6.841 Advanced Complexity Theory Mar 02, 2009

Lecture 08

Lecturer: Madhu Sudan *Scribe: Vartika Singh*

1 Alternation

It is a notion which tries to capture languages in NP and $CoNP$ in a unifying way. It defines the time-vs-space relation and hence Alternating Turing Machines.
It is a means for finding lower bounds for certain languages for which there is no obvious short certificate for membership and hence can not be characterized using non determinism alone.

1.1 Alternating Turing Machines, ATM

Definition 1 *Let M be an alternating TM. For a function $T : N \longrightarrow N$, we say that M is an $T(n)$-time ATM if for every input $x \in (0,1)^*$ and for every possible sequence of transition function choices, M will halt after at most $T(\mid x \mid)$ steps.*

Alternating Turing Machines (ATM), are generalizations of TM with the additional internal states of $\exists$ or $\forall$. The ATM can be thought of as a tree, starting from the root node. If the state reached is a deterministic one, then there is only one move to make. If it is either of $\exists$ or $\forall$, then either of $(0,1)$ moves could be made. Each node can be thought of as a configuration of M on input x, and there is an edge from configuration C or C^1, if latter can be obtained from former in one step. The nodes up a computation path are labeled by repeatedly applying the following rules, till they can not be applied anymore:

- The configuration C_{accept} where the machine is in q_{accept} is labeled "ACCEPT".
- If a configuration C is in a state labeled *exists* and one of the configurations C^1 reachable from it in one step is labeled "ACCEPT" then we label C "ACCEPT".
- If a configuration C is in a state labeled $\forall$ and both the configurations C^1 and C^2 reachable from it one step is labeled "ACCEPT" then we label C "ACCEPT".
- If a configuration C is in a deterministic state then just proceed up as in a unary computation.

We say that M accepts x if at the end of this process the starting configuration C_{start} is labeled "ACCEPT". The language accepted by M is the set of all x's such that M accepts x.
Three resources are important in defining the complexity of a ATM.

- **TIME** : $ATIME(t(n)) \; = \; \{$L | L is decided by some t(n)-time ATM$\}$
- **SPACE** : $ASPACE(s(n)) \; = \; \{$L | L is decided by some s(n)-space ATM$\}$
- **Alternations** : Most important resource that will be used is maximum number of alternations done during the course of computation. (Note: Flip from $\exists$ to $\exists$ state is not a valid alternation. Similarly for $\forall$.)

1.2 NDTMs and CoNDTMS

NDTM can viewed as a TM with extra existential state. The machine accepts if one or more of its branches ends in an accept state.
CoNDTM can be viewed as a TM with universal quantifier states. At each node, the machine can be thought of as spinning off two parallel actions and taking BOTH paths at the same time. The machine accepts if all of its branches end in the accept state.

2 Relation to Basic Complexity Classes

The question arises as to how much we can accomplish in alternating poly time or in alternating logspace. Answering the above questions, we have the following very intriguing concept of time and space:

- $ATIME(poly) \; = \; PSPACE$
- $ASPACE(\log n) \; = \; P$

That is, time and space can be defined in terms of each other respectively.
We will now explore the relationship between the classes ATIME(t(n)), ASPACE(s(n)) and classical complexity classes defined using time or space bounded deterministic Turing machines.

2.1 Relating ATIME and SPACE

Theorem 2 $SPACE(s(n)) \; \subseteq \; ATIME(O(s(n)^2)) \; \subseteq \; SPACE(O(s(n)^2))$.

Proof We first prove the second part of the theorem, i.e. $ATIME(O(s(n)^2)) \subseteq SPACE(O(s(n)^2))$. Assume a binary tree, wherein we can keep track of our position by storing the control state, left or right head movement, symbol overwritten on the work tape and the outgoing transitions known to be accepting. This information will enable us to propagate the computation backwards while labeling the nodes of the tree. This all can be done in a space efficient manner, requiring up to $O(s(n))$ space and the same amount of space for storing

the complete configuration of the node. Now, we simulate the ATM by a deterministic Turing machine in a depth first manner. Due to the space efficient representation we are able to do this in $O(s(n))$ space and in the same space we are able to propagate the computation back towards the root node. Since, the computation is run down and then up across any computation branch, the total time is of the order of $O(s(n)^2)$.
The first part of the theorem has proof lying conceptually in Savitch's theorem. We first specify the configuration of the Turing Machine M. It is the sequence of symbols stating the configuration of the machine. Here, the machine uses $O(s(n))$ space on its work tape, thus we need at least $O(s(n))$ bits to represent a configuration. The upper bound on the running time of such a machine is $2^{O(s(n))}$. Given, the machine M, which is in $SPACE(s(n))$ we construct a ATM T which runs in $2^{O(s(n))}$ time. We will use the following notation to denote our problem:

$$REACH?(C_I, C_F, 2^{O(s(n))})$$

which says that given a start configuration C_I, is it possible to reach a configuration C_F in $(2^{O(s(n))})$ time. We do this by guessing the configuration at the middle of the computation(existential) and then verifying whether all paths terminate at C_F, starting from C_I configuration in $2^{O(s(n))} - 1$ steps. We do this recursively for each halve, till we can reach the point of difference in one step. This constitutes one alternation. The running time is $O(s(n)\log(2^{O(s(n))})$ or $(s(n)^2)$ time.
Thus, every existential quantifier to universal quantifier alternation implies, quadratic blowup in time.

$$ATIME(poly) \;=\; PSPACE$$

■

2.2 Relating TIME and ASPACE

Theorem 3 $ASPACE(O(s(n)) \;\subseteq\; TIME(2^{O(s(n))})$.

Proof Let $O(s(n))$ be s for notational efficiency. Now, for a space s ATM A, there are 2^s possible configurations. In order to tell what the machine is doing at a particular state, we label each node as either existential, deterministic or universal. If we look at the problem as reachability problem, the computation path may keep looping forever and thus never terminate. Syntactically, such a computation path should lead to a reject state. In order to bypass this problem, we represent this huge graph in the form of layers, indexed from L_0 to L_{2^s}. Some of the nodes in a layer might be accepting or rejecting. This, implies that a computation path has ended before reaching the L_{2^s}th layer. All the nodes in L_{2^s}th layer are labeled rejecting, because in case a computation ever reaches there, it would mean that it is a looping computation. We label one node in layer L_0 as start node and rest of the computation in the graph follows the

notion of a branch! ing program.
Now, this machine is running in $TIME(2^s)$. If we represent these 2^s steps as a square matrix, the $(i,\ j)$ cell of which tell‘s us the current representation, then the space required would be to the order of 2^s. We can skimp on space requirement significantly by observing that in order to check a cell in the tableau, we need to just look at just 3 cells in the row above(as shown in figure 2), We

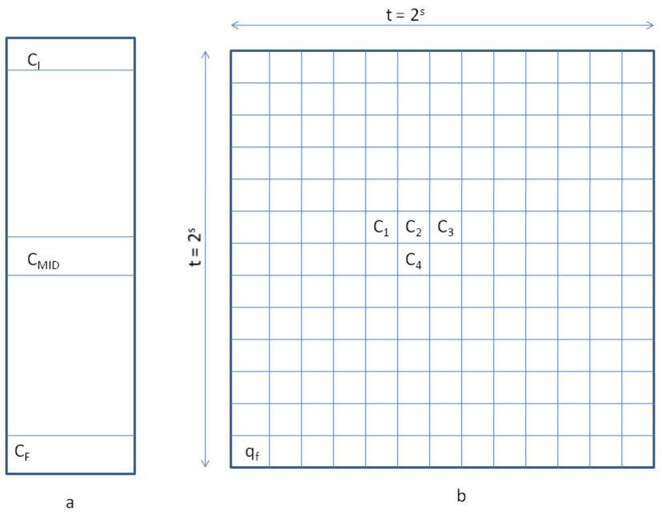

Figure 1: Figurer shows the tableau depicting the sequence of 2^s possible length $O(s(n))$ configurations.

define the problem as $CELL?(2^s, 1, q_f)$, which says that, is the configuration q_f reachable if starting from configuration t, and does it takes the accept(1) state(shown in fig 1(b)). We solve this problem generically by recursively asking the question, $CELL?(i, j, C_4)$. This can be done in $\log 2^s$ bits, as i and j need so many and C_i‘s need only a constant number of bits. We can now write the problem as: Guess a sequence of three cells C_1, C_2 and C_3. Then, for all verify that whether the following holds true:

$$\begin{aligned} & CELL?(i-1, j-1, C_1) \\ \wedge\ & CELL?(i-1, j, C_2) \\ \wedge\ & CELL?(i-1, j+1, C_3) \\ \wedge\ & VALID?((C_1, C_2, C_3), C_4, 1) \end{aligned}$$

This can be done in constant time. Thus, $CELL?(2^s, 1, q_f)$ runs in space $O(s(n))$ and reaches the accepting configuration q_f.
■

3 Fortnow's Theorerm

Theorem 4

$$\forall c < \infty,\ \exists \epsilon > 0\ s.t.$$

$$SAT \ \in \ SPACE(c\dot{\log}n) \qquad (1)$$

$$\Rightarrow SAT \ \notin \ TIME(n^{1+\epsilon}) \qquad (2)$$

3.1 Intuitive overview

Based on our belief that polynomial hierarchy does not collapse, it is implied that $NP \neq P$. We conclude from this that SAT does not have a linear-time algorithm, and that it does not have a log space algorithm. We will use alternation to derive the following contradiction:
$\longrightarrow$ If $SAT \ \in \ TIME(N^{(1+\epsilon)})$
$\Rightarrow$ Existential quantifier is not powerful
(Which would also mean that the universal quantifier is not powerful)
$\Rightarrow$ Alternation is not powerful.
$\longleftarrow$ $SAT \ \in$ small space and for small spaces alternation is powerful.
Proof We will use the stronger version of Cook's Theorem, which states that languages in $NTIME(T(n))$ reduce to SAT on formula of length $T(n)\log T(n)$. We fix T(n) to be sufficiently large and we proceed to prove that $TIME(T(n))$ computation can find a satisfiability for problem of length $T(n)$. (Which then implies that $SAT \ \in \ TIME(n^{1+\epsilon})$.) Using the above, we proceed to prove claim 1.

$$TIME(T(n)) \approx \ SAT formula of size \approx T$$

(We know SAT is in logspace)

$$\subseteq SPACE(c\log(T(n)))$$

Now, we make the following claim for small space computations. Let a be the number of alternations and s be the number of steps taken by the ATM M to reach an accepting computation. Then,

$$\forall a\ SPACE(s) \ \subseteq \ ATIME\left[a, a.s2^{s/a}\right]$$

Using the same theme as Savitch's theorem, wherein, the computation time (or configuration space) is continuously divided in half, we instead, divide the computation time in sequence of k slices. We guess these k states. If the state is existential we perform two alternations and guess the next computation universally. Thus, for each computation of s steps, and taking into account the branching, the number of times we do alternations is $2a$. Thus, the total time taken for this computation would be $2ak$. Now, with each split, the size of $k \ = \ 2^{S/a}$. Thus, we have :

$$\forall aSPACE(Os(n)) \ \subseteq \ ATIME\left[a,\ as2^{\frac{s}{a}}\right]$$

Now, using the stronger form of Cook's theorem and the condition $s = a$,

$$\forall a TIME(t) \ \subseteq \ ATIME\left[a, \ a \log t t^{c/a}\right]$$

It is clear that alternation becomes very powerful in case of time.
Now, we prove the other side, that is alternation is not powerful. Assume, a tree, with its root being an existential quantifier and a intermediate nodes labeled as universal quantifiers and all the leaf nodes labeled as the existential quantifiers. If we get rid of one existential node at the leaf level we have the inclusion:

$$ATIME\,[a, \ t] \ \subseteq \ ATIME\left[a-1, \ t^{1+\epsilon}\right]$$

We do a non-deterministic form of computation. If we proceed inductively down the tree in the above manner, after eliminating all the existential nodes at the leaf level, we eventually reach:

$$ATIME\,[a, \ t] \ \subseteq \ ATIME\left[0, \ t^{(1+\epsilon)^a}\right]$$

If we pick ϵ to be sufficiently small then

$$ATIME\,[a, \ t] \ \subseteq \ TIME\left[\ t^{1+2a\epsilon}\right]$$

where $\epsilon \ll \frac{1}{a}$. Using the stronger version of Cook's theorem, we have the following:

$$TIME(t(n)) \ \subseteq \ TIME(t(n)^{\frac{c}{a}})^{1+2\epsilon}$$

. If we pick a sufficiently large, say for example, $a = 2c$, we have a statement which contradicts time hierarchy theorem. ■