18.405J/6.841J Advanced Complexity Theory	February 3, 2000
Lecture 2	
Lecturer: Daniel A. Spielman	*Scribe: Adam Smith*

This lecture begins the study of the possibly trivial though certainly interesting classes that lie between P and PSPACE. We begin with the Immerman-Szelepcsényi theorem, which tells us in some sense that the study of non-determinism and logical quantifiers is less interesting in the a space-restricted universe than in a time-restricted one. The main part of the lecture is devoted to defining alternating Turing machines and proving some basic facts about them. This sets up the introduction of the polynomial hierarchy in the next lecture.

Theorem 1 *For* $f(n) \geq \log n$ *we have* $\mathrm{NSPACE}(f(n)) = \mathrm{coNSPACE}(f(n))$

Proofs of this theorem can be found in both Sipser and Papadimitriou's books (in Sipser, only the special case NL = coNL is proved, but the generalization is not very difficult).

Remark In the theorem above and in all theorems below, we assume that the functions f we are dealing with are "proper complexity functions" à la Papdimitriou (p. 140), that is they are non-decreasing and constructible in time $O(n + f(n))$ and space $O(f(n))$.

1 Alternating Turing machines

1.1 Non-determinism

Recall that there are different ways to define NP. On the one hand, we have the "short proof" definition, which states that NP languages are ones for which membership is easy to verify, given a short (polynomially bounded length) certificate (a.k.a. proof, witness, stool-pigeon, etc):

$$L \in \mathrm{NP} \iff \begin{cases} \exists A \in P,\ p(\cdot) \in \textsc{poly} \text{ such that} \\ \quad x \in L \iff \exists w :\ |w| \leq p(|x|) \text{ and } (x, w) \in A \end{cases}$$

where POLY is simply the set of all polynomial-bounded functions $\text{POLY} = \bigcup_{k>0} O(n^k)$. Equivalently, we can say that $L \in \text{NP}$ iff there is some poly-time non-deterministic TM which accepts L.

Of course, for the latter definition to make sense we must define what we mean by a non-determinstic TM. The simplest way to do this is to think of a normal Turing machine which has "existential states" ($\exists$). When the machine enters such a state, it may continue its computation in two different ways (imagine for example writing either a 1 or a 0 at the tape head's current location). Thus we can imagine a tree in which the nodes are configurations of the machine[1]. The root represents the initial configuration; nodes with two children will have the machine in an existential state, nodes with one child will have the machine in a regular computation state, and leaves correspond to accept or reject states.

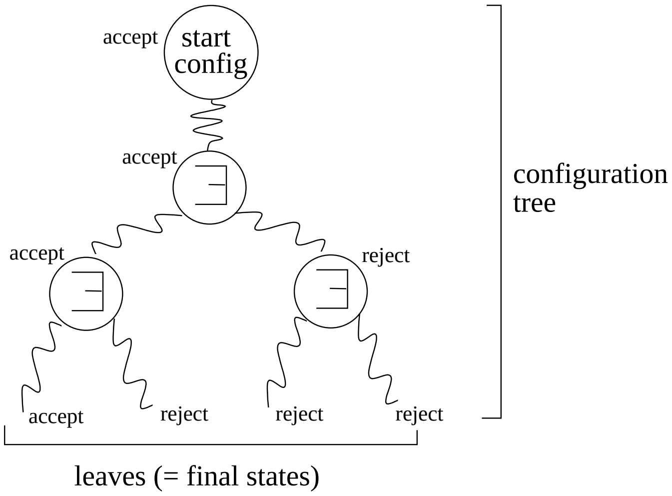

Figure 1: The configuration tree of a nondeterministic TM.

When all branches of a machine's computation tree terminate, we define the output of the machine to be the logical OR of the output at the leaves: it accepts iff a single computation path ends in an accept state. Graphically, we can imagine the tree to be a formula consisting only of OR gates. If there is an accept (true) value at any leaf, it will travel up to the root.

[1]Depending on the semantics of existential states, the tree may actually be a DAG with a single source. It is often simpler to force the computation graph to be a tree by including the choices made so far in the "configuration" of the machine. One has to be acreful with this approach when one delas with space-bounded machines as this can change the size of the representation of a configuration significantly.

In a similar vein, we can define co-non-deterministic machines to be ones with special states which are "universal" (labeled $\forall$). As before, we imagine the machine choosing one of two computation paths and thus generating a tree. However, we now say a machine (all of whose paths terminate) accepts only if *every* path ends in an accept state. The gates in the corresponding formula will now be AND gates, and it is the "reject" values which will travel up the tree to the root.

1.2 Alternation

Given the previous discussion, it's natural to consider machines that can have both between existential and universal states. These are called alternating Turing machines (ATM's). As before, we can look at the computation tree of such a machine. Nodes at which the machine is in an existential state are labeled $\exists$ and those corresponding to a universal state are labeled $\forall$.

Given a ATM and an input on which all branches terminate, we can associate a boolean formula to this tree by replacing $\exists$ nodes with OR gates, $\forall$ nodes with AND gates, and collapsing chains of regular nodes into wires. We say that the ATM accepts its input if this formula is true when each of the leaves is assigned 1 when it corresponds to an accept state and 0 otherwise.

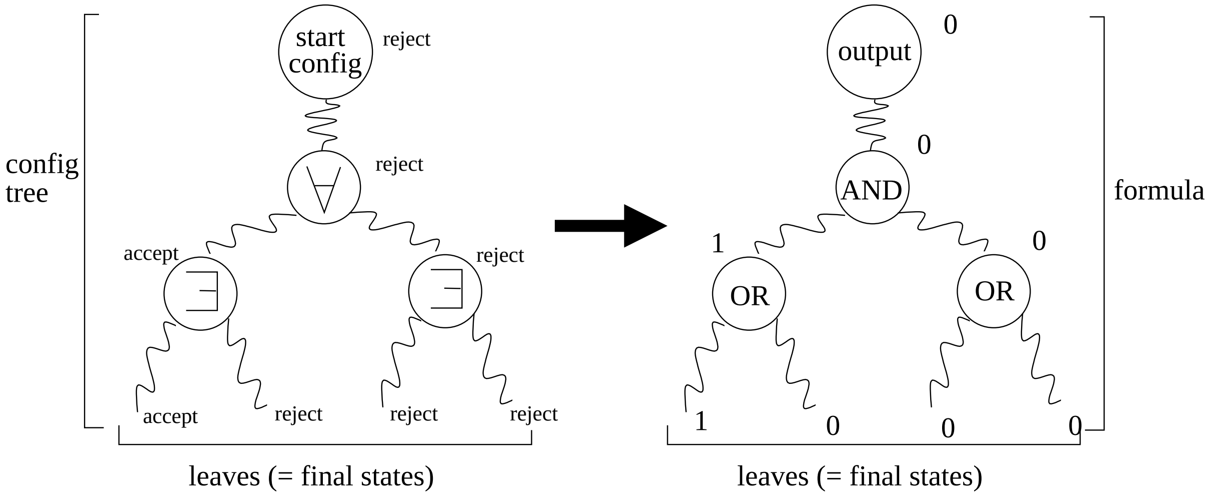

Figure 2: The configuration tree of an alternating TM and the corresponding formula.

There are several natural complexity classes related to ATM's.

$$
\begin{aligned}
\mathrm{ATIME}(f(n)) &= \{L : L \text{ is accepted by an } \textsc{atm} \text{ such that all branches} \\
&\qquad \text{terminate in time } O(f(n))\} \\
\mathrm{ASPACE}(f(n)) &= \{L : L \text{ is accepted by an } \textsc{atm} \text{ such that all branches} \\
&\qquad \text{branches terminate and none uses more than } O(f(n)) \text{ space}\} \\
\mathrm{AP} &= \bigcup_{k>0} \mathrm{ATIME}(n^k) \\
\mathrm{APSPACE} &= \bigcup_{k>0} \mathrm{ASPACE}(n^k)
\end{aligned}
$$

For example, TQBF naturally fits into AP and so we immediately get PSPACE $\subseteq$ AP. Another example of an AP problem is

MIN-BOOLEAN-FORMULA $= \{f : f$ is a formula and no smaller equivalent formula exists$\}$

Given the size ordering $<$ on formulas, this can be rewritten

$$\{f : (\forall f' < f)(\exists x)\ f(x) \neq f'(x)\}.$$

An AP algorithm for the problem consists of universally guessing a smaller circuit f', then existentially guessing x and accepting iff $f(x) \neq f(x)$.

2 Relations to deterministic classes

We will prove two results:

Theorem 2 *For $f(n) \geq n$ we have* $\mathrm{ATIME}(f) \subseteq \mathrm{SPACE}(f) \subseteq \mathrm{ATIME}(f^2)$

Theorem 3 *For $f(n) \geq \log n$ we have* $\mathrm{ASPACE}(f) = \mathrm{TIME}(2^{O(f)})$

We break the proof into four lemmas.

Lemma 4 *For $f(n) \geq n$ we have* $\mathrm{ATIME}(f) \subseteq \mathrm{SPACE}(f)$.

Proof We first prove that $\mathrm{ATIME}(f) \subseteq \mathrm{SPACE}(f^2)$ and then show that this bound can in fact be tightened. Let $L \in \mathrm{ATIME}(f)$ and let M be an $\mathrm{ATIME}(f)$ machine deciding L. We construct a space $O(f^2)$ machine that M' simulates M. On input w, m' performs a depth-first search of the computation tree of M on w, to determine which nodes in the tree are accepting. M' accepts iff the root is accepting.

How much space is required by M'? For each level in the recursion, M' stores a description of the node—$O(f)$ bits since M takes $O(f)$ time and hence at most $O(f)$ space—as well as a single bit indicating which child is currently being investigated. Since the recursion stack can only be as deep as the tree itself, we see that M' requires at most $O(f^2)$ space.

To get the space usage down to $O(f)$, note that we need only store the *current* configuration. In the recursion stack we can get away with only storing the single bit describing the path chosen at each level. This is because when we decide to backup the tree a level, we can simply recompute the needed configuration starting from the root (NB: This idea comes back time and again when computing with limited space: never store anything you can recompute). The total space is required is thus the size of a configuration plus one bit per recursion level, which is $O(f)$. ■

Lemma 5 *For* $f(n) \geq n$ *we have* $\mathrm{SPACE}(f) \subseteq \mathrm{ATIME}(f^2)$.

Proof [This follows the same ideas as the proof that TQBF is PSPACE-complete (also similar to proof of Savitch's theorem).]

Let $L \in \mathrm{SPACE}(f)$ and let M be a $\mathrm{SPACE}(f)$ machine deciding L. WLOG, there exists a unique accepting configuration, in which M erases all work tapes and stops in its accept state. Then given an input w of length n, the problem of deciding whether M accepts can be reduced to solving st-connectivity in the configuration graph of the machine, where s is the start configuration on input w and t is the accepting configuration. If we define $\mathrm{FromTo(c_1, c_2, t)}$ to be the predicate "M can go from configuration c_1 to c_2 in t or fewer steps", then we must find the value of $\mathrm{FromTo(c_{start,w}, c_{accept}, 2^{df(n)})}$, for some appropriate constant d. But notice that

$$\mathrm{FromTo(c_1, c_2, t) = (\exists c_m)\left(FromTo(c_1, c_m, \lceil \frac{t}{2} \rceil) \wedge FromTo(c_m, c_2, \lfloor \frac{t}{2} \rfloor)\right)}$$

Moreover, $\mathrm{FromTo(c_1, c_2, 1)}$ is always easy to calculate. Thus an alternating machine may recursively decide the formula by existentially guessing c_m and universally choosing either the left or right subformula to evaluate. The recursion bottoms out when $t = 1$ is reached. The number of recursive levels is $df(n)$ and each one requires $df(n)$ existential guess (to select c_m) and one universal one. Thus, the total time is $O(f^2)$ (the bottom level of the recursion takes only $O(f)$ steps and so doesn't change the total order). ■

Remark The previous two lemmas prove (again) that $AP = PSPACE$. We can also extract from the second proof that TQBF is $PSPACE$-complete.

Lemma 6 *For $f(n) \geq \log n$ we have* $\mathrm{ASPACE}(f) \subseteq \mathrm{TIME}(2^{O(f)})$.

Proof Let $L \in \mathrm{ASPACE}(f)$ and let M be a ASPACE(f) machine deciding L. Let d be a constant such that M has no more than $2^{df(n)}$ distinct configurations on input w (such a d exists since $f(n) \geq \log n$). Then we can construct a deterministic machine M' to simulate M as follows. On input w of length n, write down the entire configuration graph of M on w. Now proceed to mark nodes in the graph: on each pass through the graph, find an unmarked node with marked children, or a leaf. Then mark that node appropriately (if it's a leaf, just check if it accepts or rejects. Otherwise, base the answer on its children's markings and whether the node corresponds to an existential or universal state). Stop when the start node of M on w is marked, and accept or reject accordingly.

This process takes time polynomial in the size of the written representation of the graph. This representation itself takes $df(n)2^{df(n)} \in 2^{O(f)}$. Moreover, any polynomial of a function in $2^{O(f)}$ remains in $2^{O(f)}$; hence the total time is $2^{O(f)}$. ■

Lemma 7 *For $f(n) \geq \log n$ we have* $\mathrm{TIME}(2^{O(f)}) \subseteq \mathrm{ASPACE}(f)$.

Proof Let $L \in \mathrm{TIME}(2^{O(f)})$ and let M be a $2^{df(n)}$-time machine deciding L. WLOG, assume that M accepts by moving its head to the leftmost end of the tape. Consider the computation tableau of M on input w of length n. This a $2^{df(n)} \times 2^{df(n)}$ grid, each symbol of which represents either the contents of a tape cell or *both* the contents of a tape cell *and* the state of the machine (this is for the one cell per row that has the tape head over it).

Now we construct an alternating machine M' which verifies the tableau as follows. Since we cannot store the entire tableau, the machine stores only a pointer into the tableau. Recall that a cell's contents depend only on the three cells immediately above it in the tableau (except for top row cells, which depend only on the input).

The machine M' starts in the bottom lefthand corner, since if the machine head is there and that cell is correct, then the machine M would indeed have accepted w. M' knows what the contents of that bottom cell should be.

Now given a cell and a guess as to its contents, the machine runs the following procedure:

1. If the cell is on the top row, accept iff the cell's guessed contents are consistent with the input.

2. If the cell is in a lower row:

(a) universally choose one of the three predecessor cells

(b) existentially guess its contents, check that the successor's guessed contents follow from predecessor's guessed contents by a legal transition

(c) recursively check the predecessor's guessed contents.

Note that since this procedure is tail recursive, the machine doesn't have to remember what path it took as it recursed up through the tableau, only where it is at a given moment (the tableau is never written down). Thus the total space requirements of the machine M' are $O(f)$. ■