6.841/18.405J: Advanced Complexity **Monday, February 24, 2003**

Lecture 5

Instructor: Madhu Sudan *Scribe: Daniel Zaharopol*

1 Fortnow's time/space lower bound for SAT

Fortnow gave a lower bound for either time or space for an algorithm to solve SAT, though the proof doesn't say which actually has the bound [F00].

To be more specific, consider what we "believe" about complexity classes. We believe PH does not collapse, which in turn would imply that NP $\neq$ P. This, in turn, would imply both that SAT does not have a linear-time algorithm, and that it does not have a log space algorithm.

We will prove that, given two TMs, one cannot solve SAT in linear time while the other solves it in logarithmic space. This result is not trivial; it was open until 1997, when proven by Fortnow. The main idea of the proof, surprisingly, is to use alternation to derive a contradiction.

Let's state the theorem more formally.

Theorem 1. $SAT \in L \Rightarrow \exists \epsilon > 0$ such that $SAT \notin \text{TIME}(n^{1+\epsilon})$

We prove this, as usual, by contradiction: assume $SAT \in L$ and $SAT \in \text{TIME}(n^{1+\epsilon})$.

Intuitively, we can see where the contradiction comes from by looking at the "power" of alternation given either alternative. Suppose $SAT \in \text{DTIME}(n^{1+\epsilon})$. This would imply, in a way, that alternation was not powerful. For example, it would give us that $\Sigma_i^P \subset P \forall i$.

On the other hand, Savitch's theorem tells us that alternation is powerful for small space computation. But since $SAT \in L$, this would tell us that alternation was powerful.

In principle, this is a contradiction. We now quantify it.

The contradiction will come from the Time Hierarchy Theorem (we could also derive it, with some more work, from the Space Hierarchy Theorem). Essentially, we will derive that $\text{TIME}(T(n)) \subseteq \text{TIME}(T^{1-\delta})$, for some $\delta > 0$. We will do this by finding some intermediary inclusions.

1.1 Cook's Strong Theorem

There is a stronger version of Cook's Theorem which says that languages in $\text{NTIME}(T(n))$ reduce to SAT on formulae of length $T(n) \log T(n)$. We don't know that this bound is tight (after all, if $P = NP$, then they reduce to formulae of length 1).

Theorem 2 (Cook's Strong Theorem[C88]). *let M be a nondeterministic Turing machine running in time $t(n)$. There is a reduction running in $O(t(n) \log t(n))$ time and $O(\log t(n))$ space that maps inputs x of length n to formulae ϕ of size $O(t(n) \log t(n))$ such that*

$$x \in L \iff \phi \in SAT.$$

We can, of course, also apply this result to deterministic time to get that $\text{TIME}(T(n)) \subseteq SAT$ on length $T \log T$. Then, given our assumption about SAT, $\text{TIME}(T(n)) \subseteq \text{SPACE}(O(\log T))$. This is our first inclusion to derive our contradiction.

Lemma 1. $\text{SPACE}(s) \subseteq \text{ATIME}[2i, s2^{\frac{s}{i}} i]$

Proof. We prove this in a similar manner to Savitch's theorem.

Consider the computation of a SPACE(s) TM; we can write the states of its tape(s) in a table with width $s(n)$ and height 2^s.

In the proof of Savitch's Theorem, we continually divided the computation in half, and examined each half to see if it was a valid computation.

In this proof, we divide each time into w pieces and guess the w intermediate points. Each time we divide out, we perform two alternations.

Each time we branch out, we guess the next computation universally. This requires writing down each computation (of length s), and doing this i times if we perform $2i$ alternations in total. Then the total time taken is wsi.

Let $w = 2^{\frac{s}{i}}$, so that we continue to split the table into w pieces each time we go down a level.

Then the total time taken is $wsi = s2^{\frac{s}{i}}i$, with $2i$ alternations.

This gets us another inclusion, so we've built up to $\text{TIME}(T(n)) \subseteq \text{SPACE}(c_1 \log T) \subseteq \Sigma_{2i}^{\text{TIME}(T^{(c_2/i)})}$, for constants c_1 and c_2.

1.2 The Final Step

Lemma 2. $i\exists TQBF \in \text{TIME}(n^{(1+\epsilon)^{3i}})$

Proof. We proceed by induction.

Consider $2\exists TQBF$. A problem is of the form $\exists x_1 \ldots x_n \forall y_1 \ldots y_m \phi(x_1, \ldots, x_n, y_1, \ldots, y_m)$. But $\forall y_1 \ldots y_m \phi$ is in $\text{co}-\text{NTIME}(n)$ and therefore in $\text{TIME}(n^{1+\epsilon})$.

Then our entire formula is in $\text{NTIME}(n^{1+\epsilon})$; from Cook's Strong Theorem we know that this reduces to a satisfiability problem in SAT of length $n^{1+\epsilon}\log n$, which is in $\text{TIME}((n^{1+\epsilon}\log n)^{1+\epsilon})$. This is clearly in $\text{TIME}(n^{(1+\epsilon)^3})$.

We now proceed by induction, using the same argument, and get that $i\exists TQBF \in \text{TIME}(n^{(1+\epsilon)^{3i}})$.

This gives us that $\Sigma_{2i}^{\text{TIME}(T^{(c_2/i)})} \subseteq \text{TIME}((T^{(c_2/i)})^{(1+\epsilon)^{6i}})$.

Then $\text{TIME}(T^{(c_2/i)(1+\epsilon)^{6i}}) \supset \text{TIME}(T)$ by our chain of inclusions.

This is a contradiction for proper choices of i and ϵ; for example, if we choose $i = 10c_2$ and $\epsilon = 1/(60c_2)$.

Then we conclude that initial theorem about SAT was correct.

2 Randomness

Here we consider randomness as a resource, and look at the eight complexity classes of interest when considering randomness.

2.1 Augmented Turing Machines

We've already seen various "augmented Turing machines," of the form $M(\cdot,\cdot)$. The idea is that each one takes a "real input" — a string we wish to accept or reject — and an "auxilliary input," a string that somehow helps the calculation. We generally assume $M(x,y)$ operates in polynomial time, and that y is polynomial in the length of x.

For example, nondeterminism asks if, given x, there exists y such that $M(x,y)$ accepts.

Non-uniformity says that, for all n, there exists y_n such that, for all x with $|x| = n$, $x \in L$ if and only if $M(x,y)$ accepts.

When considering randomness, y is not related to x at all, but captures some physical process. We ask if randomness can make computations faster.

2.2 Randomness

We'd like a machine M for a language L to accept x with a high probability if $x \in L$, for the range of auxilliary inputs y.

There are several kinds of errors we can accept. A "false positive" occurs when $x \notin L$ but M accepts on some y. A "false negative" occurs when $x \in L$ but is rejected on some y.

We can require either that there are no false positives, no false negatives, none of either, or we can accept either. This gives us four natural complexity classes.

We get the other four by considering machines that operate either in polynomial time or logarithmic space.

For logarithmic machines, we stipulate that they can read y only once (they must store what information they need in their own memory).

2.3 Complexity Classes

In all of the following, the final "P" stands for "polynomial."

- ZPP — zero-error probabilistic
- RP — Randomized
- coRP — Complement-randomized
- BPP — Bounded-error probabilistic

We say that $L \in \text{ZPP}$ if there exists a TM M such that $\Pr_y[M \text{accepts} x] = 1$ if $x \in L$, $\Pr_y[M \text{accepts} x] = 0$ if $x \notin L$. This seems to give no benefit from being probabilistic, so we allow one more output for this class only: "I don't know."

RP is in a sense analogous to NP. With NP, we know that $x \in L$ if there exists a y such that $M(x, y)$ accepts. A language is in RP, meanwhile, if there exists a TM M such that $\Pr_y[M \text{accepts} x] \geq \frac{1}{2}$ if $x \in L$, and $\Pr_y[M \text{accepts} x] = 0$ if $x \notin L$.

coRP is just the complementary class, defined as the complement of languages in RP. It's easy to see that this is equivalent to there existing M such that $\Pr_y[M \text{accepts} x] = 1$ if $x \in L$, and $\Pr_y[M \text{accepts} x] \leq \frac{1}{2}$ if $x \notin L$.

As an exercise, one can prove ZPP = RP = coRP.

There more interesting class for study is BPP. Here, we say that $L \in BPP$ if there exists a TM M such that $\Pr_y[M \text{accepts} x] \geq \frac{2}{3}$ if $x \in L$, and $\Pr_y[M \text{accepts} x] \leq \frac{1}{3}$ if $x \notin L$. It turns out that these bounds can be anything as long as they differ appreciably; for example, we could even have $\frac{2}{3} + \frac{1}{|x|}$ and $\frac{2}{3}$ as our bounds.

References

[C88] S. Cook. Short propositional formulae represent non-deterministic computation. *Information Processing Letters*, 26:269-270, 1988.

[F00] L. Fortnow. Time-space tradeoffs for satisfiability. *Journal of Computer and System Sciences*, 60(2):337-353, April 2000.