**CS221: Computational Complexity** Prof. Salil Vadhan

# Lecture 10: NSPACE, Current Map of Classes, and Alternation & PSPACE

*10/11* *Scribe: Ethan Abraham*

## Contents



## 1 Recap From Last Lecture

Recall the following results from last lecture:

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

A few more things about NSPACE are worth pointing out. First, 2SAT is also **NL**-complete. The proof is in the book. Unlike with time classes, where completeness is usually used to show that problems are possibly intractable, here we have used completeness to prove a positive results, namely that **NL** $\in$ **P**. Although this proof could be accomplished without using completeness, this is not always the case. Lastly, the fact that **NL** = **co-NL** was much more surprising than **PSPACE** = **NPSPACE**. 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:

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

The following are still open:

- Does $\mathbf{NP} = \textbf{co-NP}$?
- Are any of the above inclusions strict inclusions or equalities? The only case we know is the $\mathbf{L} \notin \mathbf{PSPACE}$, which is proven later on in these notes.

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 also holds for **L** and **NL**. Natural candidates for such languages are GRAPH-ISOMORPHISM, FACTORING, and DISCRETE-LOG, as none of these are known to be **NP**-complete. Of course, these are math problems, not languages, so proving anything about them as languages would first involved having a satisfactory definition of their associated languages.

## 3 Alternation - Sections 16.2 & 19.1

The motivation for this section is to find a **PSPACE**-complete problem that is provably not in **L**.

**Definition 2 (Alternation)** *An alternating TM is a non-deterministic turing machine in which each state is labelled either existential($\exists$) or universal($\forall$).*

A configuration of an alternating TM is accepting if:

- The TM has halted and is in an accepting state.
- If the TM is in an existential state, at least one child 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 is an ATM has only $\exists$-states, then is is just a normal NTM. Thus adding the $\forall$-states allows us to basically add constraints on which computations can actually fail. We also naturally define $ATIME(f(n))$ and $ASPACE(f(n))$ as the maximum over the branches, as well as the classes **AP**, **AL**, **APSPACE**, etc.

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

First, we prove a lemma.

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

**Proof:** [Proof of Lemma] $\Leftarrow$ is immediate. An alternating algorithm in $ATIME(O(f(n)))$ can evaluate the RHS in time $O(f(n))$. We can think of the $\exists$-states as guessing an answer, and the $\forall$-states as verification. However, the $\forall$-states allow us to verify multiple conditions at the time-cost of only one, since we take the maximum time. For $\Rightarrow$, given some algorithm $M \in ATIME(f(n))$, we must only change is 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 of R to the end of the turing machine.

It is easy to see that each of these steps can be done in linear time, so the lemma holds. ■

This give us a canonical form for ATMs. We will not 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 Savage'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 Savage's Theorem, we recursively guess the midpoints, but here we use alternation.

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

Because of how we count time in an alternating TM, and second step verifies two paths in the time of one path. Thus, since $w$ 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:** [Proof of Theorem] 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.