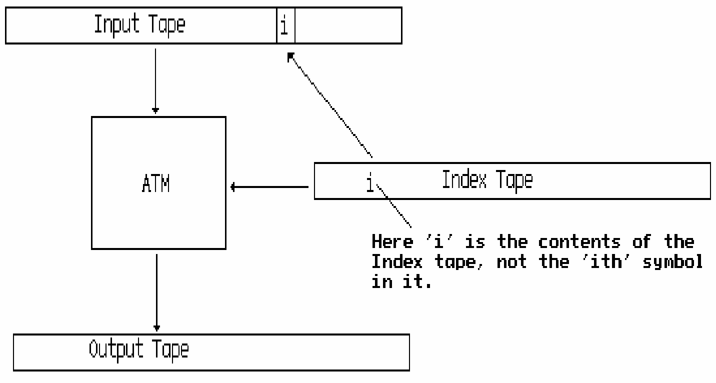


Figure 1: Alternating Turing Machine.

CSC 2401 Introduction to Computer Complexity - Fall 2001
Lecture #20, Tuesday, November 20, 2001
Lecturer: Allan Borodin
Scribe: Francisco J. Estrada

# Lecture #20

## 20.1 Review

In the previous lecture, we talked about Alternating Turing Machines (ATM's). These are Turing Machines that have special existential states, and special universal states. Figure 1 shows a schematic of an ATM, notice that in addition to the input and output tapes the machine has a special index tape, the index tape allows the machine to access any symbol in the input tape in $logn$ steps.

We also discussed the definitions of time, space, and number of alternations for ATM's, which can be generalized in the following way:

$\Sigma_k TIME(t(n)$ ($\Sigma_k ATIME(t(n)$ in the text), which comprises all languages that can be computed in time $t(n)$ on an ATM starting in an existential state, and having $k$ alternations between states; more generally, we can define $\Sigma_{q(n)} TIME(t(n))$ for $q(n)$ a function of $n$ whereas $k$ as above indicates a constant number of alternations .

$\Pi_k TIME$ is defined in a similar way, except that the ATM starts in an universal state.

We can now prove some theorems using these definitions.

## 20.2 Theorem

$\cup_l \Sigma_k TIME(n^l) = \Sigma_k^p$

*Proof:*

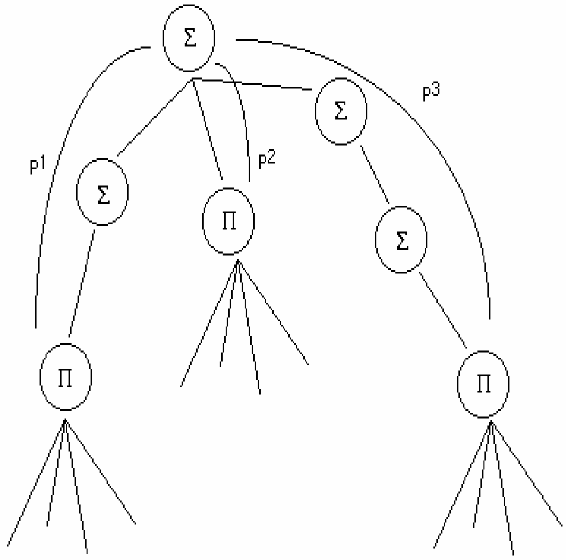


Figure 2: Computation tree for an ATM.

In the $\supseteq$ direction:

Use the $\exists y_1, |y_1| \leq p(n), \forall y_2, |y_2| \leq p(n), ..., Q(n)y_n, |y_n| \leq p(n), R(x, y_1, y_2, ..., y_n)$ characterization of $\Sigma_k^P$. This characterization can be converted to an ATM which uses an existential state to get $y_1$, a universal state to get $y_2$ and so on.

In the $\subseteq$ direction:

Consider the computation tree shown in Figure 2, we can guess the length of the paths required to get to the $\Pi$'s, each of these is the root of a $\Pi_{k-1}$ subtree, hence each of these subtrees is a $\Pi_{k-1}$ language.

Now we can use the characterization $L = \{x | \exists y_1, R(x, y_1)\}$ as in Figure 3.

## 20.3 Theorem

For $t \geq logn$, $ATIME(t) \subseteq DSPACE(t)$ (analogous to $uniform{-}DEPTH(t) \subseteq DSPACE(t)$.

*Proof:*

Consider the computation tree, do a depth first search looking for an accepting subtree

## 20.4 Theorem

$NSPACE(s) \subseteq ATIME(S^2)$ (analogous to $NSPACE(s) \subseteq DEPTH(s^2)$. Note that $ATIME$ has the same time and space compression properties discussed for Turing Machines, not so the circuit size classes.

*Proof:*

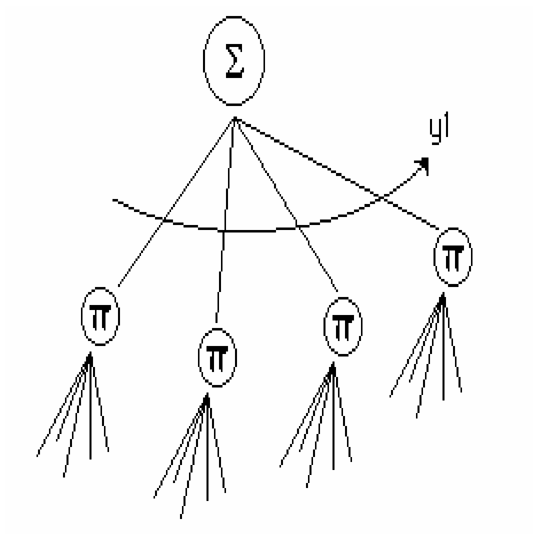


Figure 3: Here $y_1$ indicates the path to the universal state.

By divide and conquer strategy: $path(u,v,l) \equiv \exists w[path(u,w,l/2)$ and $path(w,v,l/2)]$. The ATM can use an existential state to guess the $\exists w$ part, and a universal state to verify the existence of the paths from $u$ to $w$, and from $w$ to $v$.

*Corollary:* $PSPACE = \cup_l ATIME(n^l)$.

## 20.5 Theorem

$ATIME(logn) = uniform - NC^1$.

in general:

$ATIME - SPACE(log^k n, logn) = uniform - NC^k$.

## 20.6 Theorem

$ASPACE(s(n)) = DTIME(2^{O(s(n))})$.

*Proof:*

In the $\subseteq$ direction:

We use the fact that there are $2^{O(s(n))}$ configurations. If we do a breadth first search looking for accepting subtree, and prune a tree whenever there is a repeated configuration, we'll need at most $2^{O(s(n))}$ space to keep track of the breadth first search.

In the $\supseteq$ direction:

Consider the accepting computation of a deterministic $2^{c \cdot s(n)}$ Turing Machine M:

$\alpha_0 \mapsto \alpha_1 \mapsto \ldots \mapsto \alpha_{c \cdot s(n)} = \alpha_{t(n)}$.

$|\alpha_i| = t(n)$

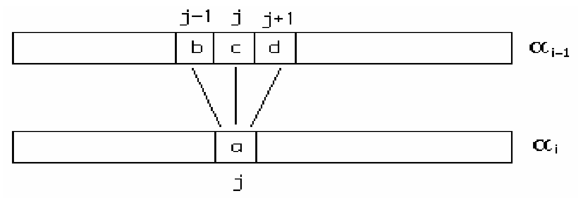


Figure 4: Verification that the $j$th symbol in configuration $i$ is equal to $a$.

Say M has an alphabet $\Gamma$, then $\alpha_i \in (\Gamma \cup (\Gamma \times Q))^*$; we can refer to any particular symbol in any configuration $\alpha_i$ with a $\log t(n) = c \cdot s(n)$ bit string index.

We will use the existential configuration $(i, j, a)$ to denote a configuration whose goal is to verify that the $jth$ symbol in configuration $\alpha_i$ is $a$. Both $i$ and $j$ have $c \cdot s(n)$ bits.

To do this, we have to guess $b, c$, and $d$ in the previous configuration $\alpha_{i-1}$, where $b$ is at position $j-1$, $c$ is at position $j$ and $d$ is at position $j+1$. Recall that the next configuration can be fully determined by at most three symbols of the previous configuration. We then use a universal state to verify that the choice of $b, c, d$ are corrrect and that the symbol $a$ at position $j$ in configuration $\alpha_i$ represents a correct move of the Turing machine. whether it is equal to $a$. Figure 4 depicts this.

*Corollaries:*

$ASPACE(logn) = P$.

$ASPACE(n) = E = \cup_c DTIME(c^n)$.

## 20.7 Some provably hard natural languages

Recall that for Regular Expressions we have:

$\Sigma = \{a, b, c, ...\}$, any $a \in \Sigma$ is a Regular Expression.

If $R_1$ and $R_2$ are Regular Expressions, then $(R_1 \oplus R_2)$ is also a Regular Expression and represents $\mathcal{L}(R_1 \oplus R_2) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$.

Similarly, we have for concatenation that: $\mathcal{L}(R_1 \odot R_2 = \mathcal{L}(R_1) \cdot \mathcal{L}(R_2)$.

$R \in \Gamma^*$, and $\Gamma = \Sigma \cup \{\oplus, \odot, \otimes, (, )\}$.

We can define TRE (Total Regular Expression) as:

$TRE = \{R | \mathcal{L}(R) = \Sigma^*\}$, R is a Regular Expression over $\Sigma$. In a similar way we can define $\overline{TRE} = \{R | \mathcal{L}(R) \neq \Sigma^*\}$.

$\overline{TRE}$ (or $TRE$) is complete for PSPACE wrt. $\leq_{log-space}$, more precisely, let $l$ be in $NSPACE(s(n))$, then $L \leq_{log-space} \overline{TRE}$. Even more precisely, let $L = \mathcal{L}(m)$ where $M$ is a $s(n)$ space non-deterministic Turing Machine; then for every $x$, $x \mapsto R_{m,x} : x \in \mathcal{L}(m)$ iff $R_{m,x} \in \overline{TRE}$.

$|R_{m,x}| = C_m \cdot [|x|^2 + s(n)]$.

Also, $\overline{TRE} \in NSPACE(s(n))$.

*Claim:*

This tells us that $\overline{TRE}$ can not be done in less than $\sqrt{n}$ space ($n =$ the length of the input regular expression). Recall that it had previously been stated that nothing has been found so far in NP that takes more than log-space. This implies that in some sense $\overline{TRE}$ is inherently hard.

Furthermore, if we extend the regular expression language to include the intersection of languages, we get $TERE$, or Total Extended Regular Expression. It can be proven that $TERE$ is complete for exponential space. This is an example of a natural language which is provably difficult (exponential time and space) to do.