CS 6810: Theory of Computing **Fall 2021**

Lecture 16: The Polynomial Hierarchy

Lecturer: Eshan Chattopadhyay *Scribe: Owen Oertell*

Announcements: Project report is due end of day on Friday.

We first consider the problem MIN-DNF. Which is just the set of DNFs such that there exists no smaller DNF which is equivalent.

We know that SAT clearly sit in NP, but is hard for P. Thus naturally the question as to where MIN-DNF sits arises. We can rephrase this problem as to: If ϕ has some size (based on an agreed upon encoding) and there exits some ψ which has a smaller size, and both are in MIN-DNF, then for some z we have that $\phi(z) \neq \psi(z)$. Written formally, if

$$\phi \in \text{MIN-DNF iff } \forall \psi, |\psi| < |\phi| \implies \exists z, \phi(z) \neq \psi(z)$$

.

Question: Is MIN-DNF in NP? P? PSPACE?

Clearly, we see that it is in PSPACE since we can enumerate all the possible DNFs to find those in the language, while reusing the space of the input. However, it is unclear whether or not it is in NP or co-NP since we need the power of $\forall$ and $\exists$ in order to express the language. Therefore we almost want some mix between the two.

Definition 0.1 ($\exists c$)**.** *Let c be any complexity class. Define $\exists c$ such that $L \in \exists c$ if for some $L' \in C$ iff $\exists y \in \{0,1\}^{poly(|x|)}, (x,y) \in L'$.*

Similarly, we define $\forall c$,

Definition 0.2 ($\forall c$)**.** *Let c be any complexity class. Define $\forall c$ such that $L \in \forall c$ if for some $L' \in C$ iff $\forall y \in \{0,1\}^{poly(|x|)}, (x,y) \in L'$.*

There are a few immediate observations. Particularly, $\exists P = NP$. Further, $\forall P = coNP$. Both of these observations follow from the definition of NP and co-NP respectively. Also we note that adding repeated exists does not increase the power of the complexity class just as adding more forall's does not increase the power of the complexity class ($\exists\exists P = \exists P$). Finally, we note that $c \subseteq \exists c, \forall c$.

Polynomial Hierarchy

We now define the Polynomial Hierarchy which is constructed by repeatedly applying these quantifiers.

We define the base of the Hierarchy as

$$\Sigma_0 = \Pi_0 = P$$

Then we define inductively

$$\Sigma_i = \exists \Pi_{i-1} \text{ and } \Pi_i = \forall \Sigma_{i-1}$$

.

Finally, we define the complexity class

$$\mathsf{PH} = \bigcup_{i \in \mathbb{N}} \Sigma_i = \bigcup_{i \in \mathbb{N}} \Pi_i$$

.

Observation 0.3. $\Pi_i \subseteq \Sigma_{i+1} \subseteq \Pi_{i+2}$

This observation follows previously by the fact that $c \subseteq \forall c, \exists c$.

We can construct a pictorial hasse diagram of the polynomial hierarchy as follows:

$$\begin{tikzcd}
\Sigma_3 \arrow[d, no head] \arrow[drr, no head] & & \Pi_3 \arrow[d, no head] \arrow[dll, no head] \\
\Sigma_2 \arrow[d, no head] \arrow[drr, no head] & & \Pi_2 \arrow[d, no head] \arrow[dll, no head] \\
\Sigma_1 = \mathsf{NP} \arrow[dr, no head] & & \Pi_1 = \mathsf{co\text{-}NP} \arrow[dl, no head] \\
& \mathsf{P} = \Sigma_0 = \Pi_0 &
\end{tikzcd}$$

Claim 0.4. $\forall i, \Pi_i = \forall \Sigma_{i-1}$

Pf. This follows by induction. Since we know that $\pi_i = \forall \Sigma_{i-1}$, let $L \in \Pi_i$ by definition, we have that $x \in L$ iff $\forall y, (x, y) \notin L'$. This is equivalent to $x \in \bar{L}$ iff $\forall y, (x, y) \in \bar{L}$. But we know that $\bar{L'} \in \Pi_{i-1}$ Therefore, $\bar{L'} \in \exists \Pi_{i-1} = \Sigma_i$

We return to the question, where does MIN-DNF sit in the Polynomial hierarchy. However, naturally because of the quantifiers used in the formulation of the problem, it would be in Π_2 of the Polynomial Hierarchy.

Claim 0.5. *If $\Sigma_i = \Pi_i$,then $PH = \Sigma_i$ (i.e. the polynomial hierarchy collapses)*

Pf. By definition we have that

$$\Sigma_{i=1} = \forall \Pi_i$$

but by assumption, we have that this is equal to

$$= \exists \Pi_i = \exists \Sigma_i = \Sigma_i$$

Similarly,

$$\Pi_{i+1} = \forall \Sigma_i = \forall \Pi_i = \Pi_i$$

(again, since we know repeated for all and repeated exists add no power to the complexity class)

Therefore we have that $\Pi_{i+1} = \Pi_{i+1} = \Sigma_i$.

It is clear that as a corollary,

Corollary 0.6. *$NP = coNP$, then $PH = NP = coNP$*

We also note that if $P = NP$ then the polynomial time hierarchy collapses to level 0.

That is, Since we know that $P = NP \implies \mathsf{NP} = \mathsf{coNP}$ which implies that PH collapses to level 1. However since $P = NP$ it therefore collapses to level 0.

PH vs PSPACE

Complete Problems

We consider what the natural complete problem would be for the polynomial hierarchy.

For Σ_i, we have that the natural complete problem is

$$\Sigma_i\text{-SAT} = \{\phi \mid \exists y_1 \forall y_2 \exists y_3 \ldots \phi(x, y_1, y_2, y_3, \ldots)\}$$

Similarly, for Π_i, we have that the natural complete problem is

$$\Pi_i\text{-SAT} = \{\phi \mid \forall y_1 \exists y_2 \forall y_3 \ldots \phi(x, y_1, y_2, y_3, \ldots\}$$

Question: We consider the question therefore, does PH have a complete problem like some other classes?

We believe the answer is no. If there were to exist a complete problem, then it collapses. Suppose that L is a complete language for PSPACE. Then since $PH = \bigcup \Sigma_i$, $L \in \Sigma_i$ for some i. Thus by definition we have that for any $L' \in PH$, $L' \leq_p L$. This $x \in L' \iff f(x) \in L$ for some f computable in P.

We further believe that $PH \subsetneq PSPACE$. Specifically, this is known because of the fact that each of these complete problems are in PSPACE (we just try each of the possibilities and remember when we stop). However since PSPACE has a complete language (TQBF), for example, we believe that this is a strict subset.

Alternating Turning Machines

We again consider the idea that there are two transition functions. δ_0, δ_1.

Every state of M has a label $\exists, \forall$.

For input x, let $G_{M,x}$ be the configuration graph of M on x. We will iteratively build up a set $ACCEPT$ and accept if the START node is in that set.

Note that the labels from the states in the TM turn into labels on the states in the configuration graph.

We follow this procedure to build up $ACCEPT$ until we hit a steady state (no more accept nodes can be added) or the start state is added to the set.

1. Initially, $ACCEPT \leftarrow 0$
2. Nodes with the accept state reached in the TM M get $ACCEPT$.
3. Further, we repeat until no more changes can be made:
   (a) if V is $\exists$ and has a neighbor in $ACCEPT$ then add v.
   (b) if V is $\forall$ and all neighbors in $ACCEPT$, then add v.
4. M accepts x if start node in $G_{M,x} \in ACCEPT$.

We will continue the discussion on Thursday.