Computational Complexity

Lecture 9
More of the Polynomial Hierarchy
Alternation
PH is in terms of verification
PH is in terms of verification

Recall $\Sigma_k^p$
PH is in terms of verification

Recall $\Sigma^P_k$

Languages $L = \{x| \exists w_1 \forall w_2 ... Q w_k F(x;w_1,w_2,...,w_k)\}$, where $F$ in $P$
PH is in terms of verification

Recall $\Sigma_k^P$

Languages $L = \{x| \exists w_1 \forall w_2 \ldots \forall w_k \; F(x;w_1,w_2,\ldots,w_k)\}$, where $F$ in $P$

Consider deterministic polynomial time machine $M$ for $F$, with $k$ read-once tapes for the certificates
**PH is in terms of verification**

- Recall $\Sigma_k^P$

- Languages $L = \{x| \exists w_1 \forall w_2 \ldots \forall w_k \ F(x;w_1,w_2,\ldots,w_k)\}$, where $F$ in $P$

- Consider deterministic polynomial time machine $M$ for $F$, with $k$ read-once tapes for the certificates

- Tapes read one after the other
PH is in terms of verification

- Recall $\Sigma_k^P$

- Languages $L = \{x| \exists w_1 \forall w_2 ... Q w_k F(x; w_1, w_2, ..., w_k)\}$, where $F$ in $P$

- Consider deterministic polynomial time machine $M$ for $F$, with $k$ read-once tapes for the certificates

- Tapes read one after the other

- $x$ in $L$ if $\exists w_1 \forall w_2 ... Q w_k$ such that $M(x; w_1, w_2, ..., w_k)$ accepts
PH is in terms of verification

Recall $\Sigma_k^P$

Languages $L = \{x | \exists w_1 \forall w_2 ... Q w_k F(x;w_1,w_2,..,w_k)\}$, where $F$ in $P$

Consider deterministic polynomial time machine $M$ for $F$, with $k$ read-once tapes for the certificates

Tapes read one after the other

$x$ in $L$ if $\exists w_1 \forall w_2 ... Q w_k$ such that $M(x;w_1,w_2,..,w_k)$ accepts

Plan: Formulate in terms of a non-deterministic TM (with no certificates)
Verification → Non-determinism
Verification → Non-determinism
Verification → Non-determinism
Verification → Non-determinism

Read from Tape 1
Verification →
Non-determinism

Read from Tape 1

Read from Tape 1
Verification → Non-determinism

Read from Tape 1
Read from Tape 1
Read from Tape 1
Verification → Non-determinism

Read from Tape 1

Read from Tape 1

Read from Tape 2
Verification → Non-determinism
Verification → Non-determinism

Read from Tape 1

Read from Tape 1

Read from Tape 2

Guess 0
Verification → Non-determinism

- Read from Tape 1
- Read from Tape 1
- Read from Tape 2

- Guess 0
- Guess 1
Verification → Non-determinism

- Read from Tape 1
- Read from Tape 1
- Read from Tape 2

[Diagram showing a decision tree with guesses 0 and 1.]
Verification →
Non-determinism

- Read from Tape 1
- Read from Tape 1
- Read from Tape 2

- Guess 0
- Guess 1
- Guess 0
- Guess 1
Verification → Non-determinism

Read from Tape 1

Read from Tape 1

Read from Tape 2

Guess 0

Guess 1

Guess 0

Guess 1

Guess 0

Guess 1

Guess 0
Verification → Non-determinism

Read from Tape 1

Guess 0
Guess 1

Guess 0
Guess 1

Guess 0
Guess 1

Guess 0
Guess 1

Guess 0
Guess 1

Guess 0
Guess 1

Guess 0
Guess 1

Guess 0
Guess 1
Verification → Non-determinism

- Read from Tape 1
- Read from Tape 1
- Read from Tape 2

- Guess 0
- Guess 1
- Guess 0
- Guess 1
- Guess 0
- Guess 1
Verification → Non-determinism

- Read from Tape 1
- Read from Tape 1
- Read from Tape 2

- Guess 0
- Guess 1
- Guess 0
- Guess 1
Verification →
Non-determinism

$\exists w_1$

$\forall w_2$

Guess 0
Guess 1

Guess 0
Guess 1

Guess 0
Guess 1
Verification $\rightarrow$ Non-determinism

Read from Tape 1

$\exists w_1$

Read from Tape 1

$\forall w_2$

Read from Tape 2
Verification $\rightarrow$ Non-determinism

Read from Tape 1

$\exists w_1$

Read from Tape 1

$\forall w_2$

Read from Tape 2
ATM

Alternating Turing Machine

Guess 0

Guess 1

Guess 0

Guess 1
Alternating Turing Machine

At each step, execution can fork into two
ATM

- Alternating Turing Machine
  - At each step, execution can fork into two
  - Exactly like an NTM or co-NTM
ATM

- Alternating Turing Machine
  - At each step, execution can fork into two
  - Exactly like an NTM or co-NTM
  - Accepting rule is more complex
**ATM**

- Alternating Turing Machine
  - At each step, execution can fork into two
  - Exactly like an NTM or co-NTM
  - Accepting rule is more complex
  - Like in the game tree for QBF
Two kinds of configurations: ∃ and ∀
Two kinds of configurations: $\exists$ and $\forall$

Depending on the state
ATM

Two kinds of configurations: $\exists$ and $\forall$

- Depending on the state
- A $\exists$ configuration is accepting if either child is accepting
ATM

Two kinds of configurations: ∃ and ∀

- Depending on the state
- A ∃ configuration is accepting if either child is accepting
- A ∀ configuration is accepting only if both children are accepting
Verification → Non-determinism

Read from Tape 1

$\exists w_1$

Read from Tape 1

$\forall w_2$

Read from Tape 2

Guess 0  Guess 1

Guess 0  Guess 1

Guess 0  Guess 1

Guess 0  Guess 1

Guess 0  Guess 1
Verification $\rightarrow$ Non-determinism

Given a verifier for $L$ using $k$ certificate tapes, can build an ATM for $L$ with at most $k$ alternations.
Given a verifier for $L$ using $k$ certificate tapes, can build an ATM for $L$ with at most $k$ alternations.

Non-deterministically guesses tape contents and runs verifier.
Verification ← Non-determinism

Read from Tape 1

\( \exists w_1 \)

Read from Tape 1

\( \forall w_2 \)

Read from Tape 2
Given ATM for $L$ with at most $k$ alternations, can build a verifier (using $k$ certificate tapes)
Given ATM for $L$ with at most $k$ alternations, can build a verifier (using $k$ certificate tapes).

Same time/space requirements (in terms of $|x|$)
Verification ←

Non-determinism

Given ATM for L with at most k alternations, can build a verifier (using k certificate tapes)

- Same time/space requirements (in terms of |x|)

|w_i| = #choices
Time, Space, Alternations
Time, Space, Alternations

 Complexity measures
Time, Space, Alternations

- Complexity measures
  - Time: Maximum number of steps in any thread
Time, Space, Alternations

Complexity measures

- Time: Maximum number of steps in any thread
- Space: Maximum space in any configuration reached
Time, Space, Alternations

- Complexity measures
  - Time: Maximum number of steps in any thread
  - Space: Maximum space in any configuration reached
  - Alternations: Maximum number of quantifier switches in any thread
ATIME

Σ_k TIME, Π_k TIME
**ATIME**

\[ \Sigma_k \text{TIME}, \Pi_k \text{TIME} \]

\[ \Sigma_k \text{TIME}(T) : \text{languages decided by ATMs with at most } k \text{ alternations starting with } \exists, \text{ in time } T(n) \]
ATIME

$\Sigma_k \text{TIME, } \Pi_k \text{TIME}$

$\Sigma_k \text{TIME}(T)$: languages decided by ATMs with at most $k$ alternations starting with $\exists$, in time $T(n)$

$\Sigma_k \text{TIME}(\text{poly}) = \Sigma_k^p$
ATIME

$\Sigma_k \text{TIME, } \Pi_k \text{TIME}$

$\Sigma_k \text{TIME}(T)$: languages decided by ATMs with at most $k$ alternations starting with $\exists$, in time $T(n)$

$\Sigma_k \text{TIME}(\text{poly}) = \Sigma_k^p$

Latter being exactly the certificate version
ATIME

- $\Sigma_k \text{TIME}$, $\Pi_k \text{TIME}$

  - $\Sigma_k \text{TIME}(T)$: languages decided by ATMs with at most $k$ alternations starting with $\exists$, in time $T(n)$

  - $\Sigma_k \text{TIME}(\text{poly}) = \Sigma_k^p$

    - Latter being exactly the certificate version

- ATIME
\textbf{ATIME}

- $\Sigma_k\text{TIME}, \Pi_k\text{TIME}$
  - $\Sigma_k\text{TIME}(T)$: languages decided by ATMs with at most $k$ alternations starting with $\exists$, in time $T(n)$
  - $\Sigma_k\text{TIME}(\text{poly}) = \Sigma_k^p$
    - Latter being exactly the certificate version

- \textbf{ATIME}
  - $\text{ATIME}(T)$: languages decided by ATMs in time $T(n)$
ATIME vs. DSPACE
ATIME vs. DSPACE

\[ \text{ATIME}(T) \subseteq \text{DSpace}(T^2) \]
ATIME vs. DSPACE

\[ \text{ATIME}(T) \subseteq \text{DSPACE}(T^2) \]

\[ \text{c.f. NTIME}(T) \subseteq \text{DSPACE}(T) \]
ATIME vs. DSPACE

\[ \text{ATIME}(T) \subseteq \text{DSPACE}(T^2) \]

c.f. \( \text{NTIME}(T) \subseteq \text{DSPACE}(T) \)

\[ \text{AP} \subseteq \text{PSPACE} \]
ATIME vs. DSPACE

- \( \text{ATIME}(T) \subseteq \text{DSPACE}(T^2) \)
- c.f. \( \text{NTIME}(T) \subseteq \text{DSPACE}(T) \)
- \( \text{AP} \subseteq \text{PSPACE} \)
- But \( \text{PSPACE} \subseteq \text{AP} \)
ATIME vs. DSPACE

- $\text{ATIME}(T) \subseteq \text{DSPACE}(T^2)$

- c.f. $\text{NTIME}(T) \subseteq \text{DSPACE}(T)$

- $\text{AP} \subseteq \text{PSPACE}$

- But $\text{PSPACE} \subseteq \text{AP}$

- TQBF in $\text{AP}$ (why?)
ATIME vs. DSPACE

- $\text{ATIME}(T) \subseteq \text{DSPACE}(T^2)$
- c.f. $\text{NTIME}(T) \subseteq \text{DSPACE}(T)$
- $\text{AP} \subseteq \text{PSPACE}$
- But $\text{PSPACE} \subseteq \text{AP}$
- $\text{TQBF in AP (why?)}$
- $\text{AP} = \text{PSPACE}$
\text{ATIME}(T) \subseteq \text{DSPACE}(T^2)
\textbf{ATIME(T)} \subseteq \textbf{DSPACE(T^2)}

- Evaluate if the start configuration is accepting, recursively
\textbf{ATIME}(T) \subseteq \textbf{DSPACE}(T^2)

- Evaluate if the start configuration is accepting, recursively
- A \exists configuration is accepting if any child is, and
  a \forall configuration is accepting if all children are
$\text{ATIME}(T) \subseteq \text{DSPACE}(T^2)$

- Evaluate if the start configuration is accepting, recursively
  - A $\exists$ configuration is accepting if any child is, and
    - a $\forall$ configuration is accepting if all children are
  - Space needed: depth $\times$ size of configuration
ATIME(T) ⊆ DSPACE(T^2)

- Evaluate if the start configuration is accepting, recursively
  - A ∃ configuration is accepting if any child is, and
  - A ∀ configuration is accepting if all children are

- Space needed: depth x size of configuration

- Depth = # alternations = O(T). Also, size of configuration = O(T) as any thread runs for time O(T)
$\text{ATIME}(T) \subseteq \text{DSPACE}(T^2)$

- Evaluate if the start configuration is accepting, recursively
- A $\exists$ configuration is accepting if any child is, and
  a $\forall$ configuration is accepting if all children are
- Space needed: depth $\times$ size of configuration
- Depth = number of alternations = $O(T)$. Also, size of configuration = $O(T)$ as any thread runs for time $O(T)$
- $O(T^2)$
ASPACE vs. DTIME
ASPACE vs. DTIME

\[ \text{ASPACE}(S) = \text{DTIME}(2^{O(S)}) \]
ASPACE vs. DTIME

\[ \text{ASPACE}(S) = \text{DTIME}(2^{O(S)}) \]

Recall, already seen \( \text{NSPACE}(S) \subseteq \text{DTIME}(2^{O(S)}) \)
ASPACE vs. DTIME

\[ \text{ASPACE}(S) = \text{DTIME}(2^{O(S)}) \]

Recall, already seen \( \text{NSPACE}(S) \subseteq \text{DTIME}(2^{O(S)}) \)

Poly-time connectivity in configuration graph of size at most \( 2^{O(S)} \)
ASPACE vs. DTIME

- \( \text{ASPACE}(S) = \text{DTIME}(2^{O(S)}) \)

- Recall, already seen \( \text{NSPACE}(S) \subseteq \text{DTIME}(2^{O(S)}) \)

- Poly-time connectivity in configuration graph of size at most \( 2^{O(S)} \)

- Instead of connectivity, can recursively label all accepting nodes (2 lookups per node: in \( \text{poly}(S) \) time). So \( \text{ASPACE}(S) \subseteq \text{DTIME}(2^{O(S)}) \)
ASPACE vs. DTIME

\[ \text{ASPACE}(S) = \text{DTIME}(2^{O(S)}) \]

- Recall, already seen \( \text{NSPACE}(S) \subseteq \text{DTIME}(2^{O(S)}) \)

- Poly-time connectivity in configuration graph of size at most \( 2^{O(S)} \)

- Instead of connectivity, can recursively label all accepting nodes (2 lookups per node: in \( \text{poly}(S) \) time). So \( \text{ASPACE}(S) \subseteq \text{DTIME}(2^{O(S)}) \)

- To show \( \text{DTIME}(2^{O(S)}) \subseteq \text{ASPACE}(S) \)
$\text{DTIME}(2^{O(S)}) \subseteq \text{ASPACE}(S)$
\[ \text{DTIME}(2^{O(S)}) \subseteq \text{ASPACE}(S) \]

To decide, is configuration after \( t \) steps accepting
DTIME($2^{O(S)}$) ⊆ ASPACE(S)

- To decide, is configuration after $t$ steps accepting

- Accept configuration, with unique first cell $\alpha$
  (blank tape cell and unique accept state)
DTIME($2^{O(S)}$) ⊆ ASPACE(S)

- To decide, is configuration after $t$ steps accepting
- Accept configuration, with unique first cell $\alpha$
  (blank tape cell and unique accept state)
- Once there, stays there
DTIME($2^{O(S)}$) $\subseteq$ ASPACE(S)

- To decide, is configuration after $t$ steps accepting
  - Accept configuration, with unique first cell $\alpha$ (blank tape cell and unique accept state)
  - Once there, stays there
  - Is first cell of config after $t$ steps $\alpha$
DTIME($2^{O(S)}$) $\subseteq$ ASPACE(S)

- To decide, is configuration after $t$ steps accepting
  - Accept configuration, with unique first cell $\alpha$
    (blank tape cell and unique accept state)
  - Once there, stays there
  - Is first cell of config after $t$ steps $\alpha$
  - $C(i,j,x)$ : if after $i$ steps, $j^{th}$ cell of config is $x$
$\text{DTIME}(2^{O(S)}) \subseteq \text{ASPACE}(S)$

- To decide, is configuration after $t$ steps accepting

- Accept configuration, with unique first cell $\alpha$
  (blank tape cell and unique accept state)

- Once there, stays there

- Is first cell of config after $t$ steps $\alpha$

- $C(i,j,x) : \text{if after } i \text{ steps, } j^{\text{th}} \text{ cell of config is } x$

- Need to check $C(t,1,\alpha)$
ATM for TM simulation
ATM for TM simulation

$C(i,j,x) : \text{if after } i \text{ steps, } j^{\text{th}} \text{ cell of config is } x$
ATM for TM simulation

\[ C(i,j,x) : \text{if after i steps, } j^{\text{th}} \text{ cell of config is } x \]

- Recall reduction in Cook’s theorem
ATM for TM simulation

- $C(i,j,x)$: if after $i$ steps, $j^{th}$ cell of config is $x$

- Recall reduction in Cook’s theorem

- If $C(i-1,j-1,a)$, $C(i-1,j,b)$, $C(i-1,j+1,c)$ then $C(i,j,x)$ iff $x=F(a,b,c)$
ATM for TM simulation

\( C(i,j,x) : \) if after \( i \) steps, \( j^{th} \) cell of config is \( x \)

- Recall reduction in Cook’s theorem
  - If \( C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c) \) then \( C(i,j,x) \) iff \( x = F(a,b,c) \)

\( C(i,j,x) : \exists a,b,c \) st \( x = F(a,b,c) \) and \( C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c) \)
ATM for TM simulation

\( C(i,j,x) \): if after \( i \) steps, \( j^{th} \) cell of config is \( x \)

\( C(i,j,x) \): if after \( i \) steps, \( j^{th} \) cell of config is \( x \)

Recall reduction in Cook’s theorem

If \( C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c) \) then \( C(i,j,x) \) iff \( x=F(a,b,c) \)

\( C(i,j,x) \): \( \exists a,b,c \) st \( x=F(a,b,c) \) and \( C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c) \)

Base case: \( C(0,j,x) \) easy to check from input
**ATM for TM simulation**

- $C(i,j,x)$: if after $i$ steps, $j^{th}$ cell of config is $x$

- Recall reduction in Cook’s theorem

- If $C(i-1,j-1,a)$, $C(i-1,j,b)$, $C(i-1,j+1,c)$ then $C(i,j,x)$ iff $x=F(a,b,c)$

- $C(i,j,x)$: $\exists a,b,c$ st $x=F(a,b,c)$ and $C(i-1,j-1,a)$, $C(i-1,j,b)$, $C(i-1,j+1,c)$

- Base case: $C(0,j,x)$ easy to check from input

- Naive recursion: Extra $O(S)$ space at each level for $2^{O(S)}$ levels!
ATM for TM simulation
ATM for TM simulation

ATM to check if C(i,j,x)
ATM for TM simulation

- ATM to check if $C(i,j,x)$

$C(i,j,x): \exists a,b,c \text{ s.t. } x = F(a,b,c) \text{ and } C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)$
ATM for TM simulation

- ATM to check if \( C(i,j,x) \)

- \( C(i,j,x) : \exists a,b,c \text{ st } x=F(a,b,c) \text{ and } C(i-1,j-1,a), \ C(i-1,j,b), \ C(i-1,j+1,c) \)

- **Tail-recursion** (in parallel forks)
ATM for TM simulation

- ATM to check if $C(i,j,x)$

- $C(i,j,x)$: $\exists a,b,c$ st $x=F(a,b,c)$ and $C(i-1,j-1,a)$, $C(i-1,j,b)$, $C(i-1,j+1,c)$

- **Tail-recursion** (in parallel forks)

- Check $x=F(a,b,c)$; then enter universal state, fork out for each of the three configurations to be checked
ATM for TM simulation

- ATM to check if \( C(i,j,x) \)
  
- \( C(i,j,x) : \exists a,b,c \text{ st } x = F(a,b,c) \text{ and } C(i-1,j-1,a), C(i-1,j-1,a), \)
  
  \( C(i-1,j+1,c) \)

- **Tail-recursion** (in parallel forks)

  - Check \( x = F(a,b,c) \); then enter universal state, fork out for each of the three configurations to be checked

  - Overwrite \( C(i,j,x) \) with \( C(i-1,\ldots) \) and reuse space
ATM for TM simulation

- ATM to check if $C(i,j,x)$

$C(i,j,x)$: $\exists a,b,c$ st $x = F(a,b,c)$ and $C(i-1,j-1,a)$, $C(i-1,j,b)$, $C(i-1,j+1,c)$

- **Tail-recursion** (in parallel forks)

  - Check $x = F(a,b,c)$; then enter universal state, fork out for each of the three configurations to be checked
  - Overwrite $C(i,j,x)$ with $C(i-1,\ldots)$ and reuse space
  - Stay within the same $O(S)$ space at each level!
ATM for TM simulation

- ATM to check if $C(i,j,x)$

- $C(i,j,x): \exists a,b,c \text{ st } x=F(a,b,c) \text{ and } C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)$

- Tail-recursion (in parallel forks)
  
  - Check $x=F(a,b,c)$; then enter universal state, fork out for each of the three configurations to be checked
  
  - Overwrite $C(i,j,x)$ with $C(i-1,...)$ and reuse space
  
  - Stay within the same $O(S)$ space at each level!

- Gets the AND check for free. No need to use a stack.
ASPACE vs. DTIME
ASPACE vs. DTIME

\[ \text{ASPACE}(S) = \text{DTIME}(2^{O(S)}) \]
ASPACE vs. DTIME

- \( \text{ASPACE}(S) = \text{DTIME}(2^{O(S)}) \)
- \( \text{APSPACE} = \text{EXP} \)
ASPACE vs. DTIME

- $\text{ASPACE}(S) = \text{DTIME}(2^{O(S)})$
- $\text{APSPACE} = \text{EXP}$
- $\text{AL} = \text{P}$
Zoo
DTISP(T,S)
DTISP(T,S)

Theorem: \( \text{NTIME}(n) \nsubseteq \text{DTISP}(n^{1+\varepsilon}, n^\delta) \) for some \( \varepsilon, \delta > 0 \)
**DTISP(T,S)**

- **Theorem:** $\text{NTIME}(n) \not\subseteq \text{DTISP}(n^{1+\epsilon},n^{\delta})$ for some $\epsilon, \delta > 0$

  i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space
**DTISP(T, S)**

**Theorem:** $\text{NTIME}(n) \not\subset \text{DTISP}(n^{1+\epsilon}, n^\delta)$ for some $\epsilon, \delta > 0$

i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space

**Commonly Believed:** can’t solve in less than exponential time or with less than linear space
DTISP(T, S)

Theorem: \( \text{NTIME}(n) \not\subset \text{DTISP}(n^{1+\epsilon}, n^\delta) \) for some \( \epsilon, \delta > 0 \)

i.e., cannot solve SAT in some slightly super-linear time \textbf{and}
slightly super-logarithmic space

\( \text{Commonly Believed: can’t solve in less than exponential time or} \)
with less than linear space

Follows (after careful choice of parameters) from
Theorem: $\text{NTIME}(n) \not\subset \text{DTISP}(n^{1+\epsilon}, n^{\delta})$ for some $\epsilon, \delta > 0$

i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space

Commonly Believed: can’t solve in less than exponential time or with less than linear space

Follows (after careful choice of parameters) from

$\text{DTISP}(T,S) \subseteq \Sigma_2 \text{TIME}(T^{1/2} S)$
Theorem: $\text{NTIME}(n) \not\subset \text{DTISP}(n^{1+\epsilon}, n^\delta)$ for some $\epsilon, \delta > 0$

i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space

Commonly Believed: can’t solve in less than exponential time or with less than linear space

Follows (after careful choice of parameters) from

$\text{DTISP}(T,S) \subseteq \Sigma_2 \text{TIME}(T^{1/2} S)$
**DTISP**(T,S)

- **Theorem:** NTIME(n) ∉ DTISP(n^{1+ε},n^δ) for some ε, δ > 0

- i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space

- Commonly Believed: can’t solve in less than exponential time or with less than linear space

- Follows (after careful choice of parameters) from

  - DTISP(T,S) ⊆ Σ₂TIME(T^{1/2} S)

  - NTIME(n) ⊆ DTIME(n^{1+ε}) ⇒ Σ₂TIME(T) ⊆ NTIME(T^{1+ε})
**DTISP(T,S)**

- **Theorem:** $\text{NTIME}(n) \not\subset \text{DTISP}(n^{1+\epsilon}, n^\delta)$ for some $\epsilon, \delta > 0$

  - i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space
    - Commonly Believed: can't solve in less than exponential time or with less than linear space

  - Follows (after careful choice of parameters) from
    - $\text{DTISP}(T,S) \subseteq \Sigma_2 \text{TIME}(T^{1/2} S)$
    - $\text{NTIME}(n) \subseteq \text{DTIME}(n^{1+\epsilon}) \Rightarrow \Sigma_2 \text{TIME}(T) \subseteq \text{NTIME}(T^{1+\epsilon})$
    - $\text{NTIME}(n) \subseteq \text{DTISP}(n^{1+\epsilon}, n^\delta) \Rightarrow \text{NTIME}(n^\dagger) \subseteq \text{NTIME}(n^{\dagger(1/2+\epsilon')})$
Today
Today

- ATM to define levels of PH
Today

- ATM to define levels of PH
- ATIME and ASPACE
Today

- ATM to define levels of PH
- ATIME and ASPACE
  - $AP = PSPACE$ and $APSPACE = EXP$
Today

- ATM to define levels of PH
- ATIME and ASPACE
  - AP = PSPACE and APSPACE = EXP
- Using $\Sigma_2$TIME for a DTISP lower-bound