There are many classes of problems that are more complex than problems in \text{NP} or \text{coNP}, but not "arbitrarily" complex.

- Problems related to regular expressions/languages
  
  E.g., containment between the languages denoted by two regular expressions

  - Universality of the language denoted by a regular expression / finite state automaton

- Games in which players alternate moves on a board

  (Note: we have to consider an \text{m} \times \text{n} board, where \text{n} determines the size of the input)

  - Existence of a winning strategy for one of the players; i.e.

    does there exist a move of \text{P}_1 \text{ s.t.}

    for all moves of \text{P}_2

    there exists a move of \text{P}_1 \text{ s.t.}

    \text{P}_1 \text{ wins}

- Problems related to special kinds of logics (that are more expressive than propositional logic, but less expressive than first-order logic)

  - Model logics
  - Temporal logics (\text{LTL}, \text{CTL}, \text{PDL}, \text{\mu-calculus})
  - Description logics, UML class diagrams, ER diagrams

We want to characterize the computational complexity of such problems.
A first step is to relax to oracle TMs (OTMs).

We define OTMs informally:

- let $\mathcal{g}$ be a function $\Sigma^* \rightarrow \Sigma^*$ (which we use as an oracle)

- an OTM $M_g$ that uses oracle $\mathcal{g}$ in a TM with two tapes, and a special oracle state $\sigma$:
  - an ordinary tape
  - an oracle tape on which the TM can read and write normally, but also consult the oracle $\mathcal{g}$ at the cost of a single transition

- to consult the oracle, $M_g$:
  - writes the input string $x$ for $\mathcal{g}$ on the oracle tape
  - enters the oracle state $\sigma$
  - this activates the oracle, which replaces $x$ with $\mathcal{g}(x)$ on the oracle tape and places the head at the beginning of $\mathcal{g}(x)$ (all in one step)
  - after consulting the oracle, $M_g$ leaves the oracle state, but can use $\mathcal{g}(x)$ on the oracle tape

- $M_g$ accepts as normal, by entering a final state

Oracle can give TMs a lot of power.

Let us consider a class $C$ of TMs computing functions.

Definition: $P^C = \{ L \mid L \text{ is accepted by a deterministic poly-time OTM with an oracle in } C \}$

$NP^C = \{ L \mid L \text{ is accepted by a non-deterministic poly-time OTM with an oracle in } C \}$
Example: Consider $\text{CoNP}$, i.e. the oracle is a polynomial-time NTM (that leaves its result on the oracle tape).

$\text{P}^{\text{NP}}$ includes both $\text{NP}$ and $\text{CoNP}$

To solve a problem in $\text{NP}$ (resp. $\text{CoNP}$) a single call to the oracle is sufficient.

We get

$\Sigma_0^T = \Pi_0^T = \Delta_0^T = \text{P}$

and for all $k > 0$:

$\Delta_k^T = \text{P}^{\Sigma_k^T}$

$\Sigma_k^T = \text{NP}^{\Sigma_{k-1}^T}$

$\Pi_k^T = \text{CoNP}^{\Sigma_k^T}$

Note: we do not know whether $\text{P}^{\text{NP}} \neq \text{NP}^{\text{NP}}$

13/12/2007

Exploiting this idea, we can define a hierarchy of classes of greater and greater apparent difficulty:

$\Sigma_1^T = \text{NP}^{\Sigma_0^T} = \text{NP}^{\text{P}} = \text{NP}$

$\Pi_1^T = \text{CoNP}^{\Sigma_0^T} = \text{CoNP}$
We define the polynomial hierarchy $PH = \bigcup_{j=1}^{\infty} \Sigma_j^P \cup \Pi_j^P \cup \Delta_j^P$.

It is not known whether the hierarchy is truly infinite, but if it collapses at one level, then it collapses also above.

**Theorem:** If for some $k \geq 1$, we have $\Sigma_k^P = \Pi_k^P$, then $\Sigma_j^P = \Pi_j^P = \Sigma_k^P$ for all $j \geq k$.

In particular, if $P = NP$, then $NP = \Sigma_2^P = \Pi_2^P$ and so $\Sigma_j^P = P$ for all $j \geq 0$, i.e. $PH = P$.

We can define completeness for the versions $\Sigma_i^P$, $\Pi_i^P$, $\Delta_i^P$, as we did for $NP$-completeness.

Are there natural problems that are complete for $\Sigma_i^P$, $\Pi_i^P$, $\Delta_i^P$?
Quantified boolean formulae (QBF)

let \( \mathbb{X} \) be a set of boolean variables partitioned into
\[
\mathbb{X} = \mathbb{X}_1 \cup \ldots \cup \mathbb{X}_n,
\]
and let \( \Phi \) be a propositional formula over \( \mathbb{X} \).

Then \( \Phi = \exists \mathbb{X}_1 \, \forall \mathbb{X}_2 \, \exists \mathbb{X}_3 \ldots \, \forall \mathbb{X}_n \, \Phi \) is a quantified boolean formula with \( n \) alternations of quantifiers (QBF\(_n\)).

\( \Phi \) is satisfiable if:
- there is an assignment to the variables in \( \mathbb{X}_i \) s.t.
  - for all
  - there is
  - there is
  - \( F \) is true

\( \text{QSAT}_i = \{ \Phi \mid \Phi \text{ is a QBF}_i \text{ and } \Phi \text{ is satisfiable} \} \)

Theorem: for all \( i \geq 1 \), \( \text{QSAT}_i \) is \( \Sigma_i \) complete.

Note: games where players alternate \( i \) moves can be encoded as a formula of QBF\(_i\).
Space and time bounded TMs

It turns out that all problems in \( \text{PH} \) can be solved by a TM that uses at most polynomial space.

\[
\text{PSPACE} = \{ L \mid \exists L \in \text{DTM} \text{ M that uses at most space that is polynomial in its input} \}
\]

Examples of PSPACE-complete problems:

- Universality of a regular expression
- Emptiness of the intersection of \( n \) DFAs (\( n \) is part of the input)
- satisfiability of quantified boolean formulas, i.e. QSAT
- board games with a polynomially bounded number of moves (existence of a winning strategy)

We said that QSAT \( \in \Sigma_1^p \), complete.

and QSAT \( \in \text{PSPACE} \), complete.

We can also define \( \text{NPSPACE} \) in an analogous way:

\[
\text{NPSPACE} = \{ L \mid L = L(N) \text{ for some } \text{NTM } N \text{ that uses at most polynomial space} \}
\]
Relationship between PSPACE, NPSPACE and P, NP

Easy facts: P \subseteq PSPACE
NP \subseteq NPSPACE

Follows from the fact that a TM that does at most \( P(n) \) steps cannot use more than \( P(n) \) tape cells.

Note: in the definition of (N)PSPACE there is nothing that requires the TM to halt, so it could in principle contain non-recursive languages.

Actually this is not the case:

**Theorem:** If \( M \) is a (N)TM with space bound \( p(n) \), and \( w \in L(M) \), the \( w \) is accepted within \( C + p(1w) \) steps, for some constant \( C \).

**Proof:** Idea: \( M \) must repeat an ID before making more than \( C + p(1w) \) moves.

If \( M \) repeats an ID and accepts, then there is a shorter sequence of IDs leading to acceptance:

\[
\begin{align*}
\alpha \rightarrow^* \beta \rightarrow^* \beta \\
\text{initial ID} & \quad \text{accepting ID} \\
\alpha \rightarrow^* \beta \rightarrow^* \gamma
\end{align*}
\]

Why must \( M \) repeat an ID?
- The number of symbols in each position is limited \( \Gamma \) and \( Q \) are finite.
- The number of positions is limited by \( p(1w) \).
Set \( t = 1^m \), \( a = 1|1, n = 1\) with only \( p(m) \) tape cells, we have at most
\[ K = \alpha \cdot p(m) \cdot t^p(m) \]
different IDs
\[ \uparrow \quad \uparrow \quad \text{symbols in the} \]
\[ \text{states positions p(m) positions} \]

Set \( c = t + a \) and consider
\[ (t + a)^{t + p(m)} = \underbrace{t + p(m) + (1 + p(m)) \cdot \alpha \cdot t^p(m)}_{c} \]

This result gives us a way to convert a poly-space TM into one that does at most an exponential number of moves.

**Theorem:** Let \( L \in \text{(N)PSPACE} \).

Then \( L \) is accepted by a poly-space bounded \((N)TM\)
that makes at most \( c^{q(m)} \) steps, for some constant \( c > 1 \)
and polynomial \( q(m) \).

**Proof:** By the previous theorem, we know that \( L = \text{L}(M) \)
for some \((N)TM\) \( M \) that runs in at most \( c^{p(1\text{w1})} \) steps.

**Idea:** We construct from \( M \) a 2-tape TM \( M_2 \) that has
a counter in base \( c \) on tape 2, and stops at least when the counter reaches \( c^{t + p(1\text{w1})} \).

**On tape 1:** \( M_2 \) simulates \( M \).

**How much tape uses \( M_2 \):**
- tape 1: at most \( p(1\text{w1}) \) cells
- tape 2: at most \( p(1\text{w1}) + p(1\text{w1}) \) cells (since the base \( c \)
counter counts at most to \( c^{t + p(1\text{w1})} \)).
We can convert \( M_2 \) to a 1-tape TM \( M_3 \)

- \( M_3 \) uses no more than \( 9 \cdot p(n) \) tape cells
  (for any input of length \( n \))

- \( M_3 \) runs in time quadratic in the running time of \( M_2 \),
  i.e., \( O(2^2 \cdot p(n)) < \cdot q(n) \) for \( q(n) = 2 \cdot p(n) + 1 \)

\[ \square \]

\[ \text{PSPACE vs. NPSPACE:} \]

- \text{Obviously: PSPACE } \subseteq \text{ NPSPACE} \]

- \text{Surprisingly: PSPACE } = \text{ NPSPACE!} \]

\text{Idea of proof:}

- simulation of \( \text{NTM } N \) with space bound \( p(n) \)
  by \( \text{DTM } D \) in \( O(p(n)^2) \)

Exploits a deterministic, recursive test for whether a
\( \text{NTM } N \) can have \( \text{ID}_i \xrightarrow{N} \text{ID}_j \)

\( \leq n \) steps

\( \text{DTM } D \) tries all intermediate \( \text{ID}_k \) reaching for \( \text{ID}_k \):

\( \text{ID}_i \xrightarrow{N} \text{ID}_k \xrightarrow{N} \text{ID}_j \)

\( \leq \frac{m}{2} \)

\( \leq \frac{m}{2} \)

We use a recursive boolean function \( \text{reach}(\text{ID}_i, \text{ID}_j, m) \):

\[ \text{reach}(\text{ID}_i, \text{ID}_j, 1) = \begin{cases} \text{true} & \text{if } \text{ID}_i = \text{ID}_j \text{ or } \text{ID}_i \xrightarrow{N} \text{ID}_j \\ \text{false} & \text{otherwise} \end{cases} \]

\[ \text{reach}(\text{ID}_i, \text{ID}_j, m) = \bigvee \left( \text{reach}(\text{ID}_i, \text{ID}_k, \frac{m}{2}) \land \text{reach}(\text{ID}_k, \text{ID}_j, \frac{m}{2}) \right) \text{ for } m > 1 \]
Note: reach cells itself several times, i.e. twice for each possible intermediate ID, but - after each ID we just need to remember a bit (and the counter used to iterate through IDs) - the two calls for the same ID can be done in sequence.

What is the depth of the recursive calls?

N does not make more than \(C^{(n)}\) moves

\[ \Rightarrow \text{we can restart with } m = C^{(n)} \]

\[ \Rightarrow \text{The recursive call depth is } \leq \log_2 C^{(n)} = O(p(m)) \]

\[ \text{(Theorem (Savitch's Theorem))} \quad \text{PSPACE} = \text{NPSPACE} \]

\[ \text{Proof: we only need to show } \text{NPSPACE} \subseteq \text{PSPACE}, \]

i.e. if \(L = L(N)\) for a NTM \(N\) with space bound \(p(m)\)

\[ L = L(D) \text{ for a DTM } D \]

(\(q(m)\) polynomials)

We can assume that \(N\) accepts \(w\) in \(\leq C^{1+p(m)}\) steps

(\(C^{1+p(m)}\) steps)

For some constant \(C\)

Given \(w\) with \(|w| = n\), \(D\) repeatedly calls reach \((ID_0, ID_{j+1})\), where:

- \(ID_0\) is the initial ID of \(N\) with input \(w\)
- \(ID_j\) is an accepting ID using at most \(p(m)\) cells

\[ m = C^{1+p(m)} \]

The depth of the recursive calls of reach is \(\leq \log_2 m\), i.e. \(O(p(m))\)
D can manage its tape as a stack:

- for each recursive call, it puts on the stack:
  - $\ldots 1 + q(n) \ldots$ cells
  - $\ldots 1 + q(n) \ldots$ cells
- an $\ldots$ in binary: $\log_2 (1 + q(n)) = O(q(n))$

$\Rightarrow$ The total length of the tape is:

$O(q(n) - (2 + 2q(n)) = O(q(n))) = O(q(n)^2)$

(Note: D also uses a scratch tape portion to enumerate through the various 1Ds.)

---

**PSPACE - completeness**

We can define PSPACE - hardness (and PSPACE - completeness) as done for NP.

**Definition:** A language $L$ is PSPACE - hard if for every language $L' \in \text{PSPACE}$ we have that $L' \leq_{\text{poly}} L$.

$L$ is PSPACE - complete if:

1) $L \in \text{PSPACE}$, and
2) $L$ is PSPACE - hard.

Note: the definition uses poly-time and not poly-space reductions, since we want to obtain similar properties as for NP-completeness.
Theorem: Let \( L \) be \( \text{PSPACE} \)-complete.

1) If \( L \in \text{P} \), then \( \text{P} = \text{PSPACE} \).
2) If \( L \notin \text{NP} \), then \( \text{NP} \neq \text{PSPACE} \).

Proof: We show (1). The proof for (2) is similar.

Consider a language \( L' \in \text{PSPACE} \). We show that \( L' \in \text{P} \).

Since \( L \) is \( \text{PSPACE} \)-complete, we have \( L' \leq_{\text{p}} L \).

Let the poly-time reduction be \( R \) and let \( R \) take time \( q(m) \).

Since \( L \in \text{P} \), it has a poly-time algorithm.

Let this poly-time algorithm run in time \( p(m) \).

Consider a string \( w \) for which we want to test whether \( w \in L' \).

Then \( w \in L' \) iff \( R(w) \in L \).

Since \( R \) takes time \( q(|w|) \), \( |R(w)| \leq q(|w|) \).

We can test whether \( R(w) \in L \) in time \( p(|R(w)|) \leq p(q(|w|)) \),

i.e., polynomial in \( |w| \).

Hence, we have a poly-time algorithm for \( L' \).

We get that \( \text{PSPACE} \subseteq \text{P} \). The other direction is obvious. \( \square \)

We show now a problem that is \( \text{PSPACE} \)-complete.

**Quantified Boolean Formulas (QBF)**

We recall the definition of QBF and slightly modify it to make it more suitable for what we want to show here.

A QBF is a boolean expression in which additionally boolean variables may be quantified.
Formally, a QBF is defined inductively as follows:
- 0 (i.e. false) and 1 (i.e. true) are QBFs
- every variable in a QBF
- if E and F are QBFs, then so are
  \[-E, \ E \land F, \ E \lor F\]
- if F is a QBF that does not include a quantification of
  variable x, then so are
  \[\forall x (E), \ \exists x (E)\]

We say that the scope of x is E.

Note: We may use parentheses to disambiguate.

For simplicity, we have chosen not to allow multiple quantifications over the same variable. This does not limit expressiveness, but is also not strictly necessary.

Example: \[\forall x (\exists y (x \land y)) \lor \forall z (x \lor z)\]

If a variable x is in the scope of \(\forall x\) or \(\exists x\), then it is said to be bound. Otherwise it is free.

The value of a QBF with no free variables is either 0 or 1.

We can compute the value \(v(F)\) of such a QBF F by induction:

- base case: \(F = 0\) or \(F = 1\)
  - then \(v(F) = F\)

Note: we cannot have \(F = x\), since x would be free.

- inductive cases:
  - \(F = \neg E\) then E is shorter than F, and we can evaluate it by induction. If \(v(E) = 1\), then \(v(F) = 0\).
  - \(v(E) = 0\) then \(v(F) = 1\).
- $F = E \land E'$ then both $E$ and $E'$ are shorter than $F$, and we can evaluate them.

  - If $\nu(E) = 1$ and $\nu(E') = 1$, then $\nu(F) = 1$
  - otherwise $\nu(F) = 0$.

- $F = E \lor E'$ similar

- $F = \forall x(E)$

  - let $E_0$ be obtained from $E$ by replacing each $x$ by 0
  - let $E_1$ similarly

  Note: $E_0$ and $E_1$ have no free variables

  - are both shorter than $F$

  Hence we can evaluate $E_0$ and $E_1$.

  - If $\nu(E_0) = 1$ and $\nu(E_1) = 1$, then $\nu(F) = 1$
  - otherwise $\nu(F) = 0$

- $F = \exists x(E)$ similar

  - If $\nu(E_0) = 0$ and $\nu(E_1) = 0$, then $\nu(F) = 0$
  - otherwise $\nu(F) = 1$.

We define the QBF problem:

$$QBF = \{ F \mid F \text{ is a QBF formula without free variables and } \nu(F) = 1 \}$$

We show now that QBF is PSPACE-complete.
Theorem: QBF ∈ PSPACE

Proof: We exploit the recursive evaluation procedure, and show that it can be implemented by a TM M that uses only polynomial space.

M keeps on its tape a stack. Each record of the stack contains a formula and an index to the subformula that M is currently working on.

Let \( F \) be the formula to evaluate, and let \(|F| = n\).

Initially, a record for \( F \) is placed on the stack.

If \( F = E \land E' \), then M proceeds as follows:

1) Place \( E \) in a record to the right of the one for \( F \).
2) Recursively evaluate \( E \).
3) If \( v(E) = 0 \),
   then return 0 as \( v(F) \)
   else replace the record of \( E \) by the one of \( E' \)
   recursively evaluate \( E' \)
   return \( v(E) \) as \( v(F) \)

If \( F = \exists x (E) \), then M proceeds as follows:

1) Create \( E_0 \) by replacing each occurrence of \( x \) in \( E \) by \( 0 \),
   and place \( E_0 \) in a record to the right of the one for \( F \).
2) Recursively evaluate \( E_0 \).
3) If \( v(E_0) = 1 \),
   then return 1 as \( v(F) \)
   else create \( E_1 \) by substituting 1 for \( x \) in \( E \)
   replace the record of \( E_0 \) by the one of \( E_1 \)
   recursively evaluate \( E_1 \)
   return \( v(E_1) \) as \( v(F) \)
The cases for $F = \neg E$, $F = \neg E \vee E'$, $F = \forall x(E)$ are similar [exercise].

When $F = 0$ or $F = 1$ then $F$ is returned immediately, without creating a further record.

Note:

1) The records to the right of the one for a formula $E$ are for formulas that are shorter than $E$.

2) When two subexpressions have to be evaluated in the cases $E \land E'$, $E \lor E'$ and $\exists x(E)$, $\forall x(E)$ the two subexpressions are evaluated in sequence, and the two records for $E$, $E'$ (resp. $E_0$, $E_1$) are never at the same time on the stack.

It follows that for $|F| = m$, there are at most $m$ records on the stack, and each record has length $O(m)$. Hence, the used tape is at most $O(m^2)$.

To show PSPACE-hardness of QBF, we encode the computation of a poly-space TM $M$ into a QBF.

Can we directly use variables $x_{j,t}$ to encode that

the symbol in position $j$ of configuration $t$ is $A$ as done for encoding the computation of a poly-time TM in SAT?

No! Since the number of steps of $M$ is exponential, and we would need exponentially many variables.

Idea: we use quantification, to let a variable represent many different configurations.
Theorem: QBF is PSPACE-hard

Consider a language $L \in \text{PSPACE}$, and let $M$ be a TM s.t. $L(M) = L$, and let $M$ use at most $p(n)$ space. Let $w$ be an input string for $M$ with $|w| = n$.

We construct from $M$ and $w$ a QBF $E$ without free variables and size polynomial in $n$ s.t. $\forall n(E)$ is true iff $w \in L(M)$.

By a previous theorem, we know that there is a constant $c$ s.t. $M$ accepts an input of length $n$ in $c^{(c+c)}$ steps.

We encode the computation using variables as follows:

- There is a constant number of configuration symbols:
  - $\Rightarrow$ We can encode each explicitly through an index in the variables

- There is a polynomial number of tape cells:
  - $\Rightarrow$ We can encode each explicitly through an index in the variables

- There are $c^{(c+c)}$ IDs
  - $\Rightarrow$ We do not encode all of them explicitly,
    - Rather we represent an IDs through variables over which we quantify. We call the set of variables $\{x\}$ representing an ID a "variable ID".

When $I$ is the variable ID represented by $x_1, \ldots, x_m$ we use EI to denote $\exists x_1 \exists x_2 \ldots \exists x_m \forall x \ldots \forall x_m$
We construct a QBF of the form:

$$3\forall, 3\exists f \ (S \land F \land N)$$

where

- $I_0$ is a variable ID representing the initial ID accepting ID
- $S$ says "starts right"
  - i.e. $I_0$ is the initial ID of $M$ with input $w$
- $F$ says "finishes right"
  - i.e. $I_f$ is an accepting ID
- $N$ says "moves right"
  - i.e. $M$ moves from $I_0$ to $I_f$

Structure of $S, F, N$.

- starts right: $S$ is the AND of literals using the variables of $I_0$
  - when the $j$-th position of the initial ID is $A$, then $y_{jA}$
  - $\overline{y_{jA}}$ is not $A$, then $\overline{y_{jA}}$
  
  $\Rightarrow |S| \text{ is linear in } p(n)$

- finishes right: $F$ is the OR of the variables $y_{jA}$ chosen from those of $I_f$ for which $A$ represents an accepting state, and $j$ is arbitrary.

- moves right: $N$ is based on the recursive splitting of the computation in halves, adding only $O(p(n))$ symbols for each split
1. For variable IDs I with variables \( y_{i,j} \), \( 1 \leq j \leq n \), we use \( I = J \) to abbreviate \( \bigwedge_{j,a} (y_{i,j} \land z_{j,a}) \lor (\overline{y}_{i,j} \land \overline{z}_{j,a}) \).

2. We construct \( N_i(I,J) \) for \( i = 1, 2, 4, 8, 16, \ldots \) to mean \( I \leq^* M \) in at most \( i \) moves.

   The only free variables of \( N_i(I,J) \) are those of \( I, J \).

   **Basis:** \( N_1(I,J) \) asserts either \( I = J \) or \( I \leq^* M \).

   To encode \( I \leq^* M \), we can proceed as for the proof of Cook's Theorem.

   **Induction:** we construct \( N_{2^k}(I,J) \) from \( N_i \).

   **Note:** we cannot use

   \[
   N_{2^i}(I,J) = \exists k (N_i(I,k) \land N_i(k,J))
   \]

   since the overall formula would become exponentially long [Verify as an exercise].

   Instead, we must use only one copy of \( N_i \) to check both \( N_i(I,k) \) and \( N_i(k,J) \), i.e.

   there exists an ID \( K \) such that for all variable IDs \( P, Q \):

   \[
   (P,Q) = (I,K) \lor (P,Q) = (K,J)
   \]

   implies \( N_i(P,Q) \) is true

   \[\Rightarrow N_{2^i}(I,J) = \exists k \forall P \forall Q \left( N_i(P,Q) \lor \neg P = I \land Q = K \right) \land \neg P = K \land Q = J)\)
Then $N = N_m(I_0, I_1)$, where $m$ is the smallest power of 2 greater or equal to $\ell^{1+r(m)}$.

The number of recursive steps to determine $N$ is

$$\log_2 (\ell^{1+r(m)}) = O(r(m))$$

Each recursive step takes time $O(r(m))$.

$\Rightarrow$ $N$ can be constructed in time $O(r(m)^2)$

One can verify that $\exists I_0, I_1, (S \land F \land N)$ has value 1

diff $\iff w \in \ell^L(M)$
Further important time and space complexity classes:

\[ \text{EXPTIME} = \{ L \mid L = \mathbb{L}(M) \text{ for some exponential DTM } M \} \]

\[ \text{EXPSPACE} = \{ L \mid L = \mathbb{S}(M) \text{ for some exponential DTM } M \} \]

\[ \text{kEXPTIME} = \{ L \mid L = \mathbb{L}(M) \text{ for some DTM } M \text{ with running time } T(n) = 2^{O(n^k)} \} \]

\[ \text{kEXPSPACE} = \{ L \mid L = \mathbb{S}(M) \text{ for some DTM } M \text{ that on input of length } n, \text{ uses space that is at most } 2^{O(n^k)} \} \]

We can define \( \text{NkEXPTIME} \)
\( \text{NkEXPSPACE} \)

as for the deterministic classes, but using NTMs instead of DTMs.

We have:

\[ \text{kEXPTIME} \cap \text{NkEXPTIME} \]

\[ H_{\text{EXPTIME}} \]

\[ \text{kEXPSPACE} = \text{NkEXPSPACE} \]

\[ \text{kEXPTIME} \subseteq \text{kEXPSPACE} \]

Natural problems in these classes are logic related.

Note: \( \text{EXPTIME} \) is the first provable intractable class, i.e., we know:

\[ P \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \]

(we don't know which of these inclusions is strict)