6.841/18.405J: Advanced Complexity Theory February 13, 2002

Lecture 3

Lecturer: Madhu Sudan *Scribe: Emily Marcus*

1 Alternation

Last time, we introduced the idea of an Alternating Turing Machine (ATM). This machine has two special "nodes" - the existential state ($\exists$) and the for-all state ($\forall$). If the machine entered a for-all node, it must accept on all of its outgoing branches of that node in order for the node to have accepted. If it enters an existential state, it must have at least one outgoing branch which has accepted.

The computation resources which are of interest in the ATM model are time, space, and total number of alternations. An alternation occurs when an ATM leaves the existential state and enters a for-all state or vice-versa.

Alternation is not actually an "implementable" or tangible property in the way that time and space are. In this respect it is more like non-determinism - you can't actually build a machine which uses alternation or non-determinism except by simulation. However, considering each of these resources gives rise to interesting classes of problems. In addition, analyzing alternation can give us more insight into the power of the more tangible properties of space and time.

The big question for today is how alternation relates to the resources we have already considered, namely time and space. Is an ATM more powerful than a TM?

In thinking about these questions, we will prove two main results, which are

- ATIME(t) = SPACE(poly(t))
- ASPACE(s) = TIME($2^{O(s)}$)

2 Alternations vs Space

Lemma 1: ATIME(t) $\subseteq$ SPACE(O(t))

> To see intuitively why this is the case, consider the tree representing the computation of some ATM A on some input. This tree will have nodes which are existential and nodes which are for-all. But in order to simulate this on a regular TM M, we only need to know where we are with respect to these nodes. The ordinary computations can be simulated by M trivially. Therefore, if M is augmented with an extra tape, which preserves a stack of the special nodes which have been encountered while simulating A, M can simulate A completely.
>
> Because A runs in ATIME(t), no branch of its computation can be deeper than O(t). But M only needs to simulate M one branch at a time, so M only requires SPACE(O(t)).

Before proving the other half of the relationship between SPACE and ATIME, which requires constructing an alternating TM, it is useful to describe intuitively how programming such a machine works. Most of the programming is similar to programming an ordinary TM. However, the two special states are worth describing.

- The existential state effectively allows you to "guess" something. So, for example, if you "guess" subsequent bits, you can effectively pull out a number which satisfies some condition.
- The for-all state allows you to verify that something is true.

With this, we can prove the other half of the equality.

Lemma 2: SPACE(s) $\subseteq$ ATIME(s^2) (this should, by now, be unsurprising)

In order to see why this is true, we must first consider what a computation in SPACE(s) looks like.

There are two assumptions we can make to simplify things slightly. First, if a computation has more than 2^s configurations, it has repeated at least one configuration and is therefore unlikely to terminate. Second, we can also assume without loss of generality that the final accepting configuration is unique.

Then, each configuration needs to contain

- the positions of all read/write heads
- the complete contents of the work tape
- the state that the machine is in

These somewhat general assumptions about the configurations are sufficient for this lemma. Later, when we are considering time instead of space, we will need to be much more precise about the configurations.

We also need to assume that s($|x|$) $\geq \log |x|$.

From here, the argument follows along lines similar to Savitch's theorem, with the s^2 term coming from the same type of argument.

So now the question we want to ask is whether our Turing Machine M can go from configuration C_0 to configuration C_1 in time 2^s.

We'll define an alternating TM A which will, on input $< C_0, C_1, t >$, tell us whether M can go from C_0 to C_1 in time t.

1. Bit by bit, guess a configuration C_3 such that M can go from C_0 to C_3 in time $\frac{t}{2}$ and can go from C_3 to C_1 also in time $\frac{t}{2}$. This "guess" can be made by using the existential state, as described above.
2. Verify, using a for-all state, that M(C_0, C_3, $\frac{t}{2}$) and M(C_3, C_1, $\frac{t}{2}$) accept.

Inductively, this method takes space $s(\log t)$, and since initially t is at most 2^s, the whole thing requires at most s^2 space. Therefore SPACE(s) $\subseteq$ TIME(s^2).

In total now, we have shown that SPACE(s) = TIME(poly(s)).

3 Alternations vs Time

Lemma 3: ASPACE(s) $\subseteq$ TIME($2^{O(s)}$)

This lemma is not easy to prove.

The basic idea is that we can create a machine which given a tree representing a computation, walks through the tree and figures out what happens at each node, and then propagates that information through the tree to figure out whether the entire computation accepts or not.

The problem with this simple idea, however, is that not all of the branches necessarily need to be resolved as being either accepting or rejecting in order for the whole computation to accept or reject.

So, in walking down the tree, if we are at a for-all node, we can check if that is accepting by checking every outgoing edge. If we are in an existential state, we need to check whether either of the edges leads to an accepting computation.

But what if there are repeated states on the way down?

The key observation is that if an existential node ends up accepting, it can't be accepting *because* of a branch below it which does not halt. So, if a configuration is ever repeated on a given branch, that branch can just be labelled as rejecting without any further inspection.

You can then work your way back up the tree, until the root node has a label; this label represents whether the entire computation accepts or rejects.

The tiem required to do all of this is $2^{O(s)}$ since examining each node requires a polynomial amount of time and there are $2^{O(s)}$ of them.

Lemma 4: TIME($2^{O(s)}$) $\subseteq$ ASPACE(O(s)).

Here we need to do a construction similar to that in Lemma 2, i.e. inspecting the sequence of configurations of a TIME($2^{O(s)}$) machine M. But in contrast to Lemma 2, the objective here is to use very little space in order to do this analysis. To allow this to be done in very little space, we'll set up the configurations so that they can be analyzed in a very localized manner.

For the i^{th} entry of the configuration in the t^{th} configuration of the computation, the following information will get stored.

- the contents of the i^{th} cell of the tape
- whether or not the read-write head is in this position
- if the answer to the above is "yes" then the cell also contains the state of the machine.

Why is this careful encoding useful? Because then we can easily verify local consistency by comparing cell (t, i) to the three above it, namely (t-1, i-1), (t-1, i), and (t-1, i+1).

So how can we use this idea of local consistency to build an ATM which checks for global consistency based on this local consistency property? The idea of recursion, used in proving Lemma 2, doesn't help us here, because there is no longer enough available space to write down an entire configuration.

Instead, what we'll do is build an ATM M(t, i, σ), which accepts its input if the original machine, M, on input x, could have σ as the contents of cell i in the configuration which arises after time t.

M(t, i, σ) behaves as follows

1. Guess r_1, r_2, and r_3 as the contents of the three cells above cell (t, i).
2. Check that r_1, r_2, and r_3 are locally consistent with σ.
3. Verify, using a for-all, that M($t-1$, $i-1$, r_1), M($t-1$, i, r_2), and M($t-1$, $i+1$, r_3) all accept.

In doing this, there isn't any need for any extra space in order to keep track of things. You only need to write down enough information to spawn a subroutine call, and it isn't necessary to preserve the input. So while this is extremely slow, it requires very little space.

It is also necessary to check some of the boundary cases (for example, confirming that the final state is an acceptance state).

In total now, we have shown that

- ATIME(t) = SPACE(poly(t))
- ASPACE(s) = TIME($2^{O(s)}$)

4 Philosophy of ATIME as a game

Philisophically, what is ATIME?

Fundamentally, ATIME problems represent a game. There are two players, one of them called "there exists" (referred to here as E), and the other one called "for all" (called A here).

Given a language L and an input x, E is trying to prove that $x \in L$ and A is trying to prove that $x \notin L$.

So, as the game proceeds, the computation is left to follow its own course. But every time the computation enters an existential state, it stops and E is allowed to decide in what direction the computation should proceed. Similarly, A gets to choose every time a for-all state is entered.

At the end of the game, you are able to tell whether A or E has won, since watching this game will convince you that one or the other of them is correct. However, before the game finishes, poly-time computation is not enough to look into the future and and predict which of the players will win.

So, philisophically speaking, ATIME(poly) can simulate two-player games and the computation itself can be viewed as a two-player game.

But since PSPACE = ATIME(poly), this implies that there are a number of games which are PSPACE-complete. Examples include Checkers, Go, and Generalized Geography [1].

Why isn't chess a PSPACE-complete game? It has a somewhat obscure rule that if a board configuration is ever repeated, the game is automatically a draw. So you can't tell just by inspecting the current state of the computation whether the game is over or not. This makes it fundamentally different from the other games named above.

Question: Is it possible to maintain a counter which is incremented for every move, and then to declare the game to be a draw if the counter gets high enough that a configuration was necessarily repeated?

No, this isn't quite sufficient, because a repeated configuration can have different outcomes if one of the two players decides to act differently the second time around.

Question: Is this still an issue if the two players are playing optimally?

We're only looking at things locally at this point. This is an intricate technical issue which we'll try to cover in more detail later.

Question: (A third question which the scribe couldn't hear; sorry)

So in general, if a game allows you to verify local validity (i.e. watching the moves and knowing that each one is legal) and you are able to inspect only the current state and know whether someone has won, the game is likely to be PSPACE-complete.

What about a game like Mah-jong? This can be played as a one-player game. But because the intial board layout is randomized, the randomness allows it to have the same complexity as a game like Go or Checkers. The details of this will be explored later in the semester.

5 Philosophy of ATIME/ASPACE as conveying information

Supoose you are trying to decide whether to vote for candidate A or B in an election. There are varying levels to which the candidates might be allowed to try to convince you to vote for them. A few of these scenarios are

1. There is no campaigning allowed. You decide which candidate to vote for based on what you already know about.

2. Each candidate is allowed a single advertisement of fixed length, which they prepare independently.

[1] A game in which players take turns naming geographical locations. Each location may be named at most once, and the first letter of the name of the location must be the last letter of the name of the previous location. A player wins when the other player cannot think of a location to name.

3. The candidates are allowed to have a full-fledged debate, which voters can watch all of.

We can't really say which of these is "better", but we can try to understand the implications of each by figuring out what they allow you to compute.

Here a voter is considered to be a polynomial-time computation. So a voter can reach conclusions in P on his or her own. The candidates in tis scenario are analagous to the game players of before. The protocol for advertising is giving the voters more "knowledge" if it gives proofs that the voters are able to verify of a more complex language L.

In this context, we can analyze how much "knowledge" each of the above scenarios would convey to the voters.

1. In the case of no advertising, the voters can only recognize L $\in$ P. This is pretty much the trivial case.
2. In the case of a single advertisement, you can recognize L $\in$ NP $\bigcup$ coNP. More generally, if the candidates are allowed to produce n ads and are allowed to see the opponents previous ads before making theirs, you get the additional resource of n alternations.
3. Full-fledged debate with as much time as the candidates need gives all of PSPACE. The candidates are able to convince the voters that one of them is correct because they will present a complete argument which the voter has enough computational power to verify, even though the voter on his/her own would not have been able to come up with the argument.

So, from a computational point of view, debates are far more powerful than either of the other two methods of advertising because they are able to convince voters of much more complex languages.

6 TQBF : A PSPACE-complete problem

TQBF = totally quantified boolean formulae

If you have n^2 boolean variables, $x_{1,1} \cdots x_{n,n}$ and define $X_1 = x_{1,1}x_{1,2} \cdots x_{1,n}$ etc, then TQBF = $\{\phi \mid \phi(X_1, X_2, \cdots X_n)$ is a 3-cnf boolean formula such that $\exists X_1 \forall X_2 \exists X_3 \cdots Q_n X_n =$ true $\}$.

Note that because all such formulae are *totally* quantified (meaning they have no free variables), they are always either true or false.

Intuitively, there is a clear correspondence between TQBF and an alternating TM. The only difference is that TQBF formulae effectively defer all of the computation until the alternation is done. And, in fact, this whole class of problems can be further reduced to 3-SAT.

In summary, it is clear that TQBF $\in$ ATIME(poly). The fact that it is also a complete problem is not hard to verify.