TECHNISCHE UNIVERSITÄT DRESDEN

COMPLEXITY THEORY

Lecture 17: The Polynomial Hierarchy

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 10th Dec 2019

Review: ATM vs. DTM

ATM vs. DTM

We have observed four major relationships between alternating and deterministic complexity classes.

ATM vs. DTM

We have observed four major relationships between alternating and deterministic complexity classes. For the special case of polynomial bounds, we got:

$$\text{APTime} \subseteq \text{PSpace}$$

How? Deterministic depth-first search on ATMs computation tree.

ATM vs. DTM

We have observed four major relationships between alternating and deterministic complexity classes. For the special case of polynomial bounds, we got:

$$\text{APTime} \subseteq \text{PSpace}$$

How? Deterministic depth-first search on ATMs computation tree.

$$\text{APTime} \supseteq \text{PSpace}$$

How? Use alternation to implement Savitch-style middle-first search in polyspace.

ATM vs. DTM

We have observed four major relationships between alternating and deterministic complexity classes. For the special case of polynomial bounds, we got:

$$\text{APTime} \subseteq \text{PSpace}$$

How? Deterministic depth-first search on ATMs computation tree.

$$\text{APTime} \supseteq \text{PSpace}$$

How? Use alternation to implement Savitch-style middle-first search in polyspace.

$$\text{APSpace} \subseteq \text{ExpTime}$$

How? Analyse the exponential ATM configuration graph deterministically.

ATM vs. DTM

We have observed four major relationships between alternating and deterministic complexity classes. For the special case of polynomial bounds, we got:

$$\text{APTime} \subseteq \text{PSpace}$$

How? Deterministic depth-first search on ATMs computation tree.

$$\text{APTime} \supseteq \text{PSpace}$$

How? Use alternation to implement Savitch-style middle-first search in polyspace.

$$\text{APSpace} \subseteq \text{ExpTime}$$

How? Analyse the exponential ATM configuration graph deterministically.

$$\text{APSpace} \supseteq \text{ExpTime}$$

How? Re-trace exponential computation path by verifying local changes.

From Deterministic Time To Alternating Space

Let $h : \mathbb{N} \to \mathbb{R}$ be a function in $O(g)$ that defines the exact time bound for $\mathcal{M}$ (no O-notation), and that can be computed in space $O(\log g)$.

```
ATMSIMULATETM(TM M, input word w, time bound h) :
  existentially guess s ≤ h(|w|)        // halting step
  existentially guess i ∈ {0, ..., s}   // halting position
  existentially guess ω ∈ Q × Γ         // halting cell + state
  if M would not halt in ω :
     return false
  for j = s, ..., 1 do :
     existentially guess ⟨ω_{-1}, ω_0, ω_1⟩ ∈ Ω^3
     if M(ω_{-1}, ω_0, ω_{+1}) ≠ ω :
        return false
     universally choose ℓ ∈ {-1, 0, 1}
     ω := ω_ℓ
     i := i + ℓ
  // after tracing back s steps, check input configuration:
  return "input configuration of M on w has ω at position i"
```

A Remark on (Non)determinism

For each cell that is to be verified:

- we guess three predecessor cells,
- which we then verify recursively.

$\leadsto$ The contents of the same cell is guessed in several places of the ATM computation tree (“in several recursive subprocesses”)

A Remark on (Non)determinism

For each cell that is to be verified:

- we guess three predecessor cells,
- which we then verify recursively.

$\leadsto$ The contents of the same cell is guessed in several places of the ATM computation tree ("in several recursive subprocesses")

If processes do not exchange information,
how do we know that the guesses are not contradicting each other?

A Remark on (Non)determinism

For each cell that is to be verified:

- we guess three predecessor cells,
- which we then verify recursively.

$\leadsto$ The contents of the same cell is guessed in several places of the ATM computation tree ("in several recursive subprocesses")

If processes do not exchange information,
how do we know that the guesses are not contradicting each other?

Because of determinism:

- The simulated TM is deterministic
- Hence, if the starting point is determined, every future cell in every position is determined too
- Therefore, for every cell, there is only one possible guess that eventually leads to the right input tape

$\leadsto$ Independent guesses, if correct, must generally be the same

A Remark on Space-Constructibility

Our algorithm needs to compute h in logarithmic space w.r.t. its absolute value to implement the line

```
existentially guess s ≤ h(|w|)    // halting step
```

A Remark on Space-Constructibility

Our algorithm needs to compute h in logarithmic space w.r.t. its absolute value to implement the line

```
02    existentially guess s ≤ h(|w|)    // halting step
```

However, we could also avoid this:

- The algorithm from line `03` on checks if the TM halts after s steps
- We can make a similar algorithm that checks if the TM does not halt after s steps
- We can then use an overall algorithm that increments s one by one (starting from 1):
  - For each value of s, guess if the TM halts after this time or not
  - Check the guess using the above procedures
  - Stop when the halting configuration has been found
- Because of the time bound on the simulated TM, s will not become larger than $2^{O(f)}$ here, so we can always store it in space f.

Summary: Alternating vs. Deterministic Classes

We can sum up our findings as follows:

$$\begin{array}{ccccccccc} \text{L} & \subseteq & \text{PTime} & \subseteq & \text{PSpace} & \subseteq & \text{ExpTime} & \subseteq & \text{ExpSpace} \\ & & \parallel & & \parallel & & \parallel & & \parallel \\ & & \text{ALogSpace} & \subseteq & \text{APTime} & \subseteq & \text{APSpace} & \subseteq & \text{AExpTime} \end{array}$$

The Polynomial Hierarchy

Bounding Alternation

For ATMs, alternation itself is a resource. We can distinguish problems by how much alternation they need to be solved.

We first classify computations by counting their quantifier alternations:

Definition 17.1: Let $\mathcal{P}$ be a computation path of an ATM on some input.

- $\mathcal{P}$ is of type Σ_1 if it consists only of existential configurations (with the exception of the final configuration)
- $\mathcal{P}$ is of type Π_1 if it consists only of universal configurations
- $\mathcal{P}$ is of type Σ_{i+1} if it starts with a sequence of existential configurations, followed by a path of type Π_i
- $\mathcal{P}$ is of type Π_{i+1} if it starts with a sequence of universal configurations, followed by a path of type Σ_i

Alternation-Bounded ATMs

We apply alternation bounds to every computation path:

Definition 17.2: A Σ_i Alternating Turing Machine is an ATM for which every computation path on every input is of type Σ_j for some $j \leq i$.
A Π_i Alternating Turing Machine is an ATM for which every computation path on every input is of type Π_j for some $j \leq i$.

Note that it's always ok to use fewer alternations ("$j \leq i$") but computation has to start with the right kind of quantifier ($\exists$ for Σ_i and $\forall$ for Π_i).

Example 17.3: A Σ_1 ATM is simply an NTM.

Alternation-Bounded Complexity

We are interested in the power of ATMs that are both time/space-bounded and alternation-bounded:

Definition 17.4: Let $f : \mathbb{N} \to \mathbb{R}^+$ be a function. $\Sigma_i\mathrm{Time}(f(n))$ is the class of all languages that are decided by some $O(f(n))$-time bounded Σ_i ATM. The classes $\Pi_i\mathrm{Time}(f(n))$, $\Sigma_i\mathrm{Space}(f(n))$ and $\Pi_i\mathrm{Space}(f(n))$ are defined similarly.

The most popular classes of these problems are the alternation-bounded polynomial time classes:

$$\Sigma_i\mathrm{P} = \bigcup_{d \geq 1} \Sigma_i\mathrm{Time}(n^d) \qquad \text{and} \qquad \Pi_i\mathrm{P} = \bigcup_{d \geq 1} \Pi_i\mathrm{Time}(n^d)$$

Hardness for these classes is defined by polynomial many-one reductions as usual.

Basic Observations

Theorem 17.5: $\Sigma_1\mathrm{P} = \mathrm{NP}$ and $\Pi_1\mathrm{P} = \mathrm{coNP}$.

Proof: Immediate from the definitions. □

Basic Observations

Theorem 17.5: $\Sigma_1\mathrm{P} = \mathrm{NP}$ and $\Pi_1\mathrm{P} = \mathrm{coNP}$.

Proof: Immediate from the definitions. □

Theorem 17.6: $\mathrm{co}\Sigma_i\mathrm{P} = \Pi_i\mathrm{P}$ and $\mathrm{co}\Pi_i\mathrm{P} = \Sigma_i\mathrm{P}$.

Proof: We observed previously that ATMs can be complemented by simply exchanging their universal and existential states. This does not affect the amount of time or space needed. □

Example

MinFormula

Input: A propositional formula φ.

Problem: Is φ the shortest formula that is satisfied by the same assignments as φ?

One can show that **MinFormula** is Π_2P-complete. Inclusion is easy:

```
MinFormula(formula φ) :
  universally choose ψ := formula shorter than φ
  existentially guess I := assignment for variables in φ
  if φ^I = ψ^I :
    return false
  else :
    return true
```

The Polynomial Hierarchy

Like for NP and coNP, we do not know if Σ_iP equals Π_iP or not.
What we do know, however, is this:

Theorem 17.7:

- $\Sigma_i\mathsf{P} \subseteq \Sigma_{i+1}\mathsf{P}$ and $\Sigma_i\mathsf{P} \subseteq \Pi_{i+1}\mathsf{P}$
- $\Pi_i\mathsf{P} \subseteq \Pi_{i+1}\mathsf{P}$ and $\Pi_i\mathsf{P} \subseteq \Sigma_{i+1}\mathsf{P}$

Proof: Immediate from the definitions. □

Thus, the classes Σ_iP and Π_iP form a kind of hierarchy:
the Polynomial (Time) Hierarchy. Its entirety is denoted PH:

$$\mathsf{PH} := \bigcup_{i \geq 1} \Sigma_i\mathsf{P} = \bigcup_{i \geq 1} \Pi_i\mathsf{P}$$

Problems in the Polynomial Hierarchy

The "typical" problems in the Polynomial Hierarchy are restricted forms of **True QBF**:

True Σ_kQBF

Input:	A quantified Boolean formula φ with at most k quantifier alternations of the form $\exists X_1^1, X_2^1, \cdots \forall X_1^2, X_2^2, \cdots \mathsf{Q}_k X_1^k, X_2^k, \cdots .\psi$.
Problem:	Is φ true?

True Π_kQBF is defined analogously, using formulae with k quantifier alternations that start with $\forall$ rather than $\exists$.

Theorem 17.8: For every k, True Σ_kQBF is Σ_kP-complete and True Π_kQBF is Π_kP-complete.

Note: It is not known if there is any PH-complete problem.

Alternative Views on the Polynomial Hierarchy

Certificates

For NP, we gave an alternative definition based on polynomial-time verifiers that use a given polynomial certificate (witness) to check acceptance. Can we extend this idea to alternation-bounded ATMs?

Certificates

For NP, we gave an alternative definition based on polynomial-time verifiers that use a given polynomial certificate (witness) to check acceptance. Can we extend this idea to alternation-bounded ATMs?

Notation: Given an input word w and a polynomial p, we write $\exists^p c$ as abbreviation for "there is a word c of length $|c| \leq p(|w|)$." Similarly for $\forall^p c$.

We can rephrase our earlier characterisation of polynomial-time verifiers:

L $\in$ NP iff there is a polynomial p and language **V** $\in$ P such that

$$\mathbf{L} = \{w \mid \exists^p c \text{ such that } (w\#c) \in \mathbf{V}\}$$

Certificates for bounded ATMs

Theorem 17.9: $\mathbf{L} \in \Sigma_k\mathrm{P}$ iff there is a polynomial p and language $\mathbf{V} \in \mathrm{P}$ such that

$$\mathbf{L} = \{w \mid \exists^p c_1.\forall^p c_2 \ldots \mathsf{Q}_k^p c_k \text{ such that } (w\#c_1\#c_2\#\ldots\#c_k) \in \mathbf{V}\}$$

where $\mathsf{Q}_k = \exists$ if k is odd, and $\mathsf{Q}_k = \forall$ if k is even.

An analoguous result holds for $\mathbf{L} \in \Pi_k\mathrm{P}$.

Proof sketch:

$\Rightarrow$: Similar as for NP. Use c_i to encode the non-deterministic choices of the ATM. With all choices given, the acceptance on the specified path can be checked in polynomial time.

$\Leftarrow$: Use an ATM to implement the certificate-based definition of **L**, by using universal and existential choices to guess the certificate before running a polynomial time verifier. □

Oracles (Revision)

Recall how we defined oracle TMs:

Definition 3.15: An Oracle Turing Machine (OTM) is a Turing machine $\mathcal{M}$ with a special tape, called the oracle tape, and distinguished states $q_?$, q_{yes}, and q_{no}. For a language **O**, the oracle machine $\mathcal{M}^{\mathbf{O}}$ can, in addition to the normal TM operations, do the following:

Whenever $\mathcal{M}^{\mathbf{O}}$ reaches $q_?$, its next state is q_{yes} if the content of the oracle tape is in **O**, and q_{no} otherwise.

Let C be a complexity class:

- For a language **O**, we write $\mathsf{C}^{\mathbf{O}}$ for the class of all problems that can be solved by a C-TM with oracle **O**.
- For a complexity class O, we write C^{O} for the class of all problems that can be solved by a C-TM with an oracle from class O.

Note: this notation will only be used for complexity classes C where it is clear what a "C-TM with an oracle" is.

The Polynomial Hierarchy – Alternative Definition

We recursively define the following complexity classes:

Definition 17.10:

- Σ_0^{P} := P and $\Sigma_{k+1}^{\mathsf{P}}$:= $\mathsf{NP}^{\Sigma_k^{\mathsf{P}}}$
- Π_0^{P} := P and Π_{k+1}^{P} := $\mathsf{coNP}^{\Pi_k^{\mathsf{P}}}$

The Polynomial Hierarchy – Alternative Definition

We recursively define the following complexity classes:

Definition 17.10:

- $\Sigma_0^{\mathsf{P}} := \mathsf{P}$ and $\Sigma_{k+1}^{\mathsf{P}} := \mathsf{NP}^{\Sigma_k^{\mathsf{P}}}$
- $\Pi_0^{\mathsf{P}} := \mathsf{P}$ and $\Pi_{k+1}^{\mathsf{P}} := \mathsf{coNP}^{\Pi_k^{\mathsf{P}}}$

Remark:

Complementing an oracle (language/class) does not change expressivity: we can just swap states q_{yes} and q_{no}. Therefore $\Sigma_{k+1}^{\mathsf{P}} = \mathsf{NP}^{\Pi_k^{\mathsf{P}}}$ and $\Pi_{k+1}^{\mathsf{P}} := \mathsf{coNP}^{\Sigma_k^{\mathsf{P}}}$.

Hence, we can also see that $\Sigma_k^{\mathsf{P}} = \mathsf{co}\Pi_k^{\mathsf{P}}$.

The Polynomial Hierarchy – Alternative Definition

We recursively define the following complexity classes:

Definition 17.10:

- $\Sigma_0^{\mathsf{P}} := \mathsf{P}$ and $\Sigma_{k+1}^{\mathsf{P}} := \mathsf{NP}^{\Sigma_k^{\mathsf{P}}}$
- $\Pi_0^{\mathsf{P}} := \mathsf{P}$ and $\Pi_{k+1}^{\mathsf{P}} := \mathsf{coNP}^{\Pi_k^{\mathsf{P}}}$

Remark:
Complementing an oracle (language/class) does not change expressivity: we can just swap states q_{yes} and q_{no}. Therefore $\Sigma_{k+1}^{\mathsf{P}} = \mathsf{NP}^{\Pi_k^{\mathsf{P}}}$ and $\Pi_{k+1}^{\mathsf{P}} := \mathsf{coNP}^{\Sigma_k^{\mathsf{P}}}$.

Hence, we can also see that $\Sigma_k^{\mathsf{P}} = \mathsf{co}\Pi_k^{\mathsf{P}}$.

Question:
How do these relate to our earlier definitions of the PH classes?

Oracle TMs vs. ATMs

It turns out that this new definition leads to a familiar class of problems:[1]

Theorem 17.11: For all $k \geq 1$, we have $\Sigma_k^{\mathsf{P}} = \Sigma_k\mathsf{P}$ and $\Pi_k^{\mathsf{P}} = \Pi_k\mathsf{P}$.

Proof: We only prove the case $\Sigma_k^{\mathsf{P}} = \Sigma_k\mathsf{P}$ – the other follows by complementation. The proof is by induction on k.

Base case: $k = 1$.
The claim follows since $\Sigma_1^{\mathsf{P}} = \mathsf{NP}^{\mathsf{P}} = \mathsf{NP}$ and $\Sigma_1\mathsf{P} = \mathsf{NP}$ (as noted before).

[1] Because of this result, both of our notations are used interchangeably in the literature, independently of the definition used.

Oracle TMs vs. ATMs (2)

Induction step: assume the claim holds for k. We show $\Sigma_{k+1}^{\mathsf{P}} = \Sigma_{k+1}\mathsf{P}$.

"$\supseteq$" Assume $\mathbf{L} \in \Sigma_{k+1}\mathsf{P}$.

Oracle TMs vs. ATMs (2)

Induction step: assume the claim holds for k. We show $\Sigma^{\mathsf{P}}_{k+1} = \Sigma_{k+1}\mathsf{P}$.

"$\supseteq$" Assume $\mathbf{L} \in \Sigma_{k+1}\mathsf{P}$.

- By Theorem 17.9, for some language $\mathbf{V} \in \mathsf{P}$ and polynomial p:
  $\mathbf{L} = \{w \mid \exists^p c_1.\forall^p c_2 \ldots \mathsf{Q}^p_{k+1} c_{k+1}$ such that $(w\#c_1\#c_2\# \ldots \#c_{k+1}) \in \mathbf{V}\}$

Oracle TMs vs. ATMs (2)

Induction step: assume the claim holds for k. We show $\Sigma^{\mathsf{P}}_{k+1} = \Sigma_{k+1}\mathsf{P}$.

"$\supseteq$" Assume $\mathbf{L} \in \Sigma_{k+1}\mathsf{P}$.

- By Theorem 17.9, for some language $\mathbf{V} \in \mathsf{P}$ and polynomial p:
  $\mathbf{L} = \{w \mid \exists^p c_1.\forall^p c_2 \ldots Q^p_{k+1} c_{k+1}$ such that $(w\#c_1\#c_2\#\ldots\#c_{k+1}) \in \mathbf{V}\}$
- By Theorem 17.9, the following defines a language in $\Pi_k\mathsf{P}$:
  $\mathbf{L}' := \{(w\#c_1) \mid \forall^p c_2 \ldots Q^p_k c_{k+1}$ such that $(w\#c_1\#c_2\#\ldots\#c_{k+1}) \in \mathbf{V}\}$.

Oracle TMs vs. ATMs (2)

Induction step: assume the claim holds for k. We show $\Sigma^{\mathsf{P}}_{k+1} = \Sigma_{k+1}\mathsf{P}$.

"$\supseteq$" Assume $\mathbf{L} \in \Sigma_{k+1}\mathsf{P}$.

- By Theorem 17.9, for some language $\mathbf{V} \in \mathsf{P}$ and polynomial p:
  $\mathbf{L} = \{w \mid \exists^p c_1.\forall^p c_2 \ldots Q^p_{k+1} c_{k+1}$ such that $(w\#c_1\#c_2\# \ldots \#c_{k+1}) \in \mathbf{V}\}$
- By Theorem 17.9, the following defines a language in $\Pi_k\mathsf{P}$:
  $\mathbf{L}' := \{(w\#c_1) \mid \forall^p c_2 \ldots Q^p_k c_{k+1}$ such that $(w\#c_1\#c_2\# \ldots \#c_{k+1}) \in \mathbf{V}\}$.
- The following algorithm in $\mathsf{NP}^{\mathbf{L}'}$ decides $\mathbf{L}$:
  on input w, non-deterministically guess c_1;
  then check $(w\#c_1) \in \mathbf{L}'$ using the $\mathbf{L}'$ oracle

Oracle TMs vs. ATMs (2)

Induction step: assume the claim holds for k. We show $\Sigma^{\mathsf{P}}_{k+1} = \Sigma_{k+1}\mathsf{P}$.

"$\supseteq$" Assume $\mathbf{L} \in \Sigma_{k+1}\mathsf{P}$.

- By Theorem 17.9, for some language $\mathbf{V} \in \mathsf{P}$ and polynomial p:
  $\mathbf{L} = \{w \mid \exists^p c_1.\forall^p c_2 \ldots \mathsf{Q}^p_{k+1} c_{k+1}$ such that $(w\#c_1\#c_2\#\ldots\#c_{k+1}) \in \mathbf{V}\}$
- By Theorem 17.9, the following defines a language in $\Pi_k\mathsf{P}$:
  $\mathbf{L}' := \{(w\#c_1) \mid \forall^p c_2 \ldots \mathsf{Q}^p_k c_{k+1}$ such that $(w\#c_1\#c_2\#\ldots\#c_{k+1}) \in \mathbf{V}\}$.
- The following algorithm in $\mathsf{NP}^{\mathbf{L}'}$ decides $\mathbf{L}$:
  on input w, non-deterministically guess c_1;
  then check $(w\#c_1) \in \mathbf{L}'$ using the $\mathbf{L}'$ oracle
- By induction, $\mathbf{L}' \in \Pi^{\mathsf{P}}_k$. Hence, the algorithm runs in $\mathsf{NP}^{\Pi^{\mathsf{P}}_k} = \mathsf{NP}^{\Sigma^{\mathsf{P}}_k} = \Sigma^{\mathsf{P}}_{k+1}$

Oracle TMs vs. ATMs (3)

Induction step: assume the claim holds for k. We show $\Sigma^{\mathsf{P}}_{k+1} = \Sigma_{k+1}\mathsf{P}$.

"$\subseteq$" Assume $\mathbf{L} \in \Sigma^{\mathsf{P}}_{k+1}$.

Oracle TMs vs. ATMs (3)

Induction step: assume the claim holds for k. We show $\Sigma^{\mathsf{P}}_{k+1} = \Sigma_{k+1}\mathsf{P}$.

"$\subseteq$" Assume $\mathbf{L} \in \Sigma^{\mathsf{P}}_{k+1}$.

- There is an $\Sigma^{\mathsf{P}}_{k+1}$-TM $\mathcal{M}$ that accepts $\mathbf{L}$, using an oracle $\mathbf{O} \in \Sigma^{\mathsf{P}}_{k}$.

Oracle TMs vs. ATMs (3)

Induction step: assume the claim holds for k. We show $\Sigma_{k+1}^{\mathsf{P}} = \Sigma_{k+1}\mathsf{P}$.

"$\subseteq$" Assume $\mathbf{L} \in \Sigma_{k+1}^{\mathsf{P}}$.

- There is an $\Sigma_{k+1}^{\mathsf{P}}$-TM $\mathcal{M}$ that accepts $\mathbf{L}$, using an oracle $\mathbf{O} \in \Sigma_{k}^{\mathsf{P}}$.
- By induction, $\mathbf{O} \in \Sigma_k\mathsf{P}$ and thus $\bar{\mathbf{O}} \in \Pi_k\mathsf{P}$ for its complement

Oracle TMs vs. ATMs (3)

Induction step: assume the claim holds for k. We show $\Sigma_{k+1}^{\mathsf{P}} = \Sigma_{k+1}\mathsf{P}$.

"$\subseteq$" Assume $\mathbf{L} \in \Sigma_{k+1}^{\mathsf{P}}$.

- There is an $\Sigma_{k+1}^{\mathsf{P}}$-TM $\mathcal{M}$ that accepts $\mathbf{L}$, using an oracle $\mathbf{O} \in \Sigma_k^{\mathsf{P}}$.
- By induction, $\mathbf{O} \in \Sigma_k\mathsf{P}$ and thus $\bar{\mathbf{O}} \in \Pi_k\mathsf{P}$ for its complement
- For an $\Sigma_{k+1}\mathsf{P}$ algorithm, first guess (and verify) an accepting path of $\mathcal{M}$ including results of all oracle queries.

Oracle TMs vs. ATMs (3)

Induction step: assume the claim holds for k. We show $\Sigma^{\mathsf{P}}_{k+1} = \Sigma_{k+1}\mathsf{P}$.

"$\subseteq$" Assume $\mathbf{L} \in \Sigma^{\mathsf{P}}_{k+1}$.

- There is an $\Sigma^{\mathsf{P}}_{k+1}$-TM $\mathcal{M}$ that accepts $\mathbf{L}$, using an oracle $\mathbf{O} \in \Sigma^{\mathsf{P}}_{k}$.
- By induction, $\mathbf{O} \in \Sigma_k\mathsf{P}$ and thus $\bar{\mathbf{O}} \in \Pi_k\mathsf{P}$ for its complement
- For an $\Sigma_{k+1}\mathsf{P}$ algorithm, first guess (and verify) an accepting path of $\mathcal{M}$ including results of all oracle queries.
- Then universally branch to verify all guessed oracle queries:

Oracle TMs vs. ATMs (3)

Induction step: assume the claim holds for k. We show $\Sigma^{\mathsf{P}}_{k+1} = \Sigma_{k+1}\mathsf{P}$.

"$\subseteq$" Assume $\mathbf{L} \in \Sigma^{\mathsf{P}}_{k+1}$.

- There is an $\Sigma^{\mathsf{P}}_{k+1}$-TM $\mathcal{M}$ that accepts $\mathbf{L}$, using an oracle $\mathbf{O} \in \Sigma^{\mathsf{P}}_{k}$.
- By induction, $\mathbf{O} \in \Sigma_k\mathsf{P}$ and thus $\bar{\mathbf{O}} \in \Pi_k\mathsf{P}$ for its complement
- For an $\Sigma_{k+1}\mathsf{P}$ algorithm, first guess (and verify) an accepting path of $\mathcal{M}$ including results of all oracle queries.
- Then universally branch to verify all guessed oracle queries:
  - For queries $w \in \mathbf{O}$ with guessed answer "no", use $\Pi_k\mathsf{P}$ check for $w \in \bar{\mathbf{O}}$

Oracle TMs vs. ATMs (3)

Induction step: assume the claim holds for k. We show $\Sigma^{\mathsf{P}}_{k+1} = \Sigma_{k+1}\mathsf{P}$.

"$\subseteq$" Assume $\mathbf{L} \in \Sigma^{\mathsf{P}}_{k+1}$.

- There is an $\Sigma^{\mathsf{P}}_{k+1}$-TM $\mathcal{M}$ that accepts $\mathbf{L}$, using an oracle $\mathbf{O} \in \Sigma^{\mathsf{P}}_{k}$.
- By induction, $\mathbf{O} \in \Sigma_k\mathsf{P}$ and thus $\bar{\mathbf{O}} \in \Pi_k\mathsf{P}$ for its complement
- For an $\Sigma_{k+1}\mathsf{P}$ algorithm, first guess (and verify) an accepting path of $\mathcal{M}$ including results of all oracle queries.
- Then universally branch to verify all guessed oracle queries:
  - For queries $w \in \mathbf{O}$ with guessed answer "no", use $\Pi_k\mathsf{P}$ check for $w \in \bar{\mathbf{O}}$
  - For queries $w \in \mathbf{O}$ with guessed answer "yes", use $\Pi_{k-1}\mathsf{P}$ check for $(w\#c_1) \in \mathbf{O}'$, where $\mathbf{O}'$ is constructed as in the $\supseteq$-case, and c_1 is guessed in the first $\exists$-phase

$\square$

Summary and Outlook

The Polynomial Hierarchy is a hierarchy of complexity classes between P and PSpace

It can be defined by stacking NP-oracles on top of P/NP/coNP, or, equivalently, by bounding alternation in polytime ATMs

The typical complete problems for the classes in the polynomial hierarchy are QBF with bounded forms of quantifier alternation

What's next?

- Some more about the polynomial hierarchy
- End-of-year consultation
- Computing with circuits