18.405J/6.841J Advanced Complexity Theory February 11, 1997

Lecture 3

Lecturer: Daniel A. Spielman *Scribe: John Dunagan*

Alternating Turing Machines[1]

ATM's (Alternating Turing Machines) are an extension of Turing Machines which are allowed to go through both $\exists$ states and $\forall$ states. The $\exists$ states capture non-determinism, while the $\forall$ states capture co-non-determinism. Alternating Turing Machines yield a different but equivalent characterization of the polynomial-time hierarchy. Define $\Sigma_k P$ to be those languages accepted by a PTIME ATM that begins with $\exists$ and has k levels (i.e., alternating between $\exists$ and $\forall$ at most k times). We define $\Pi_k P$ similarly:

$$\Sigma_k P = \{\text{languages } L : L \text{ is accepted by a polynomial-time ATM that begins with } \exists \text{ and has at most k levels}\}$$

$$\Pi_k P = \{\text{languages } L : L \text{ is accepted by a polynomial-time ATM that begins with } \forall \text{ and has at most k levels}\}$$

This is exactly equivalent to our previous definition of the polynomial time hierarchy. We similarly define QBF_k to be the Quantified Boolean Formulas beginning with $\exists$ and alternating quantifiers at most k-1 times. It can easily be shown that QBF_k is complete for $\Sigma_k P$. If $\Sigma_k P = \Pi_k P$, then the polynomial time hierarchy collapses, and $\Sigma_k P = \Sigma_n P \; \forall n \geq k$. This, as we have previously remarked, is predominantly conjectured not to happen.

Alternating Space = Time, Alternating Time = Space

The gist of today's lecture will be ASPACE = TIME, ATIME = SPACE. We will also derive sharp bounds on general QBF. For convenience, we repeat several definitions here:

$$SPACE(f(n)) = \{\text{languages } L : L \text{ is accepted by a TM that uses at most f(n) space}\}$$

$$NSPACE(f(n)) = \{\text{languages } L : L \text{ is accepted by an NTM that uses at most f(n) space}\}$$

$$ASPACE(f(n)) = \{\text{languages } L : L \text{ is accepted by an ATM that uses at most f(n) space}\}$$

$$PSPACE = \{\text{languages } L : L \in SPACE(n^k) \text{ for some } k, \text{ where } n \text{ is the length of the input}\}$$

[1] I would first like to ask anyone who can show that NP can be solved by a P quantum computer to talk to me after class.

$NPSPACE$ and $APSPACE$ are defined in the obvious manner. We pause to note that $SPACE(f(n)) \subset NSPACE(f(n)) \subset ASPACE(f(n))$. We also pause to note once again a reason to exclude constant factors and stick to Big-O notation. The space/time speedup theorems: by doubling the alphabet size and squaring the state space, we can convert any Turing Machine into one which uses half as much space and half as much time.

LSPACE and More Theory Background

Define L = LSPACE = log-space = $\bigcup c \geq 0$ SPACE($c \cdot log(n)$) = SPACE($log(n)$). Note that there is some subtelty in defining logarithmic space requirements; we must require our theoretical Turing Machine models to have a read-only input tape, a work tape, and a write-only output-tape. The work tape is bounded to have only log-space. L $\subset$ P. L $\neq$ P ? This is a standard conjecture. It is known that L $\neq$ PSPACE ; this is easy to show using the space hirearchy theorem. We pause now to note both the space and time-hierarchy theorems,

Theorem 1 (Time Hierarchy Theorem)
If $f(n) = o(g(n)/\log g(n))$, then $TIME(f(n)) \neq TIME(g(n))$.

Theorem 2 (Space Hierarchy Theorem)
If $f(n) = o(g(n))$, then $SPACE(f(n)) \neq SPACE(g(n))$.

It should be noted that there are P-complete problems under log-space reduction, just as there are NP-complete problems under poly-time reductions (what we think of when we think of NP-complete problems). As an interesing side note, every known NP-complete problem is known to be NP-complete under log-space reductions. We now proceed to two important theorems, Savitch's Theorem and the Immerman-Szelepcsenyi Theorem.

Theorem 3 (Savitch's Theorem)
$NSPACE(f(n)) \subset SPACE(f^2(n))$ if $f(n) \geq logn$.

Theorem 4 (Immerman-Szelepcsenyi Theorem)
$NSPACE(f(n)) = coNSPACE(f(n))$ if $f(n) \geq logn$.

We can easily see from the second that there is no polynomial space-hierarchy analogous to the polynomial time-hierarchy. (As a reminder, coNSPACE(f(n)) can be defined as the complement of all NSPACE(f(n)) languages or as the language accepted by NTM's with universal states in space f(n); that is, ATM's in space f(n). These are equivalent definitions.) We now move on to proving two important theorems about alternating time and space.

Theorem 5 (Well-Known Theorem A)
$ATIME(f(n)) \subset SPACE(f(n)) \subset ATIME(f^2(n))$
if $f(n) \geq n$.

Theorem 6 (Well-Known Theorem B)
$ASPACE(f(n)) = TIME(2^{O(f(n))})$
if $f(n) \geq logn$.

Proof of Well-Known Theorem A

We first show $ATIME(f(n)) \subset SPACE(f(n))$. Represent the machine we are going to simulate (the f(n)-time ATM) as a tree of $\forall$ and $\exists$ quantifiers (or normal deterministic transition states), each branch of which eventually ends in an accept or reject state. We note that

1. one can represent each node with f(n) space

2. the tree has height at most f(n)

We traverse the tree using DFS[2] to keep as little state around as possible. A first (that is, naive) implementation, would store the entire path through the tree so far. This takes space $f^2(n)$. A better implementation is to store the path so far (i.e., LRRLLRLLLLR) and to recompute the previous state when traversing the tree upwards, instead of storing it. This takes space $f(n)$.
An alternative analysis which also works is to note that the change between any two states is constant, so movement from one node to the next takes up at most a constant amount of space.
We now show $SPACE(f(n)) \subset ATIME(2^{f(n)})$. Consider the graph of the computational universe of the machine we are going to simulate (the $f(n)$-space machine), where every possible configuration is a node in the graph. Define the following predicate FromTo(x,y,k) to be true iff one can get from x to y in at most k steps. FromTo(x,y,k) = $\exists$ z : (FromTo(x,z, $\lceil \frac{k}{2} \rceil$) $\wedge$ FromTo(z,y, $\lfloor \frac{k}{2} \rfloor$)).[3] The maximum initial k to get from the start node to the accept node in our graph is $2^{O(f(n))}$. Generating an x or y or z takes $f(n)$ time. Recursion (that is, running through the whole thing) on the predicate FromTo(x,y,k) takes $log(k)$ time, for a total time of $log(2^{O(f(n))}) \cdot f(n) = O(f^2(n))$ to evaluate our initial predicate FromTo(start, accept, $2^{O(f(n))}$).
As a caveat, $f(n)$ must be time constructible in time $f^2(n)$, and we won't worry about cases in which it isn't. This concludes our proof.

Proof of Well-Known Theorem B

We first show $ASPACE(f(n)) \subset TIME(2^{O(f(n))})$. Consider a graph of all the configurations in the Alternating Turing Machine's computation history, with up to two arrows from any given $\forall$ or $\exists$ vertex to other vertices. Given a particular input, let our non-alternating Turing Machine write out the entire graph in $2^{O(f(n))}$ time. Second, don't do DFS from the start state - making the proof work in this direction requires more hacking than we need. Instead, begin at the final accept state, and mark all states which lead to the final accept state as accepting states. (By omission, we also mark rejecting states.) Repeat this process, marking all states which lead to states previously marked accepting until no new states are marked. If the start state is marked, accept; otherwise, reject. This takes constant time per node since we mark each state at most once.

We now show $TIME(2^{f(n)}) \subset ASPACE(f(n))$. Consider a tableau representing the computation history of the Turing Machine, and require a unique accept state, as always. Define the function $cell(i,t)$ to be the contents of cell i at time t. Since $cell(i,t)$ is determined completely by $cell(i-1,t-1)$, $cell(i,t-1)$, and $cell(i+1,t-1)$, we can verify an assertion of the form "$cell(i,t) = x$" by using existential quantification to guess the values of $cell(i-1,t-1)$, $cell(i,t-1)$, and $cell(i+1,t-1)$, checking that they lead to $cell(i,t) = x$, and then using universal quantification to verify each of the three guesses.[4] To check the contents of $cell(i,0)$, we just look at the input tape. We use this method to verify that the machine is in the accept state after $2^{f(n)}$ steps.

Since $0 \le t \le 2^{f(n)}$, we need $log(2^{f(n)}) = f(n)$ space to write out i and t. Moreover, the machine does not need to store any other information. This concludes our proof that $TIME(2^{O(f(n))}) = ASPACE(f(n))$. We note that corollaries of this are $AL = P$ and $AP = PSPACE$. $AP = PSPACE$ is the same as QBF complete for $PSPACE$.

[2] Depth First Search

[3] Note that $\wedge$ is the hidden universal quantifier in this expression - we could not do the same thing with a NTM - both $\exists$ and $\forall$ are necessary.

[4] Note that every pass through the $\forall$ quantifier decreases the remaining time by 1.