up a cell to the cells above it simply use the finite control to determine whether the contents are valid. Each of these circuits is constant in size since the alphabet size $|\Sigma|$ and number of states $|Q|$ are constant. Let σ be a member of $\Sigma \times Q$. The procedure $IsIn(i, j, \sigma)$ returns *TRUE* iff the (symbol, state) pair of cell (i, j) is σ. Let w_j = the jth symbol of input w, and q_{start} = the start state of M. $IsIn$ can be defined recursively as follows:

$IsIn\ (i, j, \sigma)$

    IF $i = 1$
        if $\sigma = (w_j,\ q_{start})$ return *TRUE* else return *FALSE*
    ELSE
        universally select cell triples $(i-1, j-1)$, $(i-1, j)$, $(i-1, j+1)$
        existentially guess the contents $(\sigma_1, \sigma_2, \sigma_3)$ of those three cells
        verify that these contents yield σ in cell (i, j) in one step
        return $IsIn(i-1, j-1, \sigma_1)$ *AND* $IsIn(i-1, j, \sigma_2)$ *AND* $IsIn(i-1, j+1, \sigma_3)$

We can now define an ATM N where $L(M) = L(N)$. On input w, N calls $IsIn(c^{f(n)}, 1, q_{accept})$. N accepts w iff this call returns $TRUE$.

N uses alternation to to deal with universal selection and existential guessing. At each step of the computation, we use at most $O(f(n))$ space to write the indices of the cells; verification that the guessed contents yield the correct content takes only constant space. Hence $TIME(2^{O(f(n))}) \subseteq ASPACE(f(n))$. ∎

Corollary 4 *$AL = P$.*

Corollary 5 *$APSPACE = EXPTIME$*

3 (A Brief) Introduction to the Polynomial Hierarchy

Question : How do we get a class that we haven't seen yet by using alternation?

Definition 6 *A $\Sigma_k P$-machine is a polytime alternating machine which starts with $\exists$-alternation and changes quantifiers at most $k-1$ times.*

Example: $\Sigma_1 P$ = NP.

Definition 7 *A $\Pi_k P$-machine is a polytime alternating machine which starts with $\forall$-alternation and changes quantifiers at most $k-1$ times.*

Alternatively, we could say that $\Pi_k P = co - \Sigma_k P$.

Example: $\Pi_k P$ = co-NP