Computational Complexity Theory, Fall 2010 September 15

Lecture 7: Polynomial time hierarchy

Lecturer: Kristoffer Arnsfelt Hansen Scribe: Mads Chr. Olesen

Recall that adding the power of alternation gives the following:

$$\begin{array}{ccccccc} P & \subseteq PH \subseteq & PSPACE & \subseteq & EXP & \subseteq & EXPSPACE \\ \parallel & & \parallel & & \parallel & & \parallel \\ AL & \subseteq & AP & \subseteq & APSPACE & \subseteq & AEXP \end{array}$$

Note the class PH, the polynomial time hierarchy and the topic of this lecture.

The polynomial time hierarchy

Definition 1. An alternating TM M is an Σ_k-machine if for all inputs x and all possible computations by M on x, configurations come in *intervals* satisfying:

- All configurations within an interval is of the same type.
- First interval is of type $\vee$.

Analogously define a Π_k-machine: The same, but the first interval is of type $\wedge$.

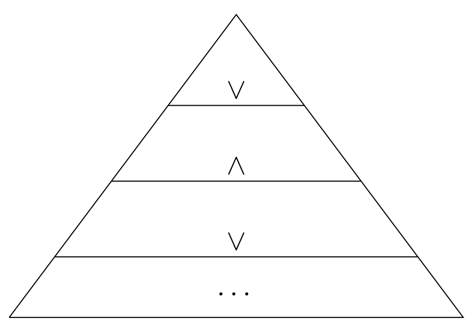


Figure 1: Computation of an Σ_k-machine on some input.

Definition 2. $\Sigma_k TIME(T(n))$ = class of languages computed by an Σ_k-machine in time $O(T(n))$. $\Pi_k TIME(T(n))$ is defined analogously.

Definition 3. $\Sigma_k^P = \Sigma_k TIME(n^{O(1)})$ and $\Pi_k^P = \Pi_k TIME(n^{O(1)})$

The polynomial hierarchy:

$$PH = \bigcup_{k=1}^{\infty} \Sigma_k^P$$

Proposition 4. $NP = \Sigma_1^P$ *and* $coNP = \Pi_1^P$.

Proof. By comparing definitions. $\square$

Proposition 5. $co - \Sigma_k^P = \Pi_k^P$.

Proof. If L is computed by a Σ_k-machine swap $\vee$ and $\wedge$ types, swap accept and reject. Result is a Π_k-machine, accepting $\bar{L}$. $\square$

Proposition 6. $\Sigma_k^P \subseteq \Sigma_{k+1}^P$ *and* $\Sigma_k^P \subseteq \Pi_{k+1}^P$.

Proof. These are all syntactic inclusions, see Figure 2

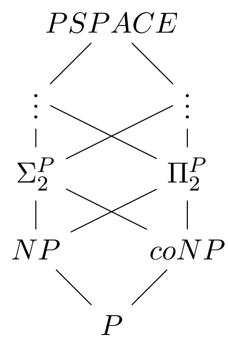


Figure 2: Syntactic inclusions in the polynomial hierarchy.

$\square$

Notation: Let p be a polynomial and $x \in \Sigma^*$ for some alphabet Σ. We write:

$$\exists^p y := \exists y \in \Sigma^*, |y| \leq p(|x|)$$

Equivalently

$$\forall^p y := \forall y \in \Sigma^*, |y| \leq p(|x|)$$

Proposition 7 (Certificates definition of Σ_k^P)**.** $L \in \Sigma_k^P$ *if and only if there is a polynomial* p *and some* $R \in P$ *such that:*

$$\forall x : x \in L \iff \exists^P y_1 \forall^P y_2 \exists^P y_3 \ldots Q^P y_k : \langle x, y_1, \ldots, y_k \rangle \in R \tag{1}$$

($Q \equiv \exists$ if k is odd, $Q \equiv \forall$ if k is even.)

Proof. Assume $L \in \Sigma_k^P$ and let M be Σ_k-machine accepting L in polynomial time $p(n)$. Define $R := \{\langle x, y_1, y_2, \ldots, y_k \rangle |$ if y_i is used to determine the nondeterministic choices in interval i, then M accepts x $\}$.

$R \in P$ and (1) holds by definition of EA (eventually accepts) for ATMs.

Conversely, given R as defined above, we can decide L using an alternating TM, guessing y_1 using $\vee$ states, y_2 using $\wedge$ states, etc. $\square$

Definition 8 ($\Sigma_k CircuitSAT$)**.**

$$\Sigma_k CircuitSAT = \{Boolean\ circuit\ C(y_1, \ldots, y_k) | \exists y_1 \forall y_2 \cdots Q y_k : C(y_1, \ldots, y_k) = 1\}$$

Analogously:

$$\Pi_k CircuitSAT = \{Boolean\ circuit\ C(y_1, \ldots, y_k) | \forall y_1 \exists y_2 \cdots Q y_k : C(y_1, \ldots, y_k) = 1\}$$

Theorem 9 (Completeness of $\Sigma_k CircuitSAT$)**.** *$\Sigma_k CircuitSAT$ is $\leq_m^{log}$-complete for Σ_k^P. $\Pi_k CircuitSAT$ is $\leq_m^{log}$-complete for Π_k^P.*

Proof. **Containment:** $\Sigma_k CircuitSAT \in \Sigma_k^P$ by use of the definitions.

Hardness: As in Cook's theorem. Let $L \in \Sigma_k^P$ be given by a polynomial p and $R \in P$, as in Proposition 7. Reduction: On input x, construct circuit for R on $|\langle x, y_1, \ldots, y_k \rangle|$ number of inputs. Then hardwire x into the circuit.

□

Reminder on *NAESAT***.** Not All Equal SAT is an NP-complete problem.
Instance: 3CNF formula ϕ.
Question: Is there an assignment that gives each clause of ϕ a true and a false literal?
Reduction: CircuitSAT $\leq$ NAESAT:

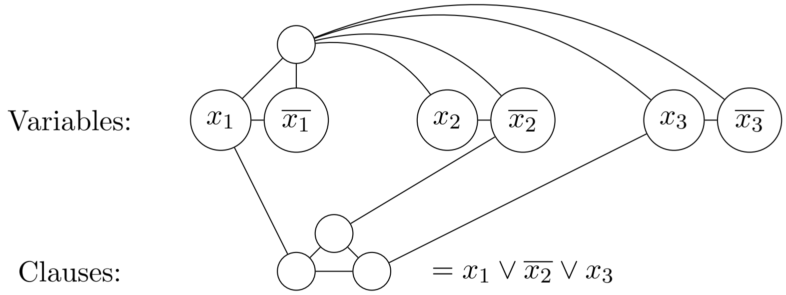
- Add fresh variable for each gate of circuit, and a clause for output.
- Express that each gate computes correctly by clauses of length 2 and 3.
- These have the property: If they are all satisfied, all length 3 clauses have both a true and false literal.
- Add a new fresh global variable to all length 2 clauses.

Definition 10 ($\forall\exists$*NAE3SAT*)**.** Instance: $C_1, \ldots, C_n$ clauses of length 3 in variables $x_1, \ldots x_k$ and $y_1, \ldots y_k$.
Question: $\forall x_1, \ldots x_k \exists y_1, \ldots, y_k$: All C_i has *both* a true and a false literal.

Theorem 11 (Completeness of $\forall\exists$*NAE3SAT*)**.** *$\forall\exists$NAE3SAT is Π_2^P-complete.*

Proof. Take Π_2CircuitSAT instance. Translate circuit to clauses as in CircuitSAT $\leq$ NAESAT reduction. All added fresh variables are now $\exists$-quantified. □

Reminder on *3COL***.** 3-Coloring problem is an NP-complete problem.
Instance: Graph $G = (V, E)$.
Question: Can G be colored by 3 colors?
Reduction: NAESAT $\leq$ 3COL



Problem: 2-Color Extension is Π_2^P-complete

Instance: Graph $G = (V, E)$, set $S \subseteq V$.
Question: Can *any* 2 coloring of the nodes in S be extended to a 3-coloring of G.
Reduction: $\forall\exists NAE3SAT \leq$ 2-color extension:

- Make graph from clauses just like NAESAT $\leq$ 3COL reduction.
- Let S be set of vertices corresponding to the universally quantified variables $x_1, \ldots, x_k$ and negations (i.e. those giving their truth assignment).

Definition 12 (Oracle Turing Machines)**.** Let $A \subseteq \Sigma^*$ be an *arbitrary* language.

Extend Turing machine model to have *oracle access to* A by:

- Oracle query tape
- New states $q_{O-?}$, q_{O-yes}, q_{O-no}.

Modification of operation: Suppose machine enters state $q_{O-?}$:

- Let y be contents of query tape
- Query tape is erased
- If $y \in A$ then machine restarts in q_{O-yes}
- If $y \notin A$ then machine restarts in q_{O-no}

at no computation cost.

We can now define oracle versions of our ususal complexity classes.

Definition 13.

- P^A = class of languages computed in polynomial time with oracle access to the language A.
- NP^A = class of languages computed in nondeterministic polynomial time with oracle access to the language A.
- Etc.

Let $\mathcal{C}$ be a class of languages. Define $P^{\mathcal{C}} = \bigcup_{B \in \mathcal{C}} P^B$ and $NP^{\mathcal{C}} = \bigcup_{B \in \mathcal{C}} NP^B$, etc.

We will now show that $\Sigma_k^P = NP^{NP^{\cdots^{NP}}}$ with $k-1$ levels of oracles. Similarly $\Pi_k^P = coNP^{NP^{\cdots^{NP}}}$ with $k-1$ levels of oracles.

Theorem 14. *Define $NP_1 := NP$ and $NP_{k+1} := NP^{NP_k}$ for $k \geq 1$.*
Then $\Sigma_k^P = NP_k$.

Proof. Induction in k. Base case holds by definition. Induction case:

$\Sigma_k^P \subseteq NP_k$: Assume $L \in \Sigma_k^P$. Then there is a polynomial p and a $R \in P$ s.t.

$$\forall x : x \in L \iff \exists^P y_1 \forall^P y_2 \exists^P y_3 \dots Q^P y_k : \langle x, y_1, \dots, y_k \rangle \in R$$

Define L': $(x, y_1) \in L' \iff \forall^P y_2 \exists^P y_3 \dots Q^P y_k : \langle x, y_1, \dots, y_k \rangle \in R$. Then $L' \in \Pi_{k+1}^P$, and thus $\overline{L'} \in \Sigma_{k+1}^P$. A nondeterministic Turing machine for L with oracle access to $\overline{L'} \in NP_k$:

- Guess y_1 using nondeterminism, then query oracle for the rest.

$NP_k \subseteq \Sigma_k^P$: Now let $L \in NP_k$. Let M be a nondeterministic Turing machine with oracle in $NP_{k-1} = \Sigma_{k-1}^P$ accepting L. Construct Σ_k-machine that does:

- Guess an accepting path of M using $\vee$-states, including answers to oracle-queries. (*Also* guess first existential phase of all Σ_{k-1}^P computations for use in (*)).
- Let $y_1, \dots, y_s$ be the queries where we guessed "yes"
- Let $w_1, \dots, w_t$ be the queries where we guessed "no"
- Branch $s + t$ way universally using $\wedge$-states.
- (*) To verify $y_1, \dots, y_s$ is a Σ_{k-1} computation: Use previous guesses to guide first phase. Rest is Π_{k-2}: just run it.
- To verify $w_1, \dots, w_t$ is a Π_{k-1} computation: just run it.

$\square$