Lecture 5

Alternating Space & Time Theorems

January 20, 1998
Lecturer: Richard Ladner
Notes: Sumeet Sobti

In this lecture, we complete the proof of the Alternating Space Theorem. We also prove the Alternating Time Theorem. Thereafter, we look at a more powerful variant of alternating Turing machines.

5.1. Alternating Space Theorem

In the last lecture, we proved one half of the Alternating Space Theorem. Here we complete the proof.

Theorem 5.1 (Alternating Space): If $s(n) \geq \log n$, then

$$\text{ASPACE}(s(n)) = \bigcup_c \text{TIME}(2^{cs(n)}).$$

Proof:

($\subseteq$) Proved in the last lecture.

($\supseteq$) Suppose $T = (K, \Sigma, \Gamma, \delta, s)$ is a single-tape deterministic Turing machine that takes at most $2^{cs(n)}$ steps on an input of length n. (Note that we are not loosing any generality in assuming T to be a single-tape Turing machine. See Tape-Reduction theorem).

For now, lets assume that the $2^{cs(n)}$ time bound is known. (Later we'll see how the proof can be modified to work without this assumption.)

Lets represent the configurations of the machine by strings of the form $\#\alpha q\beta\#\#$, where q denotes the state of the machine, $\alpha\beta$ is the tape contents ($|\alpha\beta| = 2^{cs(n)}$) and α is the string on the portion of the tape that lies to the left of the head. Thus, each machine configuration is represented by a string of length $m = 2^{cs(n)} + 4$.

We now design an ASPACE($s(n)$) algorithm to decide membership in $L(T)$.

If D is the configuration that follows from C in one step, then for any $2 \leq k \leq m - 2$, the k^{th} symbol of WD is fully determined by the $(k-1)^{th}, k^{th}, (k+1)^{th}$ and $(k+1)^{th}$ symbols of C. Thus we can define a predicate F in the following way.

$F(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma)$ is true if and only if σ must be the k^{th} symbol in the succeeding configuration if $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ are respectively the $(k-1)^{th}, k^{th}, (k+1)^{th}$ and $(k+2)^{th}$ symbols in the preceding configuration.

Also it is easy to see that F can be deterministically and computed in constant time.

Now, consider a fixed input string x of length n. Define the predicate R_x in the following manner. Let x_i be the i-th character of x.

$R_x(t, k, \sigma)$ is true iff σ is the k^{th} symbol of the t^{th} configuration of T when T runs on input x.

An *alternating algorithm* for computing $R_x(t, k, \sigma)$:

- if $(t = 0)$ then return
$$\begin{aligned}&[(k = 1 \wedge \sigma = \#) \vee\\ &(k = 2 \wedge \sigma = s) \vee\\ &(k = 3 \wedge \sigma = x_1) \vee \ldots \vee (k = n+2 \wedge \sigma = x_n) \vee\\ &(n+3 \leq k \leq m-2 \wedge \sigma = \sqcup) \vee\\ &(m-1 \leq k \leq m \wedge \sigma = \#)]\end{aligned}$$
- if $(t > 0)$ then return
$$\begin{aligned}&[((k = 1 \vee m-1 \leq k \leq m) \wedge \sigma = \#) \vee\\ &(2 \leq k \leq m-2 \wedge \exists \sigma_1, \sigma_2, \sigma_3, \sigma_4 (F(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma) \wedge\\ &\qquad R_x(t-1, k-1, \sigma_1) \wedge\\ &\qquad R_x(t-1, k, \sigma_2) \wedge\\ &\qquad R_x(t-1, k+1, \sigma_3) \wedge\\ &\qquad R_x(t-1, k+2, \sigma_4)))]\end{aligned}$$

The correctness of the above algorithm is obvious. An important thing to note here is the way this algorithm will be executed by an alternating TM. Lets see in a little detail how an alternating TM may execute this algorithm.

Suppose the alternating TM wants to compute the value of a formula $\psi_1 \vee \psi_2$. What it does is that it goes to an existential state (if it is not in one already), and 'forks' out into two branches. It computes ψ_1 on one branch and ψ_2 on the other. Similarly if it wants to compute $\psi_1 \wedge \psi_2$, it goes to a universal state and forks out into two branches.

If it needs to compute $\exists x : \psi$, then it goes to an existential state and forks out into a number of branches. On each branch, it takes a different value of x and computes ψ. Similar is the computation of $\forall x : \psi$.

When the above algorithm is executed in this manner by an alternating TM, it can be easily seen that the only space it requires is to store the parameters t, k and σ. Now $0 \leq t \leq 2^{cs(n)}$, $1 \leq k \leq m$ and $\sigma \in K \cup \Gamma \cup \{\#\}$. Thus the parameters can be stored in $O(s(n))$ space. So, the above algorithm is an ASPACE($s(n)$) algorithm.

Using the above algorithm, we can easily write an ASPACE($s(n)$) algorithm for deciding membership in $L(T)$.

An *alternating algorithm* for deciding $L(T)$:
Input x.

- return $\exists t, k : (0 \leq t \leq m \wedge 1 \leq k \leq m \wedge R_x(t, k, Y))$ where Y is the accepting state of T.

To remove the assumption about apriori knowledge of $s(n)$, we stylize T in a particular manner. We let T have a unique accepting and a unique rejecting configuration. For example, we may stipulate that T always halts with its tape contents erased and its head at the left end of the tape. In this case, the unique accepting configuration will be $\#Y \sqcup \sqcup \ldots \sqcup \#\#$, where Y is the accepting state. Similarly, the rejecting state will also be unique. Now, one can cycle through $m = 3, 4, \ldots$ until either the accepting or the rejecting configuration is found. The details are left as an exercise to the students.

□

5.2. Alternating Time Theorem

Theorem 5.2 (Alternating Time): If $t(n) \geq n$ and $s(n) \geq n$, then

$$\text{ATIME}(t(n)) \subseteq \text{SPACE}(t(n))$$

$$\text{NSPACE}(s(n)) \subseteq \text{ATIME}(s^2(n))$$

Proof: First we show that ATIME$(t(n)) \subseteq$ SPACE$(t(n))$. Suppose T is a $t(n)$ time bounded alternating TM, which on every input makes at least one move. W.l.o.g., assume that at each step, T has exactly d choices for the next move (where d is a constant).

Fix a given input x of length n. Now, by definition of acceptance in alternating TMs, $x \in L(T)$ iff the computation tree of T on x is an accepting tree. Thus, it would suffice if we can give an SPACE$(t(n))$ deterministic algorithm to test if the computation tree of T on x is accepting or not. This is exactly what we are going to do. We will a give *space-efficient* evaluation procedure for computation trees of T.

Suppose R is the computation tree of T on input string x. A computation path starting from the root of R (i.e. Init(x)) can be represented as a sequence of integers $\sigma = i_1 i_2 \ldots i_k$, where i_j indicates which of the d possible moves was made at the j^{th} step in the computation. Thus, $\forall j : 1 \leq i_j \leq d$. Also since T is a $t(n)$-time bounded machine, no computation path can be mose than $t(n)$ steps long. Therefore, $k \leq t(n)$.

For any path $\sigma = i_1 i_2 \ldots i_k$, we define the following premitives.

$last(\sigma) = \lambda$ if $k = 0$, and i_k if $k \geq 1$.
$back(\sigma) = i_1 i_2 \ldots i_{k-1}$.
$down(\sigma) = i_1 i_2 \ldots i_k 1$.
$conf(\sigma) =$ the configuration T reaches by following path σ.

Algorithm for evaluating computation tree of T on input x:

$\sigma = \lambda, b =?, v =?$.
while $\sigma \neq \lambda$ or $b \neq \lambda$ or $v =?$ do

$v = ?$:
$conf(\sigma)$ is accepting: $b = last(\sigma)$; $\sigma = back(\sigma)$; $v = T$
$conf(\sigma)$ is rejecting: $b = last(\sigma)$; $\sigma = back(\sigma)$; $v = F$
else: $\sigma = down(\sigma)$

$v = T$:
$conf(\sigma)$ is existential: $b = last(\sigma)$; $\sigma = back(\sigma)$; $v = T$
$conf(\sigma)$ is universal $\wedge b = d$: $b = last(\sigma)$; $\sigma = back(\sigma)$; $v = T$
$conf(\sigma)$ is universal $\wedge b \neq d$: $b = ?$; $\sigma = \sigma(b+1)$; $v = ?$

$v = F$:
$conf(\sigma)$ is universal: $b = last(\sigma)$;$\sigma = back(\sigma)$; $v = F$
$conf(\sigma)$ is existential $\wedge b = d$: $b = last(\sigma)$; $\sigma = back(\sigma)$; $v = F$
$conf(\sigma)$ is existential $\wedge b \neq d$: $b = ?$; $\sigma = \sigma(b+1)$; $v = ?$

endwhile

When $\sigma = \lambda$ and $b = \lambda$ the computation tree has been searched and there is a value for v. The value of $v = T$ if and only if the input is accepted.

The correctness of the algorithm can be seen because of several invariants. First, the current node in the tree being searched is represented by the path σ. Second, if the search is moving toward the root, then b is the branch just taken and v is the value of the subtree of that branch. The algorithm requires space for storing the path σ and for computing the $conf(\sigma)$ function. But since T is a $t(n)$ time bounded machine, this is $O(t(n))$ space. Hence, the algorithm works in $O(t(n))$ space.

The proof of "NSPACE$(s(n)) \subseteq$ ATIME$(s^2(n))$" is similar to that of Savitch's theorem. Remember, in the proof of Savitch's theorem, we needed to compute

$$\exists E : \text{REACH}(m, C, E, k-1) \wedge \text{REACH}(m, E, D, k-1)$$

There the machine used a stack to implement the recursion in the computation. Here the alternating machine computes the formula in the manner described earlier (See the discussion in proof of the Alternating Space Theorem).

Another point where the proof needs modification is part where the algorithm goes through all $m = 1, 2, 3, 4, 5, \ldots$ till it finds a *safe* m. But this approach takes too much time. Here we need to make m go along powers of 2, (i.e. $m = 1, 2, 4, 8, \ldots$).

Actually, it is not surprising to see that the proof shares so many ideas with the proof of Savitch's theorem. Savitch's theorem almost trivially follows from the Alternating Time theorem. □

An immediate consequence of the Alternating Time theorem is the following corollary.

Corollary 5.3: AP = PSPACE

5.3. A Variant On Alternating Turing Machines

In this section, we describe a variant on alternating Turing machines. This model is more powerful than the ordinary alternating TMs that we have been working with so far. The power of this model comes from the fact that it has *random access* on the input tape.

More concretely speaking, in this model the alternating TM has an input tape, an index tape and a work tape. It has two heads, one each for the index tape and the work tape. It doesn't need to have a head for the input tape. The reason is that the input is read indirectly using the index tape. Its next move depends on the state, symbol under the head on work tape and the symbol on the input tape that is indexed by the index currently written on the index tape.

Random access on the input tape allows us to prove a stronger version of the alternating time theorem. For this model, we can prove the alternating time theorem for any $t(n) \geq log(n), s(n) \geq log(n)$.

5.4. Complements Of Complexity Classes

For any class C, coC is defined as the class of languages whose complements are in C. For example,

Definition 5.4: $\text{coNTIME}(t(n)) = \{\overline{L} : L \in \text{NTIME}(t(n))\}$

Definition 5.5: $\text{coNSPACE}(s(n)) = \{\overline{L} : L \in \text{NSPACE}(s(n))\}$

Some facts:

$$\text{coNP} = \bigcup_k \text{coNTIME}(n^k)$$

$$\text{coNPSPACE} = \text{NPSPACE} = \text{PSPACE}$$

Open Problem 5.6: $\text{coNP} = \text{NP}$?

Theorem 5.7 (Immerman-Szelepscényi): If $s(n)$ is tape constructible and $s(n) \geq log(n)$, then

$$\text{NSPACE}(s(n)) = \text{coNSPACE}(s(n))$$