Computer Science 511

Beyond $\mathcal{NP}$: Introduction to the Polynomial Hierarchy

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #14

Goals for Today

- Presentation of ***The Polynomial Hierarchy*** — a hierarchy of complexity classes that is useful for comparing various computational problems (and associated decision problems) that are related to languages in $\mathcal{NP}$, but appear to be more difficult
- Presentation of properties and conjectures about the complexity classes in this hierarchy — including some that will be related to more “natural” questions about computational complexity that will be considered later.

A Motivating Problem

- Recall the "k-Clique" problem, which concerns whether a given undirected graph has a clique of size at (at least) k, for a given positive integer k.
- This was used to define an $\mathcal{NP}$-complete language, $L_{k\text{-Clique}}$.
- Consider a *related* question: For a given undirected graph G, and a given positive integer k, does the ***largest*** clique in G have size exactly k?
- The language of instances of this problem is the same as the language, $L_{\text{Graph+Bound}}$, of instances for the "k-Clique" problem.
- The language $L_{\text{Exact-}k\text{-Clique}}$ of "Yes-instances", associated with this decision problem, does not seem to be in $\mathcal{NP}$. It does not seem to be in co-$\mathcal{NP}$, either.
- We will return to this language shortly...

Alternating Turing Machines

An ***Alternating Turing machine*** is another variant of a Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

such that Q, Σ, Γ, q_0, q_{accept}, q_{reject}, are as usually defined.

- As with a nondeterministic Turing machines, there can be zero, one or *many* transitions that can be made so that — if this is a one-tape Turing machine —

$$\delta : (Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathtt{L}, \mathtt{R}\})$$

- Both ***single-tape*** and ***multi-tape*** alternating Turing machines can be considered.

Alternating Turing Machines

- Every non-halting state is either an ***existential state*** (an $\vee$-state) or a ***universal state*** (a $\wedge$-state).
- As with nondeterministic Turing machine a ***computation*** of an alternating Turing machine M on an input string $\omega \in \Sigma^\star$ can be modelled as a ***computation tree*** — a rooted tree with the usual ***start configuration*** for M and ω at the root.

Alternating Turing Machines

Each configuration in this tree is either ***accepting***, ***rejecting***, or ***looping***.

- If the configuration includes the accepting state q_{accept} (so this is at a *leaf* in the computation tree) then this is an ***accepting configuration***.
- If the configuration includes the rejecting state q_{reject} (so that, once again, this is at a *leaf* in the computation tree) then this is a ***rejecting configuration***.

Alternating Turing Machines

Otherwise, a *recursive definition* is used to determine whether a configuration is accepting, rejecting, or looping:

- If a configuration includes an ***existential*** state and some ***child*** of this in the computation tree is an accepting configuration, then this is an *accepting* configuration too.

  Otherwise this is a *rejecting* configuration if the subtree with this configuration as root is finite, and it is a *looping* configuration otherwise.

  Special Case: It follows that if this configuration *has* no children, this is a rejecting configuration.

Alternating Turing Machines

- If a configuration includes a ***universal*** state and ***every*** child of this configuration is an accepting configuration then this is an *accepting* configuration too.

  Otherwise this is a rejecting configuration if every child of this configuration is a *rejecting* configuation — so that the subtree of the computation tree with this node as root is finite — and it is a *looping* configuration otherwise.

  Special Case: It follows that if this configuration *has* no children, then this is an *accepting configuration*.

Alternating Turing Machines

If M is an alternating Turing machine with input alphabet Σ and $\omega \in \Sigma^\star$, then...

- $\mathcal{M}$ ***accepts*** ω if the configuration at the root of the computation tree for M and ω is an accepting configuration;
- $\mathcal{M}$ ***rejects*** ω if the configuration at the root of the computation tree for $\mathcal{M}$ and ω is a rejecting configuration, and
- $\mathcal{M}$ ***loops*** on ω otherwise.

Alternating Turing Machines

- If M is an alternating Turing machine with input alphabet Σ then (as usual) the ***language $L(M)$ of M*** is the set of strings
$$L(M) = \{\omega \in \Sigma^\star \mid M \text{ accepts } \omega\}$$
- M ***recognizes*** a language L if $L = L(M)$.

Alternating Turing Machines

- If $\mathcal{M}$'s input alphabet is Σ then $\mathcal{M}$ ***decides*** a language $L \subseteq \Sigma^\star$ if the following three conditions are satisfied:
  (a) $\mathcal{M}$ ***accepts*** every string $\omega \in \Sigma^\star$ such that $\omega \in L$.
  (b) $\mathcal{M}$ ***rejects*** every string $\omega \in \Sigma^\star$ such that $\omega \notin L$.
  (c) The computation tree for $\mathcal{M}$ and ω is ***finite*** for every string $\omega \in \Sigma^\star$.
- From now on we will only consider alternating Turing machines that ***decide*** languages.

Alternating Turing Machines

- A ***deterministic one-tape Turing machine***

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

is easily "turned into" an alternating one-tape Turing machine, with the same language,

$$\widehat{M} = (Q, \Sigma, \Gamma, \widehat{\delta}, q_0, q_{\text{accept}}, q_{\text{reject}})$$

by setting $\widehat{\delta}(q, \sigma)$ to be $\{\delta(q, \sigma)\}$ for every state $q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$ and every symbol $\sigma \in \Gamma$.

- A deterministic k-tape Turing machine is easily "turned into" an alternating k-tape Turing machine, with the same language, in essentially the same way.

Alternating Turing Machines

- A ***nondeterministic Turing machine***

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

is easily “turned into” an alternating Turing machine, with the same language, by making no changes to M, at all — and setting each of the states of M to be an ***existential*** state.

Alternating Turing Machines

- If $L \subseteq \Sigma^\star$, and

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

is a nondeterministic Turing machine that *decides* L, then an alternating Turing machine

$$\widehat{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{reject}}, q_{\text{accept}})$$

that decides the *complement* L^C of L is obtained by switching the accepting and rejecting states — and setting each of the states of $\widehat{M}$ to be a ***universal*** state.

Alternating Turing Machines

- The ***time*** used by an alternating Turing machine M, on an input string ω, is the depth of the computation tree for M and ω.
- If $f : \mathbb{N} \to \mathbb{N}$ is a total function, then ATIME($f(n)$) is the set of languages that are decidable by alternating Turing machines using time in $O(f(n))$ for every input string with length n.
- The relationships between nondeterministic Turing machines and alternating Turing machines, given, above, can be used to establish that

$$\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)) \subseteq \text{ATIME}(f(n))$$

for every total function $f : \mathbb{N} \to \mathbb{N}$.

Alternating Turing Machines

Definition:

$$\mathcal{AP} = \bigcup_{k \in \mathbb{N}} \mathsf{ATIME}(n^k).$$

- It follows, by the above, that

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{AP}.$$

Alternating Turing Machines

- The proof of Claim #3, from Lecture #8, can be modified to show that, for every function $f : \mathbb{N} \to \mathbb{N}$ and for every language $L \subseteq \Sigma^\star$ such that $L \in \mathsf{ATIME}(f)$, there exists an integer constant c (depending on L) such that $L \in \mathsf{TIME}(c^f)$. Thus

$$\mathsf{ATIME}(f) \subseteq \bigcup_{c \in \mathbb{N}} \mathsf{TIME}(c^f).$$

- This can be used to establish that

$$\mathcal{AP} \subseteq \mathit{EXPTIME}.$$

Alternating Turing Machines

Now consider the following process, given a string $\mu \in \Sigma_G^\star$.

1. Deterministically check whether $\mu \in L_{\text{Graph+Bound}}$ — ***rejecting*** μ if this is not the case. Let $G = (V, E)$ be the undirected graph and let k be the positive integer that are encoded by μ, otherwise.
2. ***Reject*** μ if $k > |V|$. Otherwise — using ***existential*** states — nondeterministically "guess" a subset $C \subseteq V$ such that $|C| = k$. Then deterministically check whether C is a clique in G — ***rejecting*** μ, if this is not the case.
3. If $k = |V|$ then ***accept*** μ. Otherwise, use ***universal*** states to give a subset $\widehat{C} \subseteq Q$ such that $|\widehat{C}| = k + 1$. Then deterministically check whether $\widehat{C}$ is a clique in G — ***rejecting*** μ if this *is* the case, and ***accepting*** μ, otherwise.

Alternating Turing Machines

- Since $L_{\text{Graph+Bound}} \in \mathcal{P}$ step #1 can be carried out deterministically in polynomial time. Furthermore, if cliques (and other subsets of V) are encoded as described in Lecture #12 then the deterministic part of steps #2 and #3 can also be carried out deterministically, using time that is at most polynomial in the length of the input string.
- Indeed, this algorithm can be implemented using an alternate Turing machine that uses time in the length of its input string — so that it decides a language $L \subseteq \Sigma_G^{\star}$ such that $L \in \mathcal{AP}$.

Alternating Turing Machines

- Since this algorithm only accepts when an input graph has a clique with size k but does not have a clique with size $k+1$, this Turing machine decides the language $L_{\text{Exact-}k\text{-Clique}}$. Thus

$$L_{\text{Exact-}k\text{-Clique}} \in \mathcal{AP}.$$

Polynomial Hierarchy

Let *i* be an integer such that $i \geq 1$.

Definition: A ***Σ_i-Alternating Turing machine*** is an alternating Turing machine is an alternating Turing machine, with some input alphabet $\Sigma^\star$, such that

- The start state is an ***existential state***, and
- There are *at most* $i - 1$ alternations between existential states and universal states, down any branch of the computation tree for ω, for any input string $\omega \in \Sigma^\star$.

The definition of a ***Π_i-Alternating Turing machine*** is the same, except that the start state is a ***universal state*** instead of an existential state.

Polynomial Hierarchy

Now let i be a positive integer and let $f : \mathbb{N} \to \mathbb{N}$ be a total function.

Definition: Σ_i-TIME$(f(n))$ is the set of languages $L \subseteq \Sigma^\star$ (for some input alphabet Σ) that can be decided using Σ_i-Alternating Turing machines using time in $O(f(n))$ in the worst case.

$$\Sigma_i\mathcal{P} = \bigcup_{k \geq 1} \Sigma_i\text{-TIME}(n^k).$$

Polynomial Hierarchy

Once again, let i be a positive integer and let $f : \mathbb{N} \to \mathbb{N}$ be a total function.

Definition: Π_i-TIME$(f(n))$ is the set of languages $L \subseteq \Sigma^\star$ (for some input alphabet Σ) that can be decided using Π_i-Alternating Turing machines using time in $O(f(n))$ in the worst case.

$$\Pi_i\mathcal{P} = \bigcup_{k \geq 1} \Pi_i\text{-TIME}(n^k).$$

Polynomial Hierarchy

Definition:

$$\mathcal{PH} = \bigcup_{i \geq 1} \Sigma_i \mathcal{P}.$$

- $\mathcal{PH}$ stands for ***Polynomial Hierarchy,*** and this is the standard name for the collection of complexity classes $\Sigma_i\mathcal{P}$ and $\Pi_i\mathcal{P}$, for $i \geq 1$, that have just been defined — along with $\mathcal{PH}$.
- Since $\Sigma_i\mathcal{P} \subseteq \mathcal{AP}$ for every integer $i \geq 1$,

$$\mathcal{PH} \subseteq \mathcal{AP}$$

as well.

Polynomial Hierarchy

Each of the following are easily proved:

(a) $\Sigma_1\mathcal{P} = \mathcal{NP}$ and $\Pi_1\mathcal{P} =$ co-$\mathcal{NP}$.

(b) $\Pi_i\mathcal{P} =$ co-$\Sigma_i\mathcal{P}$ for every positive integer i.

(c) $\Sigma_i\mathcal{P} \cup \Pi_i\mathcal{P} \subseteq \Sigma_{i+1}\mathcal{P} \cap \Pi_{i+1}\mathcal{P}$ for every positive integer i.

Polynomial Hierarchy

- The following is believe but not proved.

  Conjecture: $\mathcal{PH}$ is an infinite hierarchy. In particular, that

$$\Sigma_i\mathcal{P} \subsetneq \Sigma_{i+1}\mathcal{P} \subsetneq \mathcal{PH}$$

  for every integer $i \geq 1$.
- Properties (b) and (c), on the previous slide can be used to show that this conjecture would imply that

$$\Pi_i\mathcal{P} \subsetneq \Pi_{i+1}\mathcal{P} \subsetneq \mathcal{PH}$$

  as well.

Why Do We Care About the Polynomial Hierarchy?

Future lectures will consider complexity classes defined using two more “realistic” models:

- Computations using families of Boolean circuits
- Randomized computations

It turns out that the assumption that the Polynomial Hierarchy is an infinite hierarchy has implications concerning *these* complexity classes — and this is the reason why it is (still) included in this course.