6.841 Advanced Complexity Theory February 28, 2005

Lecture 8

Lecturer: Madhu Sudan *Scribe: Arnab Bhattacharyya*

1 A New Theme

In the past few lectures, we have concentrated on non-uniform types of computation and focused our energies on trying to prove lower bounds for various kinds of resources. But here, we are stuck; proving tighter lower bounds seems to be an extremely difficult problem. For example, we don't know how to separate randomized polynomial-time algorithms from nondeterministic exponential-time algorithms even though intuitively, they seem to be clearly different. So, now we will shift our attention in order to explore other goals of complexity theory and to discuss other interesting aspects of computation.

2 Motivation

Consider the *MinDNF* problem. Informally speaking, it is the problem of finding if a given boolean expression can be written as a small DNF expression[1]. This problem frequently comes up in VLSI design where we want to minimize the number of gates in a boolean circuit. The size of a DNF expression is defined as the number of literals in the expression. More formally, then we can define the following language to correspond to the *MinDNF* problem:

$$\text{MINDNF} = \{(\phi, k) | \phi \text{ is a CNF expression and } \exists \text{ DNF expression } \psi \text{ s.t. } |\psi| \leq k \text{ and } \psi \text{ is equivalent to } \phi\}$$

MINDNF $\in$ PSPACE, but it is not known if MINDNF is in NP or in coNP. However, it seems easier than some other PSPACE problems. Let's try to understand this more carefully.

If a language L can be decided by a deterministic TM in polynomial time, then L is in P. Adding non-determinism appears to make our computations more powerful. In a usual nondeterministic TM, a branching computation accepts if any of the forked computations accept; in other words, each node in the computation tree OR's the computations of its children. Let us call such a TM an $\exists$-TM. Clearly, if an $\exists$-TM accepts L in polynomial time, then L is in NP. Similary, a $\forall$-TM is a nondeterministic TM where a nondeterministic computation succeeds if all of the branched computations accept. If L is accepted by a $\forall$-TM in polynomial time, then L is in coNP. What happens if both existential and universal nondeterministic steps are present in a TM? Clearly such a machine can decide TQBF in polynomial time because the machine can have an existential step for each $\exists$ quantifier and a universal step for each $\forall$ quantifier in the TQBF input formula. So, a TM with an unbounded amount of alternation between existential and universal steps can decide any PSPACE problem in polynomial time.

Such a machine seems too powerful. Let us try to restrict computational power by restricting the amount of alternation permitted to the TM. Suppose a TM is such that its configuration graph can be cut into two subgraphs G_1 and G_2. All the nodes in G_1 are existential and all the nodes in G_2 are universal, and there are no edges that go from a node in G_2 to a node in G_1. In other words, there can be exactly one alternation from existential nondeterminism to universal nondeterminism. Observer that MINDNF can be decided by such a TM; this is so because MINDNF can be formulated as:

$$\exists \psi \forall x \in \{0,1\}^n (\phi(x) = \psi(x) \wedge |\psi| \leq k)$$

The class of languages that can be decided by a TM with one alternation from $\exists$-steps to $\forall$-steps was introduced by Meyer and Stockmeyer in 1973 as Σ_2^P. As we've just indicated, MINDNF $\in \Sigma_2^P$. Meyer and Stockmeyer presented the following complete problem for Σ_2^P:

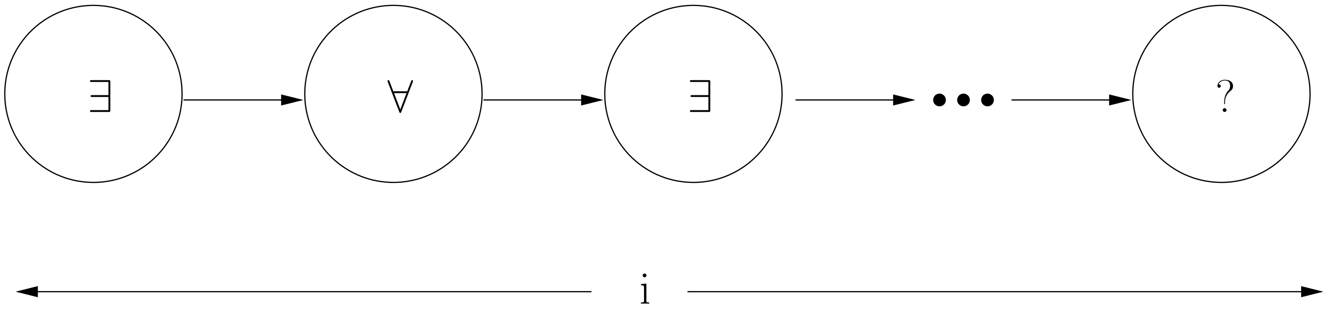
$$\{\phi | \phi \text{ is a CNF expression and } \exists x \forall y, \phi(x, y) = \mathit{true}\}$$

[1] A boolean expression ϕ is in disjunctive normal form if $\phi = \bigvee_{i=1}^{n} D_i$ where $n \geq 1$ and each of the D_i's is the conjunction of boolean literals. An example is $(x_1 \wedge x_2 \wedge \bar{x}_3) \vee (x_3 \wedge x_4)$.

It turns out that MINDNF is also a complete problem for Σ_2^P, although this was shown only in 2000 by Umans. Meyer and Stockmeyer also defined the complement of the Σ_2^P language as Π_2^P and similarly defined Σ_3^P, Π_3^P and so on.

3 The Polynomial Hierarchy

Definition 1 Σ_i^P *is the language decided by machines of the form*



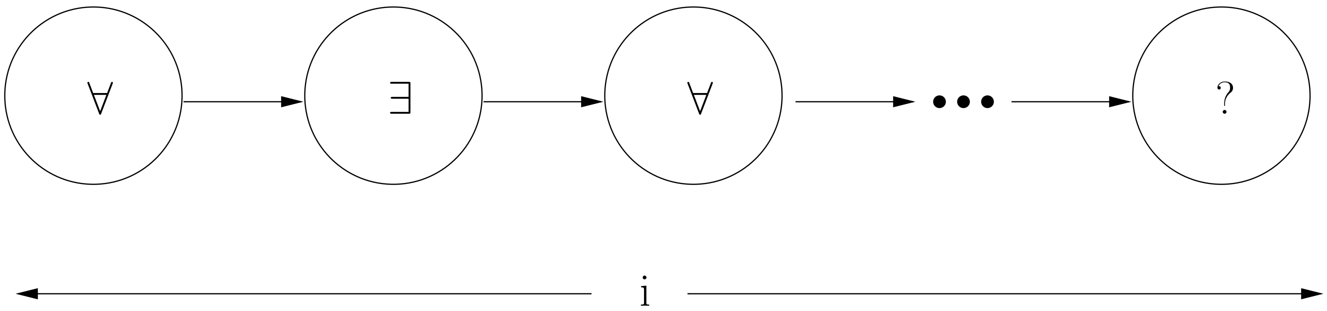
in polynomial time.

Σ_i^p has the following complete problem:

$$\{\phi | \phi \text{ is in CNF and } \exists x_1 \forall x_2 \cdots Q_i x_i \text{ such that } \phi(x_1, \cdots, x_i) = \textit{true}\}$$

There's a similar definition for the complement language of Σ_i^P:

Definition 2 Π_i^P *is the language decided by machines of the form*



in polynomial time.

Π_i^p has the following complete problem:

$$\{\phi | \phi \text{ is in CNF and } \forall x_1 \exists x_2 \cdots Q_i x_i \text{ such that } \phi(x_1, \cdots, x_i) = \textit{true}\}$$

Definition 3 $\mathrm{PH} = \bigcup \Sigma_i^P$

Directly from the definitions, we have:

- $\mathrm{NP} = \Sigma_1^P$ and $\mathrm{coNP} = \Pi_1^P$
- $\Sigma_i^P \subseteq \mathrm{PSPACE}$ and $\Pi_i^P \subseteq \mathrm{PSPACE}$
- $\Sigma_i^P, \Pi_i^P \subseteq \Sigma_{i+1}^P, \Pi_{i+1}^P$

Proof Sketch: Every string in a language in Σ_i^P can be turned into a string from a language in Σ_{i+1}^P by adding a vacuous quantifier for the extra variable at the end of the list of quantifiers in the original string. It can be turned into a string from a language in Π_{i+1}^P by adding a vacuous $\forall$ quantifier at the beginning of the original list of quantifiers. Similarly for Π_i^P. ■

- If there exists an i such that $\Sigma_i^P \subseteq \Pi_i^P$, then for all $j > i$, $\Sigma_j^P = \Sigma_i^P$.

Proof Sketch: Clearly, it suffices to show that $\Sigma_i^P \subseteq \Pi_i^P$ implies $\Sigma_i^P = \Sigma_{i+1}^P$. Suppose $L \in \Sigma_{i+1}^P$. Because of the complete problem in this class, $\phi \in L$ iff

$$\exists x_1 \forall x_2 \exists x_3 \cdots Q_{i+1} x_{i+1}\ \phi(x_1, x_2, x_3, \cdots, x_{i+1})$$

By assumption, $\Sigma_i^P \subseteq \Pi_i^P$. So, with respect to the definition of Σ_i^P, we can rewrite the above as:

$$\exists x_1 \forall (x_2, x_3) \cdots Q'_{i+1} x_{i+1}\ \phi(x_1, (x_2, x_3), \cdots, x_{i+1})$$

Here ϕ is modified to take i input variables. Thus by joining the two universal quantifiers and modifying the boolean function slightly, we see that $L \in \Sigma_i^P$. So, $\Sigma_{i+1}^P = \Sigma_i^P$. ■

The implication in the last claim, that for all $j > i$, $\Sigma_j^P = \Sigma_i^P$, is known as collapse of the polynomial hierarchy. There is a belief among complexity theorists that PH does not collapse. Observe that this is a very strong belief, stronger than the P $\neq$ NP or NP $\neq$ coNP conjectures. However, there is not much independent "evidence" supporting it, and one can very well imagine it being false. The belief that PH does not collapse is important because the negation of other unproven conjectures would make it false. For example, if NP were shown to have polynomial-sized circuits, then PH would collapse. If randomized polynomial time algorithms were shown to coincide with deterministic polynomial time algorithms, then PH would collapse. These types of implications make the PH collapse conjecture a central hacking point for complexity theorists.

4 Alternation as a Resource

To explore how adding alternation affects computational power, we define the following complexity classes:

- ATIME$(t(n)) = \{L | L$ is decided by a machine with unbounded alternation using $O(t(n))$ time$\}$
- ASPACE$(s(n)) = \{L | L$ is decided by a machine with unbounded alternation using $O(s(n))$ space$\}$
- ATISP$(a, t, s) = \{L | L$ is decided by a machine with alternation quantified by a, time quantified by t and space quantified by $s\}$

Even though NL = coNL might suggest that alternation would not be very helpful in reducing space complexity, *unbounded* alternation is very powerful. We have the following results:

Theorem 4 SPACE$(s) \subseteq$ ATIME$(s^2) \subseteq$ SPACE(s^2)

Proof Sketch: The proof of the first containment is very much like that of Savitch's theorem. We will construct an alternating TM to simulate a deterministic TM M that uses s-space. Suppose M has less than 2^{ks} configurations reachable in s space. Then, we have to determine if M can go from the initial c_i configuration to the final c_f configuration in this much time. An alternating procedure that tests if configuration c_2 is reachable from c_1 within time t first existentially branches to guess a configuration c_m between c_1 and c_2 and then universally branches to test recursively that both the paths between c_1 and c_m and between c_m and c_2 can be traversed in $t/2$ time. So, using this alternating procedure, an alternating TM can test if M can go from c_i to c_f in the required amount of time. The time needed for the test is $O(s)$ to write a configuration at each recursive step times $\log 2^{ks} = O(s)$, the depth of the recursion tree. So, the total time needed is $O(s^2)$.

To get the second containment, we can do a straightforward simulation of the alternating machine with a deterministic TM. For example, the deterministic TM can do a depth-first-search of the alternation tree to determine which nodes in the tree are accepting. The depth of the recursion stack in the DFS is the time complexity of the alternating algorithm. At each recursive step, we need only a constant amount of space to store which nondeterministic choice was made and whether the branching was existential or universal. Thus, the space complexity of the deterministic simulation is the same order as the time complexity of the alternating machine. ■

And even more suprisingly,

Theorem 5 $\mathrm{TIME}(2^s) \subseteq \mathrm{ASPACE}(s) \subseteq \mathrm{TIME}(2^{O(s)})$

Proof Sketch: We'll show the second containment first. Suppose we have an alternating TM using space $O(s)$. Then the depth of the alternating tree will be at most $2^{O(s)}$ and the total number of nodes will be at most $2^{2^{O(s)}}$. But the number of configurations is also at most $2^{O(s)}$. So, if we collapse identical configurations in each row of the alternation tree into single nodes, then we have a $2^{O(s)} \times 2^{O(s)}$ DAG, and so $2^{O(s)}$ is enough time to traverse this graph.

The first containment is a bit harder to show. Consider the tableau of a DTM A with time complexity $O(2^s)$. We will construct a machine M that checks if A has contents σ on cell i of the tableau after t steps of computation. To do this, M first existentially guesses contents σ_1, σ_2, σ_3 of cells $i-1$,i,$i+1$ at time $t-1$ to verify if they would yield σ according to A's transition function. Next, M branches universally to recursively check whether A has contents σ_1, σ_2 and σ_3 in the cells $i-1$, i and $i+1$ respectively at time $t-1$. At any one point of this checking, M needs to store a pointer to a constant number of configurations. Hence, the space complexity of M is ASPACE(s). ■

5 Conclusion

Next lecture, we'll show the following:

- $\mathrm{SAT} \in \mathrm{L} \Rightarrow \mathrm{SAT} \notin \mathrm{TIME}(\mathrm{n}^{1+\epsilon})$ (for some $\epsilon > 0$).
- $\mathrm{NP} \subseteq \mathrm{P}/poly \Rightarrow$ PH collapses

We can attempt to apply some of the results we've learned here to nonmathematical issues. For example, consider a debate between two contestants in an election. Say candidate 1 claims $x \in L$ and candidate 2 claims $x \notin L$. Computationally, candidate 1 is like an existential step in an NTM and candidate 2 is like a universal step. The interaction with the audience can be modeled as a polytime process. So, with an unbounded number of alternations between the two candidates, the debate system has the power of a ATIME($poly$) TM. By Theorem 4, therefore, such a debate system decides languages in PSPACE. If the candidates do not have anything to say in the debate (i.e. there is no alternation), then the debate system decides languages in P. And if the contestants alternate a bounded number of times, then the debate decides a language in PH. Thus, if it turns out that PH = PSPACE, then in some sense, a full-fledged debate will only be as interesting as a debate with a fixed number of rounds.

The complexity class PSPACE has been used extensively to analyze the complexity of games. Whenever winning or losing a game is a function only of the current board configuration (like Go and not like chess[2]), the game is in PSPACE. This is so because fundamentally, alternation describes a game between two players, the player with existential moves and the player with universal moves; because ATIME($poly$) = PSPACE, any such game where a polynomial number of configurations are visited from

[2]If a configuration repeats in chess 3 times, then the game ends in a draw.

start to end is in PSPACE. Now contrast such hard games to a game like Solitaire where a computer randomly deals out a deck of cards. Is playing Go against a human expert as intellectually challenging as playing Solitaire against a random distribution of cards? Surprisingly, it turns out that there are distributions of cards for which Solitaire is as challenging as Go! Configurations in Go can be reduced to Solitaire card configurations (although it is not clear if such configuration would actually arise in a real Solitaire game). A random instance of Solitaire can be shown to be hard.