# #P: Counting Problems

Stepan Kuznetsov

Discrete Math Bridging Course, HSE University

# The NP Class

- There are several equivalent definitions of the NP class.

# The NP Class

- There are several equivalent definitions of the NP class.
- Today we shall use Definition 2, with *hints*.

# The NP Class

- There are several equivalent definitions of the NP class.
- Today we shall use Definition 2, with *hints*.
- Denote the decision problem by $A(x)$.

$$A(x) = 1 \iff \exists y \, (|y| < q(|x|) \,\&\, R(x, y) = 1),$$

where $R \in P$.

# The NP Class

- There are several equivalent definitions of the NP class.
- Today we shall use Definition 2, with *hints*.
- Denote the decision problem by $A(x)$.

  $$A(x) = 1 \iff \exists y \, (|y| < q(|x|) \,\&\, R(x, y) = 1),$$

  where $R \in P$.
- Let us check $|y| < q(|x|)$ inside $R$.

# The NP Class

- There are several equivalent definitions of the NP class.
- Today we shall use Definition 2, with *hints*.
- Denote the decision problem by $A(x)$.

  $$A(x) = 1 \iff \exists y \, (|y| < q(|x|) \,\&\, R(x, y) = 1),$$

  where $R \in P$.
- Let us check $|y| < q(|x|)$ inside $R$.
- $y$ is a *hint*, given by someone to help us solve the problem.

# The NP Class

- There are several equivalent definitions of the NP class.
- Today we shall use Definition 2, with *hints*.
- Denote the decision problem by $A(x)$.

  $$A(x) = 1 \iff \exists y \, (|y| < q(|x|) \,\&\, R(x, y) = 1),$$

  where $R \in P$.
- Let us check $|y| < q(|x|)$ inside $R$.
- $y$ is a *hint*, given by someone to help us solve the problem.
- Examples of $y$: the satisfying assignment; the Hamiltonian cycle; ...

# Beyond Decision Problems

- An NP **decision** problem is the question **whether there exists** a *witness* $y$ such that $R(x, y) = 1$.

# Beyond Decision Problems

- An NP **decision** problem is the question **whether there exists** a *witness* $y$ such that $R(x, y) = 1$.
  - E.g., a satisfying assignment for $\varphi$.

# Beyond Decision Problems

- An NP **decision** problem is the question **whether there exists** a *witness* $y$ such that $R(x, y) = 1$.
  - E.g., a satisfying assignment for $\varphi$.
- We could ask for **all** witnesses, and the algorithm can yield them with **polynomial delay.**

# Beyond Decision Problems

- An NP **decision** problem is the question **whether there exists** a *witness* $y$ such that $R(x, y) = 1$.
  - E.g., a satisfying assignment for $\varphi$.
- We could ask for **all** witnesses, and the algorithm can yield them with **polynomial delay.**
- **Search problem:** yield a witness or say "no."

# Beyond Decision Problems

- An NP **decision** problem is the question **whether there exists** a *witness* $y$ such that $R(x, y) = 1$.
  - E.g., a satisfying assignment for $\varphi$.
- We could ask for **all** witnesses, and the algorithm can yield them with **polynomial delay.**
- **Search problem:** yield a witness or say "no."
- **Counting problem** (the #P class): yield the **number** of witnesses.

# Beyond Decision Problems

- *A priori*, the decision problem is the easiest one.

# Beyond Decision Problems

- *A priori*, the decision problem is the easiest one.
- Indeed, if we can solve the search problem or the counting problem, then we automatically get a solution for the decision problem (with the same $R$).

# Beyond Decision Problems

- *A priori*, the decision problem is the easiest one.
- Indeed, if we can solve the search problem or the counting problem, then we automatically get a solution for the decision problem (with the same $R$).
- However, search problems are also not harder than decision ones.

# Beyond Decision Problems

- *A priori*, the decision problem is the easiest one.
- Indeed, if we can solve the search problem or the counting problem, then we automatically get a solution for the decision problem (with the same $R$).
- However, search problems are also not harder than decision ones.
- Namely, if $\mathrm{P} = \mathrm{NP}$, then any search problem is also solvable in polynomial time.

# Beyond Decision Problems

- *A priori*, the decision problem is the easiest one.
- Indeed, if we can solve the search problem or the counting problem, then we automatically get a solution for the decision problem (with the same $R$).
- However, search problems are also not harder than decision ones.
- Namely, if $\mathsf{P} = \mathsf{NP}$, then any search problem is also solvable in polynomial time.
    - E.g., searching for SAT can be done via dichotomy using decision for SAT.

# Counting Problems

- #P is the class of counting problems corresponding to NP decision problems.

# Counting Problems

- #P is the class of counting problems corresponding to NP decision problems.
- Counting problems can be harder than the corresponding decision ones!

# Counting Problems

- #P is the class of counting problems corresponding to NP decision problems.
- Counting problems can be harder than the corresponding decision ones!

**Theorem**

*#2-SAT is not solvable in polynomial time, unless $P = NP$ (while 2-SAT as a decision problem belongs to P).*

# Counting Problems

- #P is the class of counting problems corresponding to NP decision problems.
- Counting problems can be harder than the corresponding decision ones!

**Theorem**

*#2-SAT is not solvable in polynomial time, unless $P = NP$ (while 2-SAT as a decision problem belongs to P).*

- In order to prove theorems like this one, one has to develop the theory of #P-completeness.

# Counting Reductions

- As the theory of NP-completeness is based on polynomial m-reductions (denoted by $A \leq_m^P B$), the theory of #P-completeness is based on counting reductions: $\#A \leq_c^P \#B$.

# Counting Reductions

- As the theory of NP-completeness is based on polynomial m-reductions (denoted by $A \leq_m^P B$), the theory of #P-completeness is based on counting reductions: $\#A \leq_c^P \#B$.
- A counting reduction consists of **two** functions, $f : \Sigma^* \to \Sigma^*$ on input data and $g : \mathbb{N} \to \mathbb{N}$ on counts (results).

# Counting Reductions

- As the theory of NP-completeness is based on polynomial m-reductions (denoted by $A \leq_m^P B$), the theory of #P-completeness is based on counting reductions: $\#A \leq_c^P \#B$.
- A counting reduction consists of **two** functions, $f : \Sigma^* \to \Sigma^*$ on input data and $g : \mathbb{N} \to \mathbb{N}$ on counts (results).
- Recall that $\#A$ and $\#B$ are counting problems, that is,

$$\#A(x) = |\{y \mid R(x, y) = 1\}| \in \mathbb{N},$$

and the same for $\#B$.

# Counting Reductions

- We say that $\#A \leq_c^P \#B$, if there exists a pair of polynomially computable reducing functions $f$ and $g$ such that for any input $x$ we have

$$\#A(x) = g(\#B(f(x))).$$

- This indeed allows to **reduce** $\#A$ to $\#B$. Suppose we know how to solve $\#B$. Then, in order to solve $\#A$, we take $x$, apply $f$, then solve $\#B$ (yielding a natural number) and apply $g$.

# #P-Completeness

- A counting problem $\#B$ is #P-complete, if for any other $\#A \in \#\mathrm{P}$ we have $\#A \leq_c^P \#B$...
- ... just as for NP-completeness.
- Now we can develop a theory of #P-complete problems, which is parallel to the theory of NP-completeness.

# Parsimonious Reductions

- A counting reduction $(f, g)$, where $g$ is identity, $g(n) = n$, is called a **parsimonious** reduction.
- A parsimonious reduction is also a specific kind of m-reduction, since, in particular, $g(0) = 0$, thus, it conveys the answer to the decision problem.

# Parsimonious Reductions

- The reductions in Cook–Levin theorem are parsimonious.

# Parsimonious Reductions

- The reductions in Cook–Levin theorem are parsimonious.
  - Indeed, each trajectory of the non-deterministic run (that is, each value of hint $y$) is represented by exactly one satisfying assignment.

# Parsimonious Reductions

- The reductions in Cook–Levin theorem are parsimonious.
  - Indeed, each trajectory of the non-deterministic run (that is, each value of hint $y$) is represented by exactly one satisfying assignment.
- This yields the counting version of Cook–Levin:

# Parsimonious Reductions

- The reductions in Cook–Levin theorem are parsimonious.
  - Indeed, each trajectory of the non-deterministic run (that is, each value of hint $y$) is represented by exactly one satisfying assignment.
- This yields the counting version of Cook–Levin:

**Theorem**

*#SAT is #P-complete.*

# Cook – Levin Theorem

- The sequence of configurations (protocol) of $A$ on input $x$ is encoded by a binary matrix $(b_{ij})$ of size $(m \cdot p(|x|)) \times p(|x|)$.

# Cook – Levin Theorem

- The sequence of configurations (protocol) of $A$ on input $x$ is encoded by a binary matrix $(b_{ij})$ of size $(m \cdot p(|x|)) \times p(|x|)$.
- Next, we construct a formula $\varphi_x$ with variables $b_{00}, b_{01}, \ldots$ which expresses the fact that this matrix represents a correct protocol of a successful execution.

# Cook – Levin Theorem

$\varphi_x$ is a conjunction of the following claims:

1. the first row represents the configuration with $x$ on the tape, the machine observing its first letter;
2. each next row is obtained from the previous one by one of the rules of the machine;
3. the last row includes state $q_F$ and the answer “yes” (1).

# Cook – Levin Theorem

$\varphi_x$ is a conjunction of the following claims:

1. the first row represents the configuration with $x$ on the tape, the machine observing its first letter;
2. each next row is obtained from the previous one by one of the rules of the machine;
3. the last row includes state $q_F$ and the answer “yes” (1).

This is all expressible as Boolean formulae.

# Parsimonious Reductions

- Tseitin's transformations are also parsimonious.

# Parsimonious Reductions

- Tseitin's transformations are also parsimonious.
- That is, any Boolean formula $\varphi$ can be translated into a 3-CNF $\psi$, such satisfying assignments of $\psi$ are in one-to-one correspondence with those for $\varphi$.

# Parsimonious Reductions

- Tseitin's transformations are also parsimonious.
- That is, any Boolean formula $\varphi$ can be translated into a 3-CNF $\psi$, such satisfying assignments of $\psi$ are in one-to-one correspondence with those for $\varphi$.
    - Values for new variables $t_i$ are restored uniquely.

# Parsimonious Reductions

- Tseitin's transformations are also parsimonious.
- That is, any Boolean formula $\varphi$ can be translated into a 3-CNF $\psi$, such satisfying assignments of $\psi$ are in one-to-one correspondence with those for $\varphi$.
  - Values for new variables $t_i$ are restored uniquely.
- Thus, #3-SAT is also #P-complete.

# Tseitin's Transformations

**Theorem**

*For any Boolean formula $\varphi$, there exists an equisatisfiable 3-CNF $\psi$ of polynomial size.*

- Equisatisfiability means that $\psi$ is satisfiable iff so is $\varphi$.

# Tseitin's Transformations

**Theorem**

*For any Boolean formula $\varphi$, there exists an equisatisfiable 3-CNF $\psi$ of polynomial size.*

- Equisatisfiability means that $\psi$ is satisfiable iff so is $\varphi$.
- Constructing an *equivalent* 3-CNF of polynomial size is not always possible: even translation to CNF can lead to exponential blowup.

# Tseitin's Transformations

- Tseitin's transformations look like translation into 3-address (Assembler-like) code:
  $(a + b) * (c + d)$ is translated to
  "add $a$ $b$ $t_1$; add $c$ $d$ $t_2$; mul $t_1$ $t_2$ $r$"

# Tseitin's Transformations

- Tseitin's transformations look like translation into 3-address (Assembler-like) code:
  $(a + b) * (c + d)$ is translated to
  "add $a$ $b$ $t_1$; add $c$ $d$ $t_2$; mul $t_1$ $t_2$ $r$"
- For each subformula we introduce a new variable and write the corresponding equivalences.

# Tseitin's Transformations

Example: $(p \to q) \vee (q \to (p \to r))$

# Tseitin's Transformations

Example: $(p \to q) \vee (q \to (p \to r))$

$$
\begin{aligned}
&(t_1 \leftrightarrow (p \to q)) \wedge \\
&(t_2 \leftrightarrow (p \to r)) \wedge \\
&(t_3 \leftrightarrow (q \to t_2)) \wedge \\
&(t_4 \leftrightarrow (t_1 \vee t_3)) \wedge \\
&t_4
\end{aligned}
$$

# Tseitin's Transformations

Transform into 3-CNF by the following table:

| | |
|---|---|
| $t_k \leftrightarrow (t_i \wedge t_j)$ | $(\neg t_i \vee \neg t_j \vee t_k) \wedge (t_i \vee \neg t_k) \wedge (t_j \vee \neg t_k)$ |
| $t_k \leftrightarrow (t_i \vee t_j)$ | $(t_i \vee t_j \vee \neg t_k) \wedge (\neg t_i \vee t_k) \wedge (\neg t_j \vee t_k)$ |
| $t_k \leftrightarrow (t_i \rightarrow t_j)$ | $(\neg t_i \vee t_j \vee \neg t_k) \wedge (t_i \vee t_k) \wedge (\neg t_j \vee t_k)$ |
| $t_k \leftrightarrow \neg t_i$ | $(t_i \vee t_k) \wedge (\neg t_i \vee \neg t_k)$ |

# Tseitin's Transformations

Transform into 3-CNF by the following table:

| | |
|---|---|
| $t_k \leftrightarrow (t_i \wedge t_j)$ | $(\neg t_i \vee \neg t_j \vee t_k) \wedge (t_i \vee \neg t_k) \wedge (t_j \vee \neg t_k)$ |
| $t_k \leftrightarrow (t_i \vee t_j)$ | $(t_i \vee t_j \vee \neg t_k) \wedge (\neg t_i \vee t_k) \wedge (\neg t_j \vee t_k)$ |
| $t_k \leftrightarrow (t_i \rightarrow t_j)$ | $(\neg t_i \vee t_j \vee \neg t_k) \wedge (t_i \vee t_k) \wedge (\neg t_j \vee t_k)$ |
| $t_k \leftrightarrow \neg t_i$ | $(t_i \vee t_k) \wedge (\neg t_i \vee \neg t_k)$ |

For our example, we get:

$$(\neg p \vee q \vee \neg t_1) \wedge (p \vee t_1) \wedge (\neg q \vee t_1) \wedge$$
$$(\neg p \vee r \vee \neg t_2) \wedge (p \vee t_2) \wedge (\neg r \vee t_2) \wedge$$
$$(\neg q \vee t_2 \vee \neg t_3) \wedge (q \vee t_3) \wedge (\neg t_2 \vee t_3) \wedge$$
$$(t_1 \vee t_3 \vee \neg t_4) \wedge (\neg t_1 \vee t_4) \wedge (\neg t_3 \vee t_4) \wedge t_4$$

# Parsimonious Reductions

- In contrast, our reduction used for proving NP-hardness of HAMPATH is **not** parsimonious.

# Parsimonious Reductions

- In contrast, our reduction used for proving NP-hardness of HAMPATH is **not** parsimonious.
  - Namely, a satisfying assignment of a 3-CNF $\varphi$ is simulated by *several* Hamiltonian paths in $G_\varphi$.

# Parsimonious Reductions

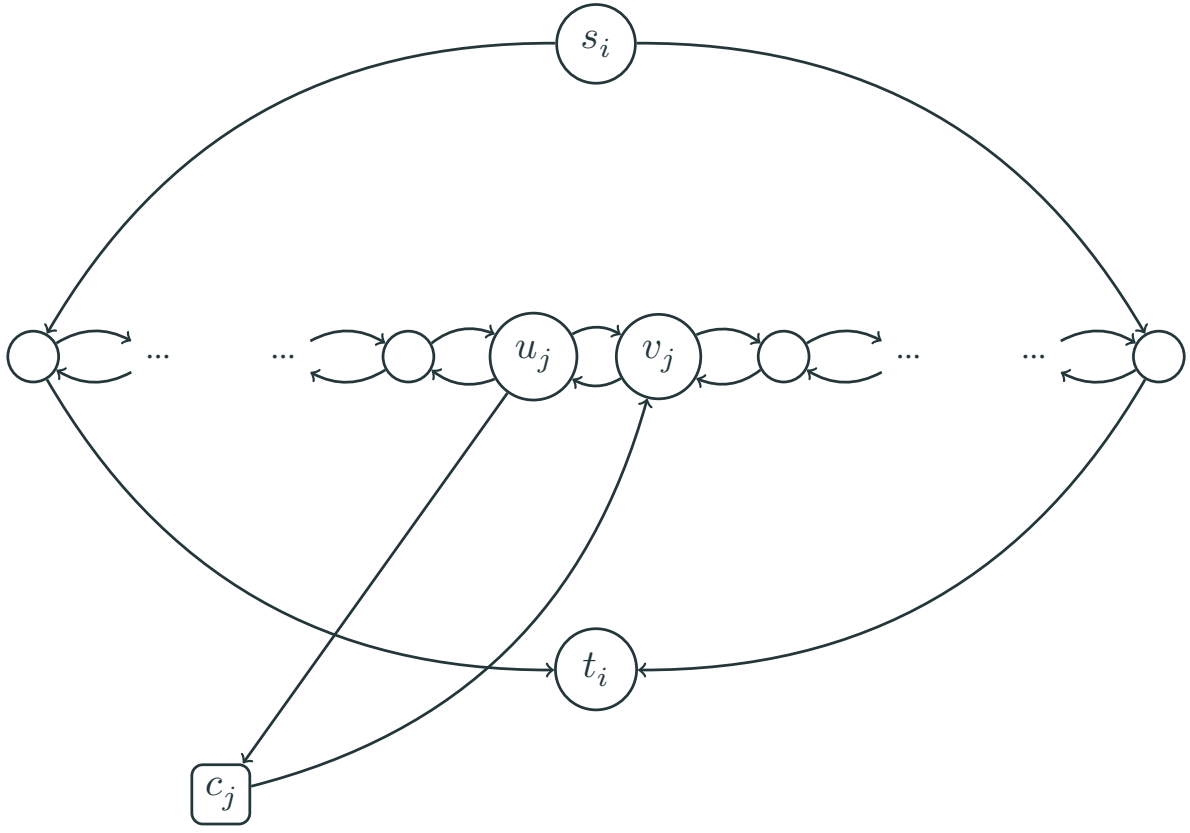
- In contrast, our reduction used for proving NP-hardness of HAMPATH is **not** parsimonious.
  - Namely, a satisfying assignment of a 3-CNF $\varphi$ is simulated by *several* Hamiltonian paths in $G_\varphi$.
- There also exists a parsimonious reduction here.

# Parsimonious Reductions

- In contrast, our reduction used for proving NP-hardness of HAMPATH is **not** parsimonious.
  - Namely, a satisfying assignment of a 3-CNF $\varphi$ is simulated by *several* Hamiltonian paths in $G_\varphi$.
- There also exists a parsimonious reduction here.
  - T. Seta. The Complexities of Puzzles, Cross Sum, and their Another Solution Problems (ASP).

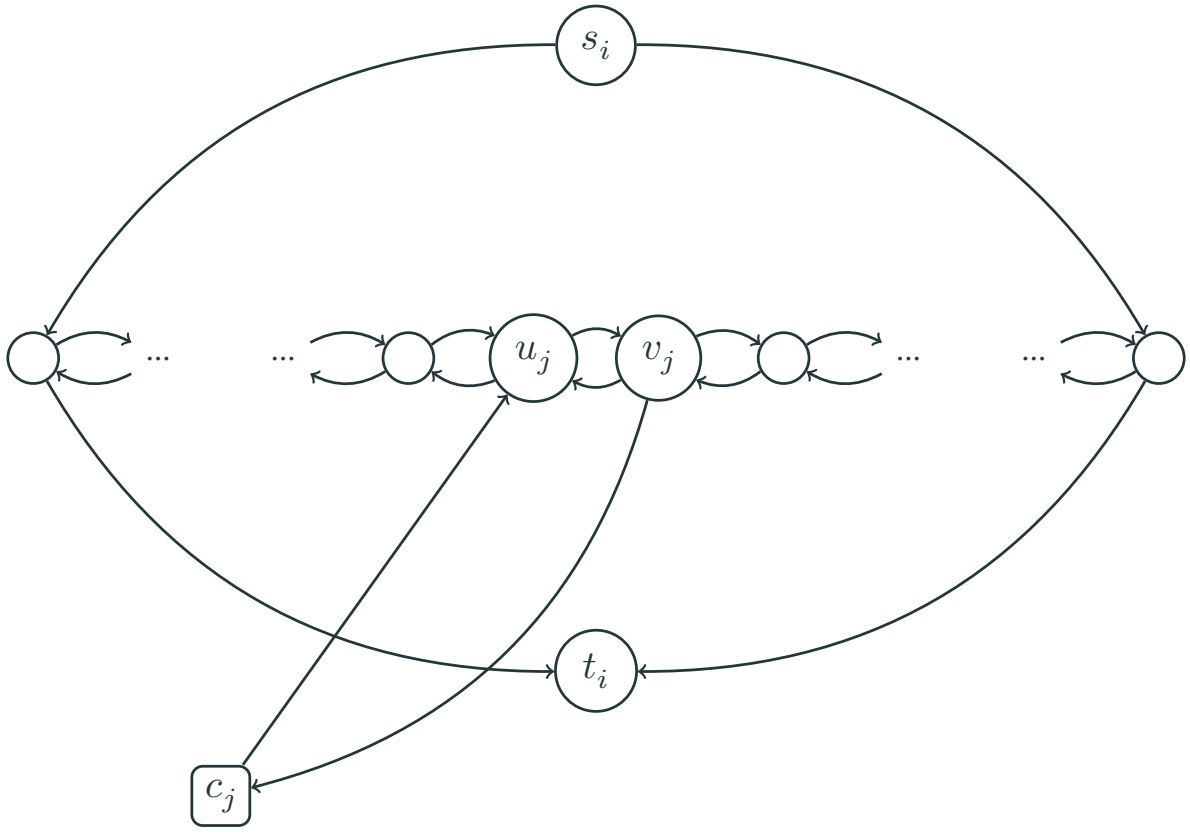
# Modelling Clauses in HAMPATH

Next, for each clause $C_j$ we add a designated vertex $c_j$. If $C_j$ includes $x_i$:



# Modelling Clauses in HAMPATH

If $C_j$ includes $\neg x_i$:



# Beyond Parsimonious Reductions

- Using **only** parsimonious reductions for establishing #P-completeness is meaningless.

# Beyond Parsimonious Reductions

- Using **only** parsimonious reductions for establishing #P-completeness is meaningless.
- Indeed, if a counting problem $\#A$ is proven #P-complete by parsimonious reductions, then its decision variant $A$ is NP-complete.

# Beyond Parsimonious Reductions

- Using **only** parsimonious reductions for establishing #P-completeness is meaningless.
- Indeed, if a counting problem $\#A$ is proven #P-complete by parsimonious reductions, then its decision variant $A$ is NP-complete.
- In this case, if P $\neq$ NP, we know that even $A$ is not polynomially solvable, nothing to say about $\#A$.

# Beyond Parsimonious Reductions

- Using **only** parsimonious reductions for establishing #P-completeness is meaningless.
- Indeed, if a counting problem $\#A$ is proven #P-complete by parsimonious reductions, then its decision variant $A$ is NP-complete.
- In this case, if P $\neq$ NP, we know that even $A$ is not polynomially solvable, nothing to say about $\#A$.
- Using more general counting reductions, however, could give interesting results.

# $A \in$ P, $\#A$ #P-complete

- Interesting cases include situations when the decision problem is polynomially decidable, while the counting problem is hard.

# $A \in$ P, $\#A$ #P-complete

- Interesting cases include situations when the decision problem is polynomially decidable, while the counting problem is hard.
- The famous example is 2-SAT.

# $A \in$ P, $\#A$ #P-complete

- Interesting cases include situations when the decision problem is polynomially decidable, while the counting problem is hard.
- The famous example is 2-SAT.
  - We know that 2-SAT $\in$ P.

# $A \in$ P, $\#A$ #P-complete

- Interesting cases include situations when the decision problem is polynomially decidable, while the counting problem is hard.
- The famous example is 2-SAT.
  - We know that 2-SAT $\in$ P.
  - We shall not give the proof of #P-completeness for #2-SAT, since it is technically hard.

# $A \in$ P, $\#A$ #P-complete

- Interesting cases include situations when the decision problem is polynomially decidable, while the counting problem is hard.
- The famous example is 2-SAT.
  - We know that 2-SAT $\in$ P.
  - We shall not give the proof of #P-completeness for #2-SAT, since it is technically hard.
  - See A. Ben-Dor, S. Halevi (1993), “Zero-one permanent is #P-complete, a simple proof” and L.G. Valiant (1979), “The complexity of enumeration and reliability problems”.

# $A \in$ P, $\#A$ #P-complete

- We shall consider an easier example: DNF-SAT vs. #DNF-SAT.

# $A \in$ P, $\#A$ #P-complete

- We shall consider an easier example: DNF-SAT vs. #DNF-SAT.
- Easily, DNF-SAT $\in$ P (as a decision problem).

# $A \in \mathsf{P}$, $\#A$ #P-complete

- We shall consider an easier example: DNF-SAT vs. #DNF-SAT.
- Easily, DNF-SAT $\in$ P (as a decision problem).
- However, in the counting case we can reduce from CNF-SAT by duality:

$$f(\varphi) = \mathsf{DNF}(\neg\varphi)$$
$$g(n) = 2^k - n$$

(where $k$ is the number of variables).

# $A \in$ P, $\#A$ #P-complete

- Indeed, the set of satisfying assingments for $\varphi$ is the complement of that for $\neg\varphi$.

# $A \in$ P, $\#A$ #P-complete

- Indeed, the set of satisfying assingments for $\varphi$ is the complement of that for $\neg\varphi$.
- If $\varphi$ is in CNF, then DNF($\neg\varphi$) is polynomially computable.

# $A \in$ P, $\#A$ #P-complete

- Indeed, the set of satisfying assingments for $\varphi$ is the complement of that for $\neg\varphi$.
- If $\varphi$ is in CNF, then DNF($\neg\varphi$) is polynomially computable.
- Thus, #CNF-SAT $\leq_c^P$ #DNF-SAT, and therefore #DNF-SAT is #P-complete.

# $A \in$ P, $\#A$ #P-complete

- Indeed, the set of satisfying assingments for $\varphi$ is the complement of that for $\neg\varphi$.
- If $\varphi$ is in CNF, then DNF($\neg\varphi$) is polynomially computable.
- Thus, #CNF-SAT $\leq_c^P$ #DNF-SAT, and therefore #DNF-SAT is #P-complete.
- **Corollary:** if P $\neq$ NP, then #DNF-SAT is not polynomially solvable.

# $A \in$ P, $\#A$ #P-complete

- Indeed, the set of satisfying assingments for $\varphi$ is the complement of that for $\neg\varphi$.
- If $\varphi$ is in CNF, then DNF($\neg\varphi$) is polynomially computable.
- Thus, #CNF-SAT $\leq_c^P$ #DNF-SAT, and therefore #DNF-SAT is #P-complete.
- **Corollary:** if P $\neq$ NP, then #DNF-SAT is not polynomially solvable.
- Otherwise so would be #CNF-SAT, and therefore CNF-SAT, which implies P $=$ NP.