

The polynomial hierarchy and PSPACE

6.1

There are many classes of problems that are more complex than problems in NP or 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 $n \times n$ board, where 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 π_1 s.t.

for all moves of π_2

there exists a move of π_1 s.t.

...

π_1 wins

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

 - modal logics

 - temporal logics (LTL, CTL, PDL, μ -calculus)

 - description logics, UML class diagrams, ER diagrams

We want to characterize the computational complexity of such problems

a first step is to resort to oracle TMs. (OTMs)

We define OTMs informally:

- let g be a function $\Sigma^* \rightarrow \Sigma^*$ (which we use as an oracle)
- an OTM M_g that uses oracle g is a TM with two tapes:
 - an ordinary tape
 - an oracle tape on which the TM can read and write normally, but also consult the oracle g at the cost of a single transition
- to consult the oracle, M_g :
 - writes the input string x for g on the oracle tape
 - enters the oracle state σ
 - this activates the oracle, which replaces x with $g(x)$ on the oracle tape and places the head at the beginning of $g(x)$ (all in one step)
 - after consulting the oracle, M_g leaves the oracle state but can use $g(x)$ on the oracle tape
- M_g accepts as usual, by entering a final state

Oracles can give TMs a lot of power.

Let us consider a class $\mathcal{C}$ of TMs computing functions:

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

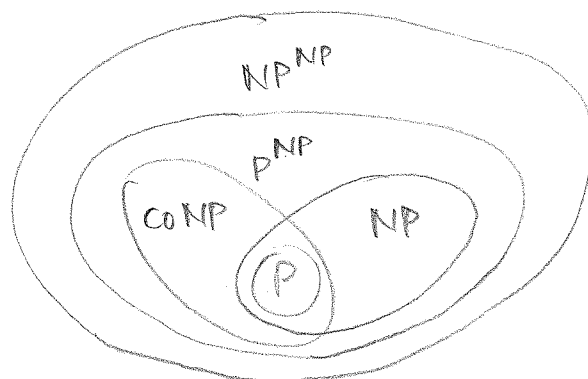
$NP^{\mathcal{C}} = \{L \mid L \text{ is accepted by a non-deterministic poly-time OTM with an oracle in } \mathcal{C}\}$

Example: Consider $L = NP$, i.e. the oracle is a poly-time NTM (that leaves its result on the oracle tape)

P^{NP} includes both NP and $coNP$

To solve a problem in NP (resp. $coNP$) a single call to the oracle is sufficient.

We get



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

NP^NP

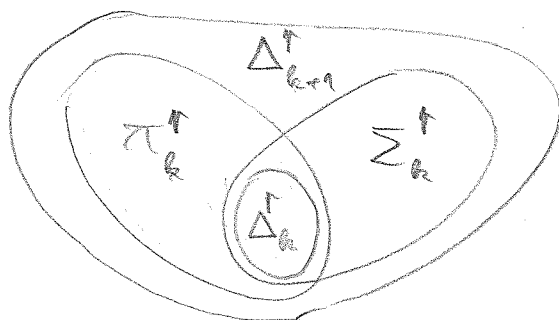
Exploiting this idea, we can define a hierarchy of classes of greater and greater apparent difficulty:

$$\Sigma_0^+ = \Pi_0^+ = \Delta_0^+ = P$$

$$\text{and for all } k \geq 0: \Delta_{k+1}^+ = P^{\Sigma_k^+}$$

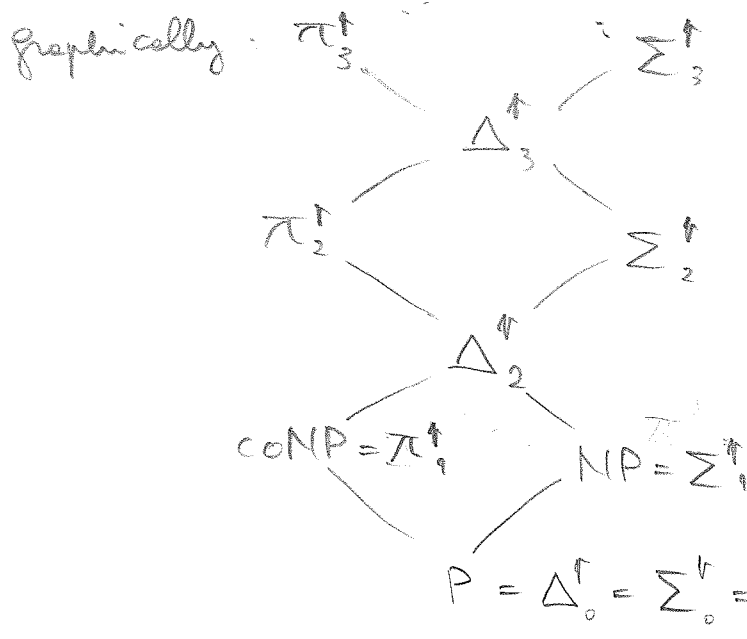
$$\Sigma_{k+1}^+ = NP^{\Sigma_k^+}$$

$$\Pi_{k+1}^+ = co\text{-}\Sigma_{k+1}^+$$



$$\text{Note: } \Sigma_1^+ = NP^{\Sigma_0^+} = NP^P = NP$$

$$\Pi_1^+ = co\Sigma_1^+ = coNP$$



We define the polynomial hierarchy $PH = \bigcup_{j=1}^{\infty} \Sigma_j^P$.

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

$$\Sigma_j^P = \Pi_j^P = \Sigma_k^P \quad \text{for all } j \geq k$$

In particular, if $P = NP$, then $NP = \Sigma_1^P = \Pi_1^P$ and so $\Sigma_j^P = P$ for all $j \geq 0$, i.e. $PH = P$.

We can define completeness for the various Σ_i^P , Π_i^P , Δ_i^P as we did for NP-completeness.

Are there natural problems that are complete for Σ_i^P , Π_i^P ?

Quantified boolean formulae (QBF)

(6.5)

let X be a set of boolean variables partitioned into

$$X = X_1 \dot{\cup} \dots \dot{\cup} X_i$$

and let F be a propositional formula over X .

Then $\phi = \exists X_1 \forall X_2 \exists X_3 \dots Q X_i F$ is a quantified boolean formula with i alternations of quantifiers (QBF _{i})

ϕ is satisfiable if:

there is an assignment to the variables in X_1 s.t.
for all X_2
there is X_3 s.t.
 $\vdots$
 F is true

$$QSAT_i = \{ \phi \mid \phi \text{ is a QBF}_i \text{ and } \phi \text{ is satisfiable} \}$$

Theorem: For all $i \geq 1$ $QSAT_i$ is Σ_i^1 -complete.

Note: games where players alternate moves can be encoded as a formula of QBF _{i}

It turns out that all problems in PH can be solved by a TM that uses at most polynomial space

$$\text{PSPACE} = \{ L \mid L = \mathcal{L}(M) \text{ for some DTM } M \text{ that uses at most space that is polynomial in its input} \}$$

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 $\text{QSAT}_n \in \Sigma_n^1$ -complete
and $\text{QSAT} \in \text{PSPACE}$ -complete

We can also define NPSPACE in an analogous way

$$\text{NPSPACE} = \{ L \mid L = \mathcal{L}(N) \text{ for some NTM } N \text{ that uses at most polynomial space} \}$$

22/12/2015 (6.7)

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.

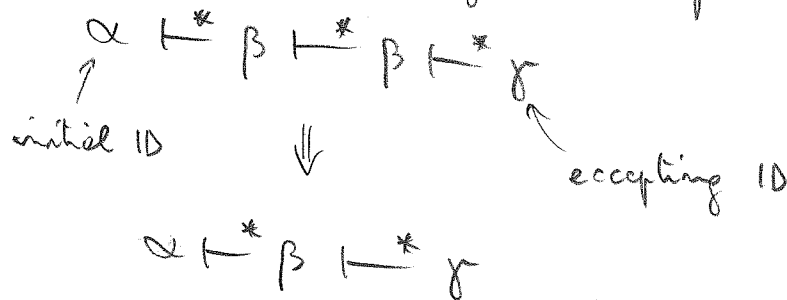
Note: in the definition of (N)PSPACE there is nothing that requires the TM to halt, so it could in principle contain non-recursive languages.

Actually this is not the case:

Theorem: If M is a (N)TM with space bound $p(n)$, and $w \in L(M)$, then w is accepted within $c^{1+p(n)}$ steps, for some constant c .

Proof: Idea: M must repeat an ID before making more than $c^{1+p(n)}$ moves.

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



Why must M repeat an ID?

- the number of symbols in each position is limited (Γ and Q are finite)
- the number of positions is limited by $p(n)$

Let $t = |\Gamma|$, $s = |Q|$, $n = |w|$

with only $q(n)$ tape cells, we have at most

$$K = s \cdot q(n) \cdot t^{q(n)} \quad \text{different IDs}$$

$\uparrow$ states $\uparrow$ positions of head $\uparrow$ symbols in the $q(n)$ positions

Let $c = t + s$ and consider

$$(t+s)^{1+q(n)} = t^{1+q(n)} + (1+q(n)) \cdot s \cdot t^{q(n)} + \dots \geq K \quad \square$$

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 $L \in (N)PSPACE$.

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

Proof: By the previous theorem, we know that there is a (N)TM M accepting every $w \in L$ in at most $c^{1+q(|w|)}$ 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 latest when the counter reaches $c^{1+q(|w|)}$. On tape 1, M_2 simulates M .

If M accepts (within $c^{1+q(|w|)}$ steps), then also M_2 accepts.

If the counter reaches $c^{1+q(|w|)}$ without M accepting, then M_2 stops and rejects.

How much tape uses M_2 ?

- Tape 1: at most $q(|w|)$ cells

- Tape 2: at most $1+q(|w|)$ cells (since the base- c counter counts at most to $c^{1+q(|w|)}$)

The total running time of M_2 is $O(q(n) \cdot c^{1+q(n)})$. (6.9)
increment of counter

We can convert M_2 to a 1-tape TM M_3 :

- M_3 uses no more than $1+q(n)$ tape cells
 (for any input of length n)

- M_3 runs in time quadratic in the running time of M_2 ,
 i.e. $O((c^{q(n)})^2) < c^{q(n)}$ for $q(n) = 2 \cdot q(n) + d$.

□

PSPACE vs. NPSPACE:

Obviously: $PSPACE \subseteq NPSPACE$

Surprisingly: $PSPACE = NPSPACE$!

Idea of proof:

simulation of NTM N with space bound $q(n)$

by DTM D - i.e. - $O(q(n)^2)$

Exploits a deterministic, recursive test for whether a
 NTM N can have $ID_i \xrightarrow[N]{*} ID_j$
 $\leq m$ steps

DTM D tries all intermediate IDs searching for ID_k :

$$ID_i \xrightarrow[N]{*} ID_k \xrightarrow[N]{*} ID_j$$

$\leq \frac{m}{2} \qquad \leq \frac{m}{2}$

We use a recursive boolean function $reach(ID_i, ID_j, m)$:

$$reach(ID_i, ID_j, 1) = \begin{cases} \text{true, if } ID_i = ID_j \text{ or } ID_i \xrightarrow[N]{*} ID_j \\ \text{false, otherwise} \end{cases}$$

$$reach(ID_i, ID_j, m) = \bigvee_{ID_k} (reach(ID_i, ID_k, \frac{m}{2}) \wedge reach(ID_k, ID_j, \frac{m}{2})) \quad \text{for } m > 1$$

Note: reach calls itself several times, i.e. twice
for each possible intermediate ID_k

But - after each ID_k we just need to remember a bit
- (and the counter used to iterate through ID_n)
- the two calls for the same ID_k can be done in
sequence.

What is the depth of the recursive calls?

N does not make more than $c^{p(n)}$ moves

$\Rightarrow$ we can start with $m = c^{p(n)}$

$\Rightarrow$ The recursive call depth is $\leq \log_2 c^{p(n)} = O(p(n))$

Theorem (Levin's Theorem) $PSPACE = NPSPACE$

Proof: we only need to show $NPSPACE \subseteq PSPACE$,
i.e. if $L = L(N)$ for a NTM N with space bound $p(n)$
then $L = L(D)$ for a DTM D $q(n)$
(with $p(n), q(n)$ polynomials)

We can assume that N accepts in $\leq c^{1+p(n)}$ steps
(for some constant c)

Given w with $|w| = n$, D repeatedly calls reach (ID_0, ID_j, m)
where - ID_0 is the initial ID of N with input w
- ID_j is an accepting ID using at most $p(n)$ cells
- $m = c^{1+p(n)}$

The depth of the recursive calls of reach is $\leq \log_2(m)$,
i.e. $O(p(n))$

D can manage its tape as a stack:

for each recursive call, it puts on the stack

$ID_0 \dots 1 + p(n)$ cells

$ID_j \dots 1 + p(n)$ cells

intermediate ID through which to enumerate $\dots 1 + p(n)$ cells

In addition, the tape contains the counter:

$$\log_2 m = \log_2 2^{1+p(n)} = O(p(n)) \text{ cells}$$

The total length of the tape is:

$$O(p(n) + p(n) \cdot 3 \cdot (1 + p(n))) = O(p(n)^2) \text{ cells}$$

□

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 PSPACE-complete

- 1) If $L \in P$, then $P = PSPACE$
- 2) If $L \in NP$, then $NP = PSPACE$

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

Consider a language $L' \in PSPACE$. We show that $L' \in P$.

Since L is PSPACE-complete, we have $L' \leq_{poly} L$.

Let the poly-time reduction be R and let R take time $q(n)$.

Since $L \in P$, it has a poly-time algorithm.

Let this poly-time algorithm run in time $p(n)$.

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 $PSPACE \subseteq P$. The other direction is obvious. $\square$

7/1/2016

We show now a problem that is PSPACE-complete.

Quantified Boolean Formulae (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 is a QBF
- if E and F are QBFs, then so are
 $\neg E$, $E \wedge F$, $E \vee F$
- 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: $\forall x (\exists y (x \wedge y) \vee \forall z (\neg x \vee z))$

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 $F=1$

$$\text{Then } v(F) = F$$

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 $v(F)=0$.
 If $v(E)=0$, then $v(F)=1$.

- $F = E \wedge E'$ then both E and E' are shorter than F ,
and we can evaluate them.
If $v(E) = 1$ and $v(E') = 1$, then $v(F) = 1$
otherwise $v(F) = 0$.

- $F = E \vee E'$ similar

- $F = \forall x (E)$

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

- let E_1 - " -

1

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 $v(E_0) = 1$ and $v(E_1) = 1$, then $v(F) = 1$
otherwise $v(F) = 0$

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

⋮

If $v(E_0) = 0$ and $v(E_1) = 0$, then $v(F) = 0$
otherwise $v(F) = 1$.

We define the QBF problem:

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

We show now that QBF is PSPACE-complete.

Theorem: $QBF \in PSPACE$

(6.15)

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 $F = E \wedge E'$, 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 $F = \neg E$, $F = E \vee E'$, $F = \forall x(E)$ 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 $E \wedge E'$, $E \vee E'$ 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 $|F| = n$, there are at most n records on the stack, and each record has length $O(n)$.

Hence, the used tape is at most $O(n^2)$. □

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

Can we directly use variables $x_{j,t,A}$ 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. $\mathcal{L}(M) = L$, 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. $w(E) = 1$ iff $w \in \mathcal{L}(M)$.

By a previous theorem, we know that there is a constant c s.t. M accepts an input of length n in $c^{1+p(n)}$ 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 $c^{1+p(n)}$ IDs
 $\Rightarrow$ we do not encode all of them explicitly,
 rather we represent an ID through variables over which we quantify. We call the set of variables y_i representing an ID a "variable ID".

When I is the variable ID represented by $x_1, \dots, x_m$

we use $\exists I$ to denote $\exists x_1 \exists x_2 \dots \exists x_m$

" " $\forall I$ " " $\forall x_1 \forall x_2 \dots \forall x_m$

We construct a QBF of the form:

$$\exists I_0. \exists I_f (S \wedge F \wedge N)$$

where

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

Structure of S, F, N :

- starts correctly: 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}
 - " - " - " is not A , then $\overline{y_{jA}}$ $\Rightarrow |S|$ is linear in $p(m)$
- finishes correctly: 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 correctly: N is based on the recursive splitting of the computation in halves, adding only $O(p(m))$ symbols for each split.

- We construct $N_i(I, J)$ for $i = 1, 2, 4, 8, 16, \dots$

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

To encode $I \vdash_M J$, we can proceed as for the proof of Cook's Theorem.

Note: we cannot use

since the overall formula would become exponentially long [Verify as in 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 :

$$(P, Q) = (I, K) \quad \text{or} \quad (P, Q) = (K, J)$$

implies $N_i(P, Q)$ is true

$$\Rightarrow N_{2i}(I, J) = \exists K \forall P \forall Q (N_i(P, Q) \vee (\neg(P=I \wedge Q=K) \wedge \neg(P=K \wedge Q=J)))$$

Then $N = N_m(I_o, I_f)$, where m is the smallest power of 2 greater or equal to $C^{1+q(n)}$. (6.20)

The number of recursive steps to determine N is

$$\log_2(C^{1+q(n)}) = O(q(n))$$

Each recursive step takes time $O(q(n))$.

$\Rightarrow N$ can be constructed in time $O(q(n)^2)$

One can verify that $\exists I_o, \exists I_f (S \wedge F \wedge N)$ has value 1
iff $w \in L(M)$ □

Further important time and space complexity classes:

6.21

$$\text{EXPTIME} = \{L \mid L = \mathcal{L}(M) \text{ for some exptime DTM } M\}$$

$$\text{EXPSPACE} = \{L \mid L = \mathcal{L}(M) \text{ for some expspace DTM } M\}$$

$$\text{kEXPTIME} = \{L \mid L = \mathcal{L}(M) \text{ for some DTM } M \text{ with running time } T(n) = \underbrace{2 \dots 2}_{k \text{ times}}^{O(n)}\}$$

$$\text{kEXPSPACE} = \{L \mid L = \mathcal{L}(M) \text{ for some DTM } M \text{ that, on input of length } n, \text{ uses space that is at most } \underbrace{2 \dots 2}_{k \text{ times}}^{O(n)}\}$$

We can define NkEXPTIME
 NkEXPSPACE

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

We have:

$$\begin{array}{l} \text{kEXPTIME} \\ \text{IN} \leftarrow \text{?} \text{ open whether inclusion is strict} \\ \text{NkEXPTIME} \\ \text{IN} \\ \text{kEXPSPACE} = \text{NkEXPSPACE} \quad \text{The same proof as for Savitch's Theorem can be used.} \\ \text{IN} \\ \text{(k+1)EXPTIME} \end{array}$$

Natural problems in these classes are logic related

Note: EXPTIME is the first provable intractable class,
i.e. we know:

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$$

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