CS221: Computational Complexity Prof. Salil Vadhan

Lecture 10: Alternation

10/11 *Scribe: Ethan Abraham*

Contents

1 Recap From Last Lecture

Recall the following results from last lecture:

- RCH $\in \mathbf{P}$, so $\mathbf{NL} \subseteq \mathbf{P}$, since RCH is **NL**-complete.
- RCH $\in \mathbf{L}^2$ by Savitch's Theorem, thus $\mathbf{NL} \subseteq \mathbf{L}^2$ and $\mathbf{PSPACE} = \mathbf{NPSPACE}$
- RCH $\in$ **co-NL** by Immerman–Szelepcényi, so $\mathbf{NSPACE}(f(n)) = \mathbf{co\text{-}NSPACE}(f(n)) \quad \forall \text{proper } f(n) \geq \log(n)$

A few more things about nondeterministic space are worth pointing out. First, 2SAT is also **NL**-complete. The proof is in the book. While in previous courses, you have mostly seen completeness as a way to show that problems are possibly intractable, here we have used completeness to prove a positive results, e.g. $\mathbf{NL} \subseteq \mathbf{P}$. Although this proof could be accomplished without using completeness, this is not always the case. Lastly, the fact that $\mathbf{NL} = \mathbf{co\text{-}NL}$ was much more surprising than $\mathbf{PSPACE} = \mathbf{NPSPACE}$. Some form of the former was first posed around 1950, and, until it was proved, most people believed it was false.

2 Current Classes

The current situation is:

$$\mathbf{L} \subseteq \mathbf{NL} = \textbf{co-NL} \subseteq \mathbf{P} \subseteq \mathbf{NP}?\textbf{co-NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE}$$

The following are still open:

- Does $\mathbf{NP} = \textbf{co-NP}$?
- Are any of the above inclusions strict inclusions or equalities? The only separation we know for the above classes is that $\mathbf{NL} \subsetneq \mathbf{PSPACE}$, which follows from the Space Hierarchy Theorem and Savitch's Theorem.

It is also important to note that even if $\mathbf{P} \neq \mathbf{NP}$, this does not mean that all problems in $\mathbf{NP} - \mathbf{P}$ are **NP**-complete. In fact, the following theorem states exactly the opposite:

Theorem 1 (Ladner) *If* $\mathbf{P} \neq \mathbf{NP}$*, there exist languages in* **NP** *that are neither in* **P** *nor* **NP***-complete.*

The proof is by diagonalization, and analogues also probably hold for most pairs of complexity classes (like **L** and **NL**). Natural candidates for such languages are Graph Isomorphism, Factoring, Discrete Log, as none of these are known to be **NP**-complete or in **P**. (The last two problems are not languages, but still one can ask whether they are **NP**-hard via Cook reductions or can be solved in polynomial time.)

3 Alternation - Sections 16.2 & 19.1

The motivation for this section is to find a **PSPACE**-complete problem (which will be provable not in **L**, or even **NL**, by the Space Hierarchy Theorem).

Definition 2 (Alternation) *An* alternating TM *is a nondeterministic Turing machine in which each state is labelled either existential(∃) or universal(∀).*

A configuration of an alternating TM is defined to be *accepting* if:

- The TM has halted and is in an accepting state.

- If the TM is in an existential state, at least one child configuration is accepting.
- If the TM is in an a universal state, all children are accepting.

We say that an ATM M *accepts* x is the initial configuration on x is accepting. Note that an ATM which has only $\exists$-states is just like a normal NTM. Thus adding the $\forall$-states allows us to basically add constraints on which computations can actually fail. We can think of the $\exists$-states as used for "guessing" as with NTMs, and the $\forall$-states as ways to verify multiple conditions at the complexity cost of verifying only one (since we take the maximum over all branches). We also naturally define time and space of an ATM as the maximum over all computation paths, and thereby get the complexity classes **ATIME**$(f(n))$, **ASPACE**$(f(n))$, **AP**, **AL**, etc.

Theorem 3 (Alternating Time = Deterministic Space) AP = PSPACE

First, we prove a lemma.

Lemma 4 *For proper $f(n) \geq n$,*
*$L \in$ **ATIME**$(O(f(n)))$ iff $\exists$ a linear-time relation R such that $x \in L \Leftrightarrow$ $\exists y_1 \in \{0,1\} : \forall y_2 \in \{0,1\} \exists y_3 \in \{0,1\}, \ldots, (\exists or \forall)\, y_{O(f(n))}\, R(x, y_1, y_2, \ldots, y_{O(f(n))}) = 1$.*

Proof of Lemma: $\Leftarrow$ is immediate. An alternating algorithm in **ATIME**$(O(f(n)))$ can evaluate the RHS in time $O(f(n))$. For $\Rightarrow$, given some algorithm $M \in$ **ATIME**$(f(n))$, we must only change it to the appropriate form. This is not difficult however, and can be accomplished in the following steps.

- Wherever a node has more than 2 children, add dummy nodes in between to ensure that each node has exactly 2 children.
- To ensure alternation, whenever two $\forall$ follow each other, add an $\exists$-node in between, with both inputs coming from the first $\forall$-node, and both outputs going to the second. Do the opposite for two $\exists$-nodes.
- Move the computation to the end, after all nondeterministic choices have been made.

It is easy to see that each of these steps can be done with only a linear cost in time, so the lemma holds. Alternatively, you can simply define R to

be an algorithm which simulates the NTM M using the y_i's to determine it's nondeterministic choices (using the next y_i which has the same kind of quantifier as the current state of M). Note that the nondeterministic choices of **ATIME**$(f(n))$ algorithm can indeed be described by at most $O(f(n))$ bits. ■

This give us a canonical form for ATMs. We will now prove general inclusion of **ATIME** and **SPACE** classes going either way, which will then allow us to conclude the theorem.

Lemma 5 **ATIME**$(f(n)) \subseteq$ **SPACE**$(O(f(n)))$

Proof: Given any ATM in the canonical form and some input, we can evaluate the alternating tree recursively. As we move down the tree, we only need to remember whether each previous node in the tree is a 0 or 1. We know exactly what kind of node each previous one is, as we are assuming the canonical form. Since the depth of the tree is $O(f(n))$, and calculation of the relation takes time linear in $f(n)$ (and thus use $\leq O(f(n))$ space), we use space at most $O(f(n))$. ■

Lemma 6 **SPACE**$(f(n)) \subseteq$ **ATIME**$(f(n)^2)$

Proof: The proof is similar to that of Savitch's Theorem. Say we are given a TM M : $L_M \in$ **SPACE**$(f(n))$. On input x with length $n = |x|$, the configuration graph has $O(2^{O(f(n))})$ vertices, and each configuration can be stored in space $O(f(n))$. Given this graph, we would like to find out if we can get from a start state to an accepting state. Thus we say that $Path(u, v, i) = 1$ if $\exists$ a path with length$\leq 2^i$ from configuration u to configuration v. Now, we can evaluate $Path(start, accept, O(f(n)))$ in **ATIME**$(f(n)^2)$ by using alternating-recursion. For the base, if $i = 0$, $Path(u, v, i) = 1$ if $u = v$ or if u's child is v. Now, like in Savitch's Theorem, we recursively guess the midpoints, but here we use alternation.

- First, guess a midpoint w (using $\exists$ states).
- Next, we can check if both $path(u, w, i-1)$ and $path(w, v, i-1)$ are 1 (using a $\forall$ state).

Because of how we count time in an alternating TM, and second step verifies two conditions in the time of one. Thus, since each vertex w has description length $O(f(n))$, and we are guessing $O(f(n))$ vertices, we use at most **ATIME**$(f(n)^2)$. ■

Finally, we can now prove the theorem.

Proof Theorem 3: Since the class of polynomials is closed under the squaring-operation, the two lemmas become inclusion each way **AP** and **PSPACE**, thus **AP** = **PSPACE**. ■

How can we think about alternating problems? One simple interpretation is as a game, where the two types of states represent the two players, and R represents a polynomial-time referee. The $\exists$ player tries to make $R = 1$, while the $\forall$ player tries to make $R = 0$. Then, in general, we can say that $x \in L \Leftrightarrow \exists$ a winning strategy for the $\exists$-player. Since this gives us a good way of thinking about problems in **AP**, it also gives a good way to think about problems in **PSPACE**, since we have just shown that these are equal.