CS 710: Complexity Theory 9/29/2011

Lecture 7: The Polynomial-Time Hierarchy

Instructor: Dieter van Melkebeek Scribe: Xi Wu

In this lecture we first finish the discussion of space-bounded nondeterminism and show that NSPACE($s(n)$) is closed under complement. After showing this theorem, we continue by introducing polynomial-time hierarchy, prove some of its basic properties, and give complete problems for each level of the hierarchy. Besides the proof presented in the lecture, we give an alternative construction for the complete problem of Σ_k^p, by going through circuits, and then transform circuits into formulas using Tseitin transformation.

1 Nondeterministic Space is Closed under Complement

In this section we prove that $\overline{st\text{-CONN}} \in$ NL. Recall that for $\overline{st\text{-CONN}}$, the input instance is a directed graph G with two distinct vertices s and t and the instance is a YES instance iff there is *no* directed path from s to t. The core argument for this is to cast the question of "no path exists from s to t" to the problem of checking every vertex reachable from s is not t, where we show the latter can be done using nondeterministic log-space. From the last lecture we know that st-CONN is complete for NL under $\leq_m^{\log}$ reduction, and hence its complement is complete for coNL under the same reduction. Therefore it suffices to argue that $\overline{st\text{-CONN}} \in$ NL, thus coNL $\subseteq$ NL, and hence coNL = NL. Note here we use the following property regarding any complexity class.

Proposition 1. *For any complexity class C, if $co\mathcal{C} \subseteq \mathcal{C}$ then $co\mathcal{C} = \mathcal{C}$*

Proof. It suffices to show that $\mathcal{C} \subseteq co\mathcal{C}$ as well. $L \in \mathcal{C} \Rightarrow coL \in co\mathcal{C} \Rightarrow coL \in \mathcal{C} \Rightarrow L \in co\mathcal{C}$. □

To argue that $\overline{st\text{-CONN}}$ is in NL we must construct a nondeterministic Turing machine so that on input, if *at least one* branch accepts then we conclude there is no directed path from s to t. We know that checking whether there is one path from s to t can be easily done in this way, but it seems difficult that we can do in this way to check no st-path exists, naively we would think this requires to check all paths and reject every of them to be an st-path.

The first argument towards this construction is that we can cast the question of "no path exists from s to t" to the problem that "every vertex reachable from s is not t". Imagine that we already know the number of vertices reachable from s, say it is c, then we can easily check that t is not reachable from s using a nondeterministic log-space machine as follows. We maintain a counter to record the number of reachable vertices from s we find. For each vertex v other than t in the graph, we guess a path from s to v and check whether this is indeed a path. If so, we increase the counter. Finally if the counter equals c, then we accept, otherwise we reject. The pseudo-code for this step is shown in Algorithm 1:

Algorithm 1: st-NOT-CONNECTED
Input: $s, t \in V$ and $c \in \mathbb{N}$ which is the number of vertices reachable from s
Output: Accept if t is *not* reachable from s
(1) $cnt \longleftarrow 0$
(2) **foreach** vertex v other than t
(3) Guess a path from s to v, if valid, $cnt \longleftarrow cnt + 1$.
(4) **if** $cnt = c$ accept, otherwise reject.

Algorithm 1 works provided c is correct. The observation is that for any v *not* reachable from s, the algorithm never makes a mistake by increasing the counter. Now, if t is not reachable from s, then there *exists* a branch where we correctly guess the path for each v reachable from s, and the counter has to be c and we correctly accept. Otherwise, if t is indeed reachable from s, then without checking t we can never have the counter set to c, hence all branches reject.

So it suffices to argue that we can correctly compute c using a nondeterministic log-space machine. This is where we use the technique of inductive counting. The idea is to compute c_i, which is the number of vertices reachable from s within at most i steps, we keep computing $c_0 = 1, c_1, \ldots, c_{n-1} = c$. In each step to compute c_{i+1} the algorithm only needs to know c_i, hence at least for input we only need log-space.

Now restrict the eyes to compute c_{i+1} based on c_i. If we can correctly do this then we are done because the base of the computing is trivial as $c_0 = 1$. The idea is that, from c_i, we can recover the set of vertices reachable from s in at most i steps. The way to do it is still guess and check. Inside this procedure we maintain a counter, say c', to record the number of vertices reachable from s within i steps. For each vertex v, we guess a path of length at most i and check whether that path is valid. If so then we increase c'. Finally if $c' = c_i$ then we know we must be correct. Again, provided that c_i is correct, we are sure that one branch will have $c' = c_i$. Then with the set of vertices reachable from s in i steps, we can easily compute c_{i+1} by going one step further. Looks good but we have an issue: the set of vertices maintained may break the space requirement. Nevertheless, we can still write down the wrapper procedure of the inductive counting, as follows.

Algorithm 2: Inductive-Count
Input: s, $0 \leq i \leq |V| - 1$ and c, the number of vertices reachable in i steps
Output: c': the number of vertices reachable in $i + 1$ steps
(1) $c' \longleftarrow 0$
(2) **foreach** vertex $v \in V$
(3) **if** Reachable(s,v,i,c)
(4) $c' \leftarrow c' + 1$;
(5) return c'.

One more twist is needed to solve this issue. We trade time for space. The idea is, rather than reconstructing the set and store it for later use, for every v we *re-experience* the set reachable within i steps, one by one and test whether from this set we can reach v in one step. Specifically for one v, we initialize $c' = 0$ and scan every vertex w in the graph. For each w we guess a path of length at most i: first, if the path is valid then we increase c', second, if v can be reached from w within one step, then continue to the final check. In the final check a branch is accepted if $c' = c_i$, which is the case that all vertices reachable within i steps have been tried.

Algorithm 3: Reachable
Input: $s, v \in V$, i and c, the number of vertices reachable in i steps
Output: Accept if v is reachable in $i+1$ steps
(1) $cnt \longleftarrow 0$
(2) **foreach** vertex $w \in V$
(3) Guess a path of length at most i from s to w,
(4) **if** path is valid
(5) (1. $cnt \leftarrow cnt + 1$.
(6) (2. check whether v is reachable from w in one step, if so *accept*.
(7) **if** $cnt < c$ **then** output ? (abort on this branch), **else** *reject*.

The wrapper procedure is implemented simply as follows:

Algorithm 4: Wrapper
Input: Directed acyclic graph $G = (V, E)$ and $s, t \in V$
Output: Accept if t is *not* reachable from s, using nondeterministic log-space.
(1) $c = 1$; // s is reachable in 0 steps...
(2) **foreach** $i = 0, 1, \ldots, |V| - 2$
(3) $c \leftarrow$ Inductive-Count(s, i, c)
(4) st-NOT-CONNECTED(s, t, c);

We conclude this section with the theorem and two of its immediate corollary.

Theorem 1. $\overline{st\text{-CONN}}$ *is in* NL.

Corollary 1. NL = coNL.

Corollary 2. NSPACE$(s(n))$ = coNSPACE$(s(n))$.

Proof. Essentially these space bounded computation can be viewed as st-connectivity problem where each vertex is represented using $s(n)$ bits. The same argument carries through. □

2 Polynomial Time Hierarchy

Polynomial time hierarchy is an important concept in Computational Complexity theory which generalizes the concept of NP and coNP by defining a hierarchy of complexity classes. One motivation of PH is to consider the MIN-FORMULA problem, the input of this problem is a Boolean formula ϕ. We say that two formulas ϕ and ψ are *equivalent* if for any assignment x, $\phi(x) = \psi(x)$. Now the membership of MIN-FORMULA is that ϕ is a YES instance if and only if there is no equivalent formula ψ of ϕ that $|\psi| < |\phi|$. It is clear that we can characterize this problem with two quantifiers as follows:

$$\phi \in \text{MIN-FORMULA} \iff \forall\psi\exists x\big[|\psi| < |\phi| \to \psi(x) \neq \phi(x)\big]$$

It should be clear that $|\psi|$ and $|x|$ are polynomially bounded by $|\phi|$, as $|\psi| \leq |\phi|, |x| \leq |\phi|$. Note that $L \in$ NP can be characterized as $x \in L \iff (\exists y \in \Sigma^{|x|^c})\langle x, y\rangle \in V$ where V is some deterministic polynomial time predicate. For ease of notation, we will denote by $\exists^p y$ and $\forall^p y$ where $|y|$ is polynomially bounded by $|x|$. That is, $y \in \Sigma^{|x|^{O(1)}}$ where Σ is the alphabet.

Definition 1 (Σ_k^p)**.**

$$\Sigma_k^p = \Big\{ L \;\Big|\; \textit{membership of } L \textit{ can be expressed as: } x \in L \iff \exists^p y_1 \forall^p y_2 \cdots Q_k^p y_k \big[\langle x, y_1, \ldots, y_k\rangle \in V\big] \textit{ where } V \textit{ is some polynomial time predicate.} \Big\}$$

Definition 2 (Π_k^p)**.**

$$\Pi_k^p = \Big\{ L \;\Big|\; \textit{membership of } L \textit{ can be expressed as: } x \in L \iff \forall^p y_1 \exists^p y_2 \cdots Q_k^p y_k \big[\langle x, y_1, \ldots, y_k\rangle \in V\big] \textit{ where } V \textit{ is some polynomial time predicate.} \Big\}$$

By definition $\Sigma_0^p = \mathrm{P}$, $\Sigma_1^p = \mathrm{NP}$ and $\Pi_1^p = \mathrm{coNP}$. It should be clear that $\Sigma_k^p = co\Pi_k^p$.

Proposition 2. $\Sigma_k^p \cup \Pi_k^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$

Proof. We argue that $\Sigma_k^p \cup \Pi_k^p \subseteq \Sigma_{k+1}^p$, and by similar argument we can put the union in Π_{k+1}^p, hence the proposition follows. Let $L \in \Sigma_k^p$, then L is automatically in Σ_{k+1}^p by adding a dummy variable and an appropriate quantifier at the end of all quantifiers. If $L \in \Pi_k^p$ then we can add a dummy variable and an $\exists$ quantifier at the start of all quantifiers, which puts L in Σ_{k+1}^p. □

Now we are in position to define the polynomial time hierarchy, where the last equality holds because the property we just proved: $\Sigma_k^p \cup \Pi_k^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$.

Definition 3 (Polynomial Time Hierarchy: PH)**.**

$$\mathrm{PH} = \bigcup_{k \geq 0} \Sigma_k^p = \bigcup_{k \geq 0} \Pi_k^p$$

By definition, *syntactically* PH is an infinite hierarchy of complexity classes. However whether *semantically* it is infinite remains a big open problem. Figure 1 gives a pictorial description of polynomial time hierarchy. Let's begin the study of PH by proving some of its basic properties.

Proposition 3. *If* $\Sigma_k^p = \Pi_k^p$ *then* $\Sigma_k^p = \Sigma_{k+1}^p$.

Proof. It suffices to argue that if the condition holds then $\Sigma_{k+1}^p \subseteq \Sigma_k^p$. For this consider $L \in \Sigma_{k+1}^p$, by definition there is a polynomial time predicate V so that:

$$x \in L \iff (\exists y_1 \in \Sigma^{|x|^c})(\forall y_2 \in \Sigma^{|x|^c}) \cdots (Q_k y_k \in \Sigma^{|x|^c}) \big[\langle x, y_1, \ldots, y_k\rangle \in V\big]$$

Define L' as follows:

$$L' = \Big\{ \langle x, y_1\rangle \;\Big|\; \langle x, y_1\rangle \in L' \iff (\forall y_2 \in \Sigma^{|x|^c}) \cdots (Q_k y_k \in \Sigma^{|x|^c}) [\langle x, y_1, \ldots, y_k\rangle \in V] \Big\}$$

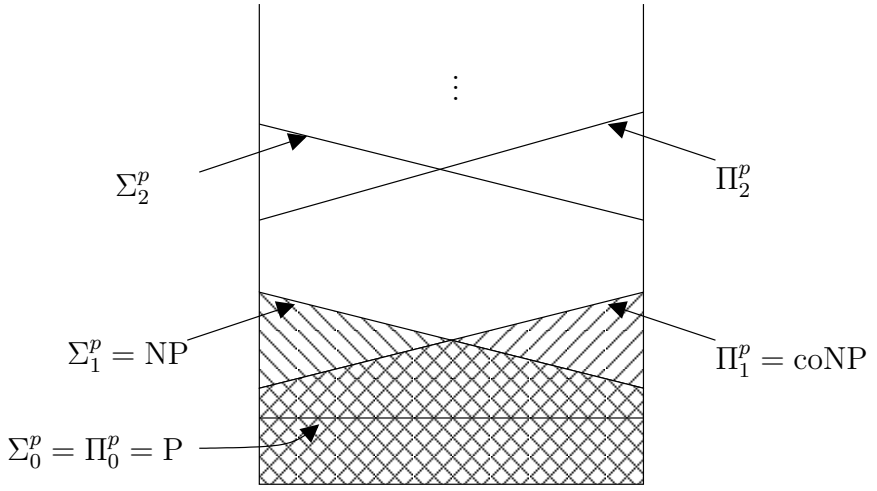

Figure 1: Pictorial illustration of the polynomial hierarchy

It is clear that $L' \in \Pi_k^p$. By assumption $\Pi_k^p = \Sigma_k^p$, therefore we have a predicate V' such that

$$L' = \Big\{ \langle x, y_1 \rangle \;\Big|\; \langle x, y_1 \rangle \in L' \iff (\exists y_1' \in \Sigma^{|\langle x, y_1 \rangle|^{c'}}) \cdots (Q_k' y_{k-1}' \in \Sigma^{|\langle x, y_1 \rangle|^{c'}}) \big[\langle x, y_1, y_1', \ldots, y_{k-1}' \rangle \in V' \big] \Big\}$$

Observe that $|y_1| = |x|^{O(1)}$, hence $y_1', y_2', \ldots, y_{k-1}'$ all have lengths polynomially bounded by $|x|$. Now we can plug in this back and combine the first two existential quantifiers, which shows that $L \in \Sigma_k^p$. The proof is now complete. □

Theorem 2. *If* $\Sigma_k^p = \Pi_k^p$ *then* $\mathrm{PH} = \Sigma_k^p$.

Proof. By proposition 3 we have that $\Sigma_k^p = \Sigma_{k+1}^p$. Then by proposition 2, $\Sigma_{k+1}^p = \Sigma_k^p \subseteq \Pi_{k+1}^p$, hence by proposition 1 $\Sigma_k^p = \Sigma_{k+1}^p = \Pi_{k+1}^p$. All the way up by applying proposition 3, then for every $j \geq k$, $\Sigma_k^p = \Sigma_j^p = \Pi_j^p$, this means the entire polynomial time hierarchy collapses to Σ_k^p. □

Finally we introduce a natural complete problem under polynomial-time mapping reduction for each level of the hierarchy. Define

$$T\text{-}\Sigma_k = \Big\{ \text{TRUE fully quantified } \Sigma_k\text{-formulas}: (\exists y_{1,1}, \ldots, y_{1,\lambda_1})(\forall y_{2,1}, \ldots, y_{2,\lambda_2}) \cdots (Q_k y_{k,1}, \ldots, y_{k,\lambda_k}) \Phi(y_{1,1}, \ldots, y_{1,\lambda_1}, y_{2,1}, \ldots, y_{2,\lambda_2}, \ldots, y_{k,1}, \ldots, y_{k,\lambda_k}) \Phi \text{ is a CNF formula when } k \text{ is odd and a DNF formula when } k \text{ is even.} \Big\}$$

To ease notation, we will write $y_1 := (y_{1,1}, \ldots, y_{1,\lambda_1})$ to denote the vector of λ_1 Boolean variables.

Proposition 4. *T-Σ_k is complete for Σ_k^p under $\leq_m^p$.*

Proof. Consider the case that k is odd. Define the language L' as follows:

$$L' = \Big\{ \langle x, y_1, \ldots, y_{k-1} \rangle \Big| \langle x, y_1, \ldots, y_{k-1} \rangle \in L' \iff \exists y_k V(\langle x, y_1, \ldots, y_{k-1} \rangle, y_k) = 1 \Big\}$$

We view the tuple $\langle x, y_1, \ldots, y_{k-1} \rangle$ as u, and clearly $L' \in$ NP. Now recall what NP-completeness proof of SAT really tells us is that there is a CNF formula Φ, which *only depends on* $|u|$ such that

$$u \in L' \iff \exists z \Phi_{|u|}(u, z) = 1$$

The formula only depends on $|u|$ because we represent inputs as Boolean variables (to be filled by assignment), and the only place we use these variables is where we check the values assigned to variables representing the input value read at input head, equal the assignment to these variables. All other parts of the formula use some other variables to check whether every move of the computation is valid, which only depends on input length. For more details the reader can refer to notes of Lecture 4 (NP-completeness).

Now this argument gives the quantifiers over $y_1, \ldots, y_{k-1}$ for free. That is,

$$x \in L \iff \exists^p y_1 \forall^p y_2 \ldots, \forall^p y_{k-1} \exists z \Phi_{|u|}(u, z) = 1$$

This completes the case when k is odd.

The case for even k is similar. The difference is that now inner-most quantifier is $\forall$. Define L' in a similar way we have that $u \in L' \iff \forall y_k [V(u, y_k) = 1]$. And hence $u \in \overline{L'} \iff \exists y_k [\overline{V}(u, y_k) = 1]$, now again by NP-completeness proof of SAT, we have a formula Φ in CNF form so that $u \in \overline{L'} \iff \exists z [\Phi(u, z) = 1]$. Hence $u \in L' \iff \forall z [\overline{\Phi}(u, z) = 1]$. Now $\Psi = \overline{\Phi}$ is in DNF-form, which proves the case when k is even. □

2.1 An Alternative Construction

In this section we give an alternative construction that T-Σ_k is complete for Σ_k^p under $\leq_m^p$ by going through circuits and then transforming the circuit into Boolean formula.

Proof. Consider a language in $L \in \Sigma_k^p$. By definition there is a polynomial time verifier V so that:

$$x \in L \iff (\exists y_1 \in \Sigma^{|x|^c})(\forall y_2 \in \Sigma^{|x|^c}) \cdots (Q_k y_k \in \Sigma^{|x|^c}) \big[\langle x, y_1, \ldots, y_k \rangle \in V \big]$$

There are two steps in the proof. In the first step, we argue that the computation of the polynomial time predicate V can be transformed into a polynomial size circuit C, where the circuit *only depends on the input length* $|x|$ and that:

$$x \in L \iff (\exists z_1 \in \{0,1\}^{|x|^{O(1)}}) \cdots (Q_k z_k \in \{0,1\}^{|x|^{O(1)}}) \big[C_x(z_1, \ldots, z_k) = 1 \big] \tag{1}$$

We emphasize again that C_x only depends on $|x|$ and it works on input $z_1, \ldots, z_k$, which consists of Boolean variables. The realization of this circuit is by the method of computation tableau, which is similar to NP-completeness proof of SAT. The concrete construction is presented later.

The second step transforms C_x into a formula Φ_x while keeping the membership, and the transformation depends on whether k is even or odd. We will use Tseitin transformation and its two forms.

Lemma 1 (Tseitin-I)**.** *Let C be a circuit on n variables $\{x_1, \ldots, x_n\}$, and let the set of its internal/output gates be $\{1, \ldots, m\}$, and assume the output gate is m. Then there exists a* **CNF** *formula Φ on $n+m$ variables $\{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ so that for every assignment $a = (a_1, \ldots, a_n)$:*

$$C(a) = 1 \iff \exists v \in \{0,1\}^m \Phi(x = a, y = v) = 1$$

Lemma 2 (Tseitin-II)**.** *Let C be a circuit on n variables $\{x_1, \ldots, x_n\}$, and let the set of its internal/output gates be $\{1, \ldots, m\}$, and assume the output gate is m. Then there exists a* **DNF** *formula Ψ on $n+m$ variables $\{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ so that for every assignment $a = (a_1, \ldots, a_n)$:*

$$C(a) = 1 \iff \forall v \in \{0,1\}^m \Psi(x = a, y = v) = 1$$

We leave the proofs of these two lemmas later. Now we show how to use them to transform the circuit. Suppose C_x has m gates. There are two cases. If k is odd, then $Q_k = \exists$. Applying Tseitin-I to C_x gives a CNF Φ_x such that:

$$\begin{aligned} x \in L \iff & (\exists z_1 \in \{0,1\}^{|x|^{O(1)}})(\forall z_2 \in \{0,1\}^{|x|^{O(1)}}) \cdots (\exists z_k \in \{0,1\}^{|x|^{O(1)}}) \\ & (\exists v \in \{0,1\}^m) \big[\Phi_x(z_1, \ldots, z_k, v_1, \ldots, v_m) = 1\big] \end{aligned}$$

Combining the last two $\exists$ quantifiers proves this case. Otherwise k is even, hence $Q_k = \forall$. Applying Tseitin-II to C_x gives a DNF Ψ_x such that:

$$\begin{aligned} x \in L \iff & (\exists z_1 \in \{0,1\}^{|x|^{O(1)}})(\forall z_2 \in \{0,1\}^{|x|^{O(1)}}) \cdots (\forall z_k \in \{0,1\}^{|x|^{O(1)}}) \\ & (\forall v \in \{0,1\}^m) \big[\Psi_x(z_1, \ldots, z_k, v_1, \ldots, v_m) = 1\big] \end{aligned}$$

Again, combining the last two $\forall$ quantifiers proves this case. The theorem now follows. □

Now we prove Tseitin transformations.

Proof. (Tseitin-I) Let the set of internal/output gates be $\{1, \ldots, m\}$, and input gates be $x_1, \ldots, x_n$, let the output gate be m. For each gate i introduce a variable y_i. To construct a CNF formula, the idea is to introduce, for each gate, a constraint C_i (a clause), to enforce the computation at this gate is correct.

For this, consider one case where i is a AND-gate, and the two inputs are internal gates j and k, therefore we build a constraint $C_i : y_i = (y_j \wedge y_k)$, where the equality can be written in CNF:

$$\begin{aligned} & \big(y_i \vee (\overline{y_j \wedge y_k})\big) \wedge \big(\overline{y_i} \vee (y_j \wedge y_k)\big) \\ \iff & (y_i \vee \overline{y_j} \vee \overline{y_k}) \wedge \big(\overline{y_i} \vee y_j\big) \wedge (\overline{y_i} \vee y_k) \end{aligned}$$

Now it is natural to define Φ as:

$$\Phi = \bigwedge_i C_i \wedge (y_m = 1)$$

Φ is interpreted as "computation at every gate must be consistent, and the output is 1". It is clear that for any assignment a, $C(a) = 1$ if and only if there exists an assignment to $y \in \{0,1\}^m$ such that $\Phi(a, y) = 1$. □

Proof. (Tseitin-II) Apply Tseitin-I to $\overline{C}$, hence $\overline{C}(a) = 1$ if and only if $\exists v \in \{0,1\}^m \Phi(x = a, y = v) = 1$. Therefore $\overline{\Phi}$ is the desired formula because.

$$\begin{aligned} C(a) = 1 &\iff \overline{C}(a) = 0 \\ &\iff \neg[\exists v \in \{0,1\}^m \Phi(x = a, y = v) = 1] \\ &\iff \forall v \in \{0,1\}^m \overline{\Phi}(x = a, y = v) = 1 \end{aligned}$$

□

It remains to argue that we can transform any polynomial time predicate with alternating inputs into a circuit with alternating inputs. We will not get into technical details because it's similar to the NP-completeness proof of SAT. Here is the idea: we construct the computation tableau, which is bounded by $t(n) \times t(n)$, where $t(n)$ is the running time of V on input of length n, and view each row as configuration of V. Then circuit C_x consists of the following for $0 \leq i \leq t(n)$:

- The first row ($i = 0$) implements a circuit to check the input is x, the alternating inputs are also put on the first row.
- For any $1 \leq i \leq t(n) - 1$, we implement a circuit that computes, based on the last configuration, along with the alternating inputs, the next configuration.
- The last row ($i = t(n)$) implements a circuit to check we are in accept configuration.

Because of the high locality of Turing machine, each circuit at each row consists of polynomially many constant size circuits, hence overall the resulting circuit is of size polynomial in $|x|$, further its computation on alternating inputs is exactly the same as the computation of V, this full fills the requirement of (1).

3 Next Time

We will introduce three alternative, but equivalent characterizations of PH:

1. $\Sigma_{k+1}^p = \mathrm{NP}^{\Sigma_k^p}$ and $\Pi_{k+1}^p = \mathrm{NP}^{\Pi_k^p}$.
2. By uniform exponential size constant-depth circuits.
3. By alternating Turing machines.

Acknowledgements

In writing the notes for this lecture, I perused the notes for lecture 5 by Matthew Anderson and lecture 6 by Piramanayagam Arumuga Nainar, from the Spring 2007 offering of CS 810. I use the picture for polynomial time hierarchy by Piramanayagam Arumuga Nainar for lecture 6 from the Spring 2007 offering of CS 810.