COMP3630 / COMP6363

week 9: **PSPACE to (N)EXPTIME**

This Lecture Covers Chapter 11 of HMU: Additional Classes of Problems

slides created by: By Dirk Pattinson and Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2025

Content of this Chapter

- A **PSPACE**-complete problem: QBF
- **PSPACE** vs. **NPSPACE** (Savitch's Theorem)
- **PSPACE** vs. **co-PSPACE**
- **P** vs. **PSPACE** vs. **EXPTIME**
- Overview and Outlook of other classes

Additional Reading: Chapter 11 of HMU.

A **PSPACE**-complete Problem: QBFs

Quantified Boolean Formulae (QBFs)

Definition w9.1

If V is a set of variables, then the set of quantified boolean formulae over V is given by:

- Every variable $v \in V$ is a QBF, and so are $\top$ and $\bot$
- If ϕ, ψ are QBF, then so are $\phi \wedge \psi$ and $\phi \vee \psi$
- If ϕ is a QBF, then so is $\neg\phi$.
- If ϕ is a QBF and $v \in V$ is a variable, then $(\exists v)\phi$ and $(\forall v)\psi$ are QBF.

Definition w9.2

In a QBF ϕ, a variable v is bound if it is in the scope of a quantifier $\forall v$ or $\exists v$. The variable v is free otherwise.

If $x \in \{\top, \bot\}$ is a truth value, then $\phi[x/v]$ is the result of replacing all free occurrences of v with x (think of / as $=:$, i.e., $x =: v$).

Example

$$(\forall x)\,(\exists y)\,(\,((\exists x)\,(x \vee y)) \wedge \neg(x \wedge y)\,)$$

$$(\forall x)\,(\exists y)\,(\,((\exists x)\,(x \vee y)) \wedge \neg(x \wedge y)\,) = (\forall x)\,(\exists y)\,(\,((\exists z)\,(z \vee y)) \wedge \neg(x \wedge y)\,)$$

- Usually, one writes these formulae without the parentheses pairs around the quantified variables, e.g, $\forall x \phi$ instead of $(\forall x)\phi$.
- Note how inner quantifiers have precedence over outer ones.
- Also, this formula does not have free variables, i.e., all are bound.

The formula above has the following policy as solution:

x	y	z
$\bot$	$\top$	$\top$
$\top$	$\bot$	$\top$

$$x = \bot: \quad (\top \vee \top) \wedge \neg(\bot \wedge \top) = \top \wedge \neg\bot = \top$$
$$x = \top: \quad (\top \vee \bot) \wedge \neg(\top \wedge \bot) = \top \wedge \neg\bot = \top$$

- We have two lines because we need to provide an assignment for each x
- Had we more universally quantified variables, we had more "branching"

Evaluation of QBFs

Observation.

A QBF ϕ without free variables can be evaluated to a truth value:

- $\mathsf{evalQBF}(\forall v \phi) = \phi[\top/v] \wedge \phi[\bot/v]$
- $\mathsf{evalQBF}(\exists v \phi) = \phi[\top/v] \vee \phi[\bot/v]$

and quantifier-free formulae without free variables can be evaluated.

QBFs versus boolean formulae.

A boolean formula ϕ with variables $v_1, \ldots, v_n$ is:

- satisfiable if $\exists v_1 \exists v_2 \ldots \exists v_n \phi$ evaluates to true.
- a tautology if $\forall v_1 \forall v_2 \ldots \forall v_n \phi$ evaluates to true.

Definition w9.3

The QBF problem is the problem of determining whether a given quantified boolean formula without free variables evaluates to true:

$$\mathrm{QBF} = \{\langle \phi \rangle \mid \phi \text{ a true QBF without free variables}\}$$

QBFs vs Boolean Formulae

- Evaluating a boolean formula (not a QBF!) without free variables (i.e., with variables substituted by $\top$ or $\bot$) is in **P**.
- So, an idea is to substitute all bound variables by its truth values:
  - $(\forall v \phi) \rightsquigarrow \phi[\top/v] \wedge \phi[\bot/v]$
  - $(\exists v \phi) \rightsquigarrow \phi[\top/v] \vee \phi[\bot/v]$
- Which runtime does this approach have? Shows **EXPTIME** membership due to doubling the formula with each substitution.

Q. Can we do better?

A. Interestingly, we don't need to reduce the runtime below **EXPTIME**, so it's fine that even a "better result" still requires **EXPTIME**! But we need to reduce the space requirements to **PSPACE** to get to a lower class. (So, note that the procedure above also incurs **EXPSPACE**.)

QBF is in **PSPACE**

Main Idea.

- to evaluate $\forall v \phi$, don't write out $\phi[\top/v] \land \phi[\bot/v]$.
- instead, evaluate $\phi[\top/v]$ and $\phi[\bot/v]$ in sequence (avoids exponential space blowup).

Recursive Algorithm evalQBF(ϕ)

- case $\phi = \top$: return $\top$
- case $\phi = (\psi_1 \land \psi_2)$: if evalQBF($\psi_1$) then return evalQBF($\psi_2$) else return $\bot$
- case $\phi = \forall v \psi$: if evalQBF($\psi[\top/v]$) then return evalQBF($\phi[\bot/v]$) else return $\bot$
- other cases: analogous

Analysis.

Given QBF ϕ of size n:

- at most n recursive calls active
- each call stores a partially evaluated QBF of size n
- total space requirement $\mathcal{O}(n^2)$

This shows **PSPACE** membership of QBF.

QBF is **PSPACE**-hard

Proof Idea/Overview.

To show hardness, we have to reduce any problem in **PSPACE** to QBF:

- Let L be in **PSPACE**.
- Then L is accepted by a polyspace-bounded TM with (cell) bound $p(n)$.
- If $w \in L$, then M accepts in $\leq c^{p(n)}$ moves (for some constant c). Why?
  After an exponential number of moves over the available cells, we will run into a cycle. But by assumption, the TM halts. (Alternatively: how many IDs of some maximal length can we have?)
- Construct QBF ϕ: "there is a sequence of $c^{p(n)}$ IDs that accepts w".
- Use "recursive doubling" to perform this reduction in polytime.

This has similarities to Cook's *SAT* encoding, why?

- For SAT, we used boolean formulae to represent poly many poly-bounded IDs.
- Now, we use QBFs to represent all runs of poly-bounded IDs.

Hardness Proof, Overview

Variables.

- We use two sets of variables, $x_{j,s}$ and $y_{j,s}$. Need $\mathcal{O}(p(n))$ variables to represent an ID:
- variables $x_{j,s}/y_{j,s} = \top$ iff the j-th symbol of the resp. ID, $0 \leq j \leq p(n)$, is s.

Structure of the QBF.

$$\phi = (\exists X)(\exists Y)(S \wedge N \wedge A \wedge U)$$

- We use X as the tuple of all x-variables, and Y as the tuple of all y-variables. They will be used to encode the current and successor configuration.
  - $(\exists \mathbf{X})$ is short for $\exists x_{0,q_0} \dots \exists x_{0,q_{|Q|}} \exists x_{0,\gamma_1} \dots \exists x_{0,\gamma_{|\Gamma|}} \dots \exists x_{p(n),q_0} \dots \exists x_{p(n),\gamma_{|\Gamma|}}$, i.e., we quantify all x variables.
  - $(\exists \mathbf{Y})$ is the very same as X, but works on all the y variables instead.
- **S**: says that X initially represents $ID_0 = q_0 w$, just as in Cook's theorem.
  $x_{0,q_0} \wedge x_{1,w_1} \cdots \wedge x_{|w|,w_{|w|}} \wedge x_{|w|+1,B} \wedge \cdots \wedge x_{p(n),B}$
- **A**: says that Y represents an accepting ID ID_a, just as in Cook's theorem.
  $\bigvee_{\substack{0 \leq i \leq p(n) \\ q \in F}} y_{i,q}$
- **U**: says that every ID has at most one symbol per position, just as in Cook's theorem.
- **N**: transition from $X \approx ID_0$ to some $Y \approx ID_a$ in $\leq c^{p(n)}$ steps (see next slide).

Recursive Doubling

- $N = N(ID_0, ID_a)$: have sequence of length $\leq c^{p(n)}$ from (formula satisfying) ID_0 to (formula satisfying) ID_a. (I.e., from the start ID to some accepting ID)
- Detour: $N_0(X, Y) = X \vdash^* Y$ in ≤ 1 steps: as for Cook's theorem
- Detour: $N_i(X, Y) = X \vdash^* Y$ in $\leq 2^i$ steps:

$$\begin{aligned} N_i(X, Y) = (\exists K)(\forall P)(\forall Q)[& \\ & ((P, Q) = (X, K) \vee (P, Q) = (K, Y)) \\ & \to N_{i-1}(P, Q)] \end{aligned}$$

  - Could also say $(\exists K)(N_{i-1}(X, K) \wedge N_{i-1}(K, Y))$
  - this would write out N_{i-1} twice, doubling formula size at each step
  - above trick is key step in proof to keep formula size small (prevent doubling)
- Let $N(X, Y) = N_k(X, Y)$ where $2^k \geq c^{p(n)}$ (note $k \in \mathcal{O}(p(n))$)
- each N_i can be written in $\mathcal{O}(p(n))$ many steps, plus the time to write N_{i-1}
- so $\mathcal{O}(p(n)^2)$ overall

By construction, $\phi = \top$ iff M accepts w.

PSPACE vs. **NPSPACE**
(Savitch's Theorem)

Savitch's Theorem: **PSPACE** = **NPSPACE**

Note

The following is (maybe?) remarkable because we do not know whether **P** = **NP**.

Theorem w9.1

PSPACE = **NPSPACE** *Savitch's Theorem, 1970*

Proof.

- Let $L \in$ **NPSPACE** and M be non-det. TM, polyspace-bounded by $p(n)$ deciding L.
- We are allowed to assume that M has the following properties:
  - M has just a single accepting state, which is a halting state.
  - When it accepts, the tape is empty.
  - Taken together, there is just a single halting configuration. (We call it J.)
- Recall that M has $c^{p(n)}$ different IDs, were $n = |w|$.
- Design a deterministic TM M', which decides whether $I \vdash^* J$ is possible within at most $c^{p(n)}$ steps. M' is space-bounded by $p(n)$.
- We formalize this via predicate $P(ID_1, ID_2, m)$, initialized to $P(I, J, c^{p(n)})$.
  Wait, isn't the exponential problematic? No, since we encode it logarithmically. □

Savitch's Theorem: Recursive Doubling

Goal. Implement $P(I, J, m) = I \vdash^* J$ in deterministic polyspace

```
P(I, J, m): for all IDs K with length <= p(n) + 1 do {
  if P(I, K, m/2) and P(K, J, m/2) then return true
}
return false
```

Q. How much space does this implementation need? (Time does not matter!)

$P(I, K_0 = J, m)$

$P(I, K_1, {}^m/_2)$ $\quad$ $P(K_1, K_0 = J, {}^m/_2)$

$P(I, K_2, {}^m/_4)$ $\quad$ $P(K_2, K_1 {}^m/_4)$

$\dots \quad \dots$

$P(I, K_i, {}^m/_{2^i})$ $\quad$ $P(K_i, K_{i-1}, {}^m/_{2^i})$

- Required space: $\mathcal{O}(log(c^{p(n)}) \cdot p(n)) = \mathcal{O}(p^2(n))$.

Q. Earlier we were assuming that there's a unique J. Did we have to?

A. No, we could have just generated all possible (accepting) IDs and try all of them!

PSPACE vs. co-PSPACE

Recap

Recall: A problem is in **co-X** if and only if its complement is in **X**.

Applied to $\mathbf{X} = \mathbf{PSPACE}$: Let $\bar{L} = \Sigma^* \setminus L$

$$\mathbf{DSPACE}(t(n)) = \{L \mid \exists \text{ DTM } M \text{ with } L(M) = L \text{ that decides } L \text{ with } \mathcal{O}(t(n)) \text{ cells } \}$$
$$\mathbf{co\text{-}DSPACE}(t(n)) = \{L \mid \bar{L} \in \mathbf{DSPACE}(t(n))\}$$

$$\mathbf{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathbf{DSPACE}(n^k) \qquad \mathbf{co\text{-}PSPACE} = \bigcup_{k \in \mathbb{N}} \mathbf{co\text{-}DSPACE}(n^k)$$

Also, hardness and completeness is again defined as always, also for **co-PSPACE**.

Example $\mathsf{ALL}_{\mathsf{NFA}}$

$$\mathsf{ALL}_{\mathsf{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\mathsf{ALL}_{\mathsf{NFA}} \in$ **NP** nor whether $\mathsf{ALL}_{\mathsf{NFA}} \in$ **co-NP**.

Q. Why don't we know $\in$ **NP**? **A.** Unclear what the certificate should be.
Q. Why don't we know $\in$ **co-NP**? **A.** Words can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\mathsf{ALL}^c_{\mathsf{NFA}}$ – the complement, which accepts $\langle A \rangle$ if $L(A) \neq \Sigma^*$

Technically, the complement also contains all strings that are not even NFAs. But testing for this is trivial, so no need mentioning this!

Example ALL_{NFA}

$$\text{ALL}_{\text{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\text{ALL}_{\text{NFA}} \in$ **NP** nor whether $\text{ALL}_{\text{NFA}} \in$ **co-NP**.

Q. Why don't we know $\in$ **NP**? **A.** Unclear what the certificate should be.
Q. Why don't we know $\in$ **co-NP**? **A.** Words can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\text{ALL}^c_{\text{NFA}}$ – the complement, which accepts $\langle A \rangle$ if $L(A) \neq \Sigma^*$

Let M implement the following non-deterministic procedure when called with input $\langle A \rangle$ and $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA.

1. Mark q_0 (as being visited). If $q_0 \notin F$, accept. // Then, $\epsilon \notin L(A)$, thus $L(A) \neq \Sigma^*$
2. Repeat $2^{|Q|}$ times:
   1. Let $m \subseteq Q$ be the currently marked states.
   2. Non-deterministically pick some $a \in \Sigma$ and change m to $\bigcup_{q \in m} \delta(q, a)$.
   3. If $m \cap F = \emptyset$, accept. // Then, we found a state that's not accepted.
      // I.e., not all reachable states are accepting states, then some word $wa \notin L(A)$.
3. reject // Since we can't find a word that's rejected, so $L(A) = \Sigma^*$

- Hence $\text{ALL}^c_{\text{NFA}} \in$ **NPSPACE** – and thus, by definition, $\text{ALL}_{\text{NFA}} \in$ **co-NPSPACE**
- Since **NPSPACE** = **PSPACE**, we also get $\text{ALL}_{\text{NFA}} \in$ **co-PSPACE**

PSPACE vs. **co-PSPACE**

Theorem w9.1

co-PSPACE = **PSPACE** *(and hence* **co-NPSPACE** = **NPSPACE** = **PSPACE***)*

Proof.

First, show **co-PSPACE** $\supseteq$ **PSPACE**, i.e., $L \in$ **PSPACE** implies $L \in$ **co-PSPACE**.

- Let $L \in$ **PSPACE**. Note that $L \in$ **co-PSPACE** iff $L^c \in$ **PSPACE**.
- Prove L^c in **PSPACE** via:
  - First, decide $w \in L$ using **PSPACE** (possible by assumption).
  - Then, flip result for w. This decides L^c, taking poly-space.

Then, show **PSPACE** $\supseteq$ **co-PSPACE** in the analogous way. □

A lot to unpack here ... Some essential key points to understand:

- Why flipping the result is not allowed in **NP** / **co-NP** proofs,
- but it is allowed in all deterministic classes.
- Thus, **DSPACE** (hence **NSPACE**) and **DTIME** are closed under complementation.

On "Flipping Results", Part 1: Why **NP**/**co-NP** fails.

Disclaimer:

- The following proof is the same as before, but for **NP**/**co-NP**.
- Unless $L \in \mathbf{P}$, or $\mathbf{NP} = \textbf{co-NP}$, the following will fail!

Let $L \in \mathbf{NP}$ and you want to prove $L \in \textbf{co-NP}$:

- Let $L \in \mathbf{NP}$. Note that $L \in \textbf{co-NP}$ iff $L^c \in \mathbf{NP}$.
- Prove L^c in **NP** via:
  - First, decide $w \in L$ using **NP** (possible by assumption).
  - Then, flip result for w. This decides L^c, taking poly-time.

Now, what's wrong about that proof?

- To show $L^c \in \mathbf{NP}$, we need o provide an NTM M with $L(M) = L^c$.
- But the above is not an NTM! It looks like it, but try to give the TM code, you'll fail!
  - Flipping a "yes"-answer to a no is easy: Just turn all accepting states into rejecting ones. So, if there existed an accepting branch, then no more.
  - But flipping the "no"-answer is not possible since we only get a "no" if all branches are rejecting, which we cannot recognize easily. Flipping individual rejecting branches into accepting ones would be wrong since there might be other accepting branches, so the final flipped answer should be "no", not "yes"!
- But for PSPACE, the pseudocode from above does describe a TM! (See next slide!)

On "Flipping Results", Part 2: Why **PSPACE**/**co-PSPACE** works.

In our proof:

- Let $L \in$ **PSPACE**. Note that $L \in$ **co-PSPACE** iff $L^c \in$ **PSPACE**.
- Prove L^c in **PSPACE** via:
  - First, decide $w \in L$ using **PSPACE** (possible by assumption).
  - Then, flip result for w. This decides L^c, taking poly-space.

Why now were we allowed to flip the result?!

- To show $L^c \in$ **PSPACE**, we need o provide a DTM M with $L(M) = L^c$.
- So, this proof implicitly claims that the procedure above is such an M. Is it?
- Yes! The entire procedure describes a DTM M that uses an "inner" DTM M'.
  - M' is a DTM that decides L in **PSPACE** (exists by assumption).
  - M simulates M' on w, which terminates (deterministically) after poly space. M' then flips this result, so this does not change the type of TM and returns the right result.
- The exact same argument works for all deterministic complexity classes.

Corollaries

Corollary w9.2

- *All space classes are closed under complementation.*
- *All deterministic time classes are closed under complementation.*

Corollary w9.3

- $\text{ALL}_{NFA} \in \textbf{PSPACE}$
- $\text{ALL}^c_{NFA} \in \textbf{PSPACE}$

Thus, to prove membership in space or deterministic time classes, you can choose to decide the complement of the language instead. Pick whatever is easier!

P vs. **PSPACE** vs. **EXPTIME**

PSPACE vs. EXPTIME

Theorem w9.1

PSPACE $\subseteq$ **EXPTIME**

Proof.

- Let $L \in$ **PSPACE**. We will show that any **PSPACE** decider runs in exponential time.
- Then, L is decided by some TM M with $L(M) = L$, such that for all $w \in \Sigma^*$ it decides $w \in L$ with $|w| = n$ within $\mathcal{O}(n^k)$ space for some constant k.
- How many different TM configurations can we see before running into a loop? (Note that we can't run into a loop! Since M is a decider.)
  - Each cell can have at most $|\Gamma|$ different symbols.
  - So we have at most $\mathcal{O}(|\Gamma|^{(n^k)})$ different tape configurations.
  - We have $|Q|$ states and at most $\mathcal{O}(n^k)$ head positions.
  - In total we have at most $c^{p(n)} = \mathcal{O}(|Q| \cdot (n^k) \cdot |\Gamma|^{(n^k)})$ TM configurations.
- So, we can just run M and know that it will not run longer than $c^{p(n)}$, hence in **EXPTIME**

$\square$

P vs. **EXPTIME**

Intuitively, **P** $\subsetneq$ **EXPTIME** should hold, since under poly-transformations, we stay withing the respective class, and so have strictly more time available in **EXPTIME**. But, are there really problems that need **EXPTIME**?

To answer this, we require the definition of "small-o" (o), in analogy to "big-O" ($\mathcal{O}$).

Intuitively:

- $f(n) \in \mathcal{O}(g(n))$: f grows at most as fast as g.
- $f(n) \in o(g(n))$: f grows strictly less than g.

Small-o Notation

Definition w9.2

Let $f, g : \mathbb{N} \longrightarrow \mathbb{R}_{\geq 0}$. We say that $f(n) = o(g(n))$ (or $f(n) \in o(g(n))$) if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \ .$$

that is, for any $c > 0$ there exist $n_0 > 0$ such that $f(n) < c \cdot g(n)$, for all $n \geq n_0$.

In comparison:
$f(n) = \mathcal{O}(g(n))$ if there exist $c, n_0 > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

Observe that

1. $f = \mathcal{O}(f)$ but $f \neq o(f)$.
2. $f = o(g) \Rightarrow f = \mathcal{O}(g)$ but in general $f = o(g) \not\Leftarrow f = \mathcal{O}(g)$

Examples:

- $n \neq o(2n)$ (although $2n$ grows faster than n, but only a constant factor)
- $n = o(\frac{1}{2} n \log n)$ and $n \log n = o(n^2)$

The Time Hierarchy Theorem

Theorem w9.3

If $f : \mathbb{N} \longrightarrow \mathbb{R}_{\geq 0}$, then there exists a decision problem which cannot be solved in worst-case deterministic time $o(f(n))$ but can be solved in worst-case deterministic time $\mathcal{O}(f(n)log(f(n)))$. Thus, $\textbf{DTIME}(o(f(n))) \subsetneq \textbf{DTIME}(f(n)log(f(n)))$.

Proof skipped (we only show this for the sake of completeness).

Examples:

- Let $f(n) = n$. Then, $\exists\ L \in \textbf{DTIME}(n \log n)$, but $L \notin \textbf{DTIME}(o(n))$.
- Let $k \geq 1$. Then, $\exists\ L_k \in \textbf{DTIME}(n^{k+1})$, but $L_k \notin \textbf{DTIME}(n^k)$:

  Let $f(n) = n^k$. Then,
  - $\exists L_k \in \textbf{DTIME}\big(n^k \log(n^k)\big) = \textbf{DTIME}\big(n^k k \log(n)\big) \subseteq \textbf{DTIME}\big(n^{k+1}\big)$
  - $L_k \notin \textbf{DTIME}\big(o(f(n))\big) = \textbf{DTIME}\big(o(n^k)\big)$.

Back to **P** vs. **EXPTIME**

Corollary w9.4

$\mathbf{P} \subsetneq \mathbf{EXPTIME}$

Proof.

Let $f(n) = 2^n$. By the Time Hierarchy Theorem there is a language L such that

$L \in \mathbf{DTIME}\big(f(n) log(f(n))\big) = \mathbf{DTIME}\big(2^n n\big)$ and $L \notin \mathbf{DTIME}\big(o(f(n))\big) = \mathbf{DTIME}\big(o(2^n)\big)$.

Since every polynomial n^k satisfies $n^k = o(2^n)$, no poly-time TM decides L. Thus, $L \notin \mathbf{P}$.
Also, $n\,2^n = \mathcal{O}\big(2^{cn}\big)$ for some constant $c > 1$, so $L \in \mathbf{DTIME}(2^{cn}) \subseteq \mathbf{EXPTIME}$.

Therefore, $\mathbf{P} \subsetneq \mathbf{EXPTIME}$. □

Relationship Among Complexity Classes: Recap/Summary (Part 1)

$$\mathbf{P} \overset{(1)}{\subseteq} \mathbf{NP} \overset{(2)}{\subseteq} \mathbf{PSPACE} \overset{(3)}{\subseteq} \mathbf{EXPTIME} \overset{(4)}{\subseteq} \mathbf{NEXPTIME} \overset{(5)}{\subseteq} \mathbf{EXPSPACE}$$

$$\mathbf{PSPACE} \overset{(7)}{\neq} \mathbf{EXPSPACE}$$

$$\mathbf{P} \overset{(6)}{\neq} \mathbf{EXPTIME}$$

Where/how proven?

- (1),(4): Follows trivially: DTMs are a special case of NTSMs.
- (2): Trivial with **NP** $\subseteq$ **NPSPACE**: time is clearly a subset of space.
- (3): Proved today: Search over all reachable configurations.
- (5): Didn't cover that explicitly, but also follows, since Savitch's theorem also applies to higher space classes. Hence, this theorem is trivial with **NEXPTIME** $\subseteq$ **NEXPSPACE**. (And we exploit **NEXPSPACE** $=$**EXPSPACE**.)
- (6): Follows from the time hierarchy theorem.
- (7): Follows from the space hierarchy theorem (not covered).

Relationship Among Complexity Classes: Recap/Summary (Part 2)

Relationships to N- and co-classes:

$$\begin{array}{ccccccccccc}
\textbf{co-P} & \subseteq & \textbf{co-NP} & \subseteq & \textbf{co-PSPACE} & \subseteq & \textbf{co-EXPTIME} & \subseteq & \textbf{co-NEXPTIME} & \subseteq & \textbf{co-EXPSPACE} \\
= & & ? & & = & & = & & ? & & = \\
\textbf{P} & \subseteq & \textbf{NP} & \subseteq & \textbf{PSPACE} & \subseteq & \textbf{EXPTIME} & \subseteq & \textbf{NEXPTIME} & \subseteq & \textbf{EXPSPACE} \\
 & & & & = & & & & & & = \\
 & & & & \textbf{NPSPACE} & & & & & & \textbf{NEXPSPACE} \\
 & & & & = & & & & & & = \\
 & & & & \textbf{co-NPSPACE} & & & & & & \textbf{co-NEXPSPACE}
\end{array}$$

Voluntary homework:
Check all additional subset relations for their origin. (I.e., why they hold.)

Outlook

What we did not cover; some highlights:

- Between **P** and **NP** sit the **GI**-complete problems (GI = Graph Isomorphy), the question whether two graphs can be renamed to make them isomorphic. Practically extremely relevant!
- Between **NP** and **PSPACE** sits the polynomial hierarchy – an infinite hierarchy of complexity classes. They can be defined via special cases of QBFs. At least Σ_2^p (the next harder class after $\Sigma_1^p = \mathbf{NP}$) is very important for many optimization problems.
- There are probabilistic complexity classes, where TMs have different acceptance criteria (and potentially probabilities).
- The chain of complexity classes is infinite. E.g., for any $k \geq 1$, there is a class $k-$**EXPTIME** and $k-$**EXPSPACE**.
- Beyond that are even others, such as the **Ackermann** class, sitting above all $k-$**EXPTIME** and $k-$**EXPSPACE** classes.
- There are TM models with different kinds of states (existential and universal), which are convenient for some proofs/problems. They are covered in week 10!
- **And certainly many more ...** Check out complexityzoo.net – 550 classes so far!)