# Are all Problems in **NP** Equivalent?

- The Maximum Independent Set, a problem we already know about, is in **NP**.

# Are all Problems in **NP** Equivalent?

- The Maximum Independent Set, a problem we already know about, is in **NP**.
- The language of, say, palindrome words, is easy to be proved in **P**, thus in **NP**.

# Are all Problems in **NP** Equivalent?

- The Maximum Independent Set, a problem we already know about, is in **NP**.
- The language of, say, palindrome words, is easy to be proved in **P**, thus in **NP**.
- Intuitively, however, the inherent difficulties of solving the two problems *should be* different.

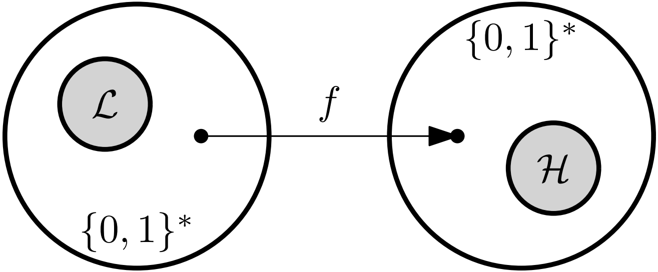
# Are all Problems in **NP** Equivalent?

- The Maximum Independent Set, a problem we already know about, is in **NP**.
- The language of, say, palindrome words, is easy to be proved in **P**, thus in **NP**.
- Intuitively, however, the inherent difficulties of solving the two problems *should be* different.
- What can we thus conclude from the fact that a language $\mathcal{L}$ is **in** the class **NP**?
  - Not much, actually! We can only conclude that it is not *too* complicated to solve it.

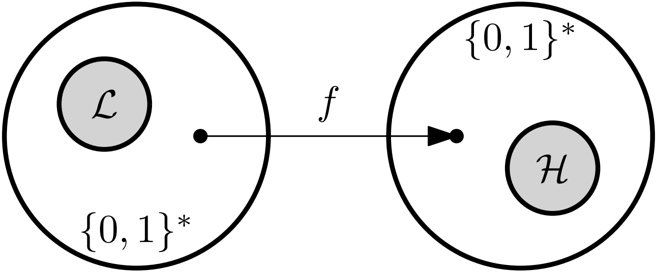
# Are all Problems in **NP** Equivalent?

- The Maximum Independent Set, a problem we already know about, is in **NP**.
- The language of, say, palindrome words, is easy to be proved in **P**, thus in **NP**.
- Intuitively, however, the inherent difficulties of solving the two problems *should be* different.
- What can we thus conclude from the fact that a language $\mathcal{L}$ is **in** the class **NP**?
  - Not much, actually! We can only conclude that it is not *too* complicated to solve it.
- We need something else, namely a (pre-order) relation between languages such that two languages being in relation tells us something precise about the **relative** difficulty of deciding them.

# Reductions



# Reductions




- The language $\mathcal{L}$ is said to be **polynomial-time reducible** to another language $\mathcal{H}$ iff there is a polytime computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that $x \in \mathcal{L}$ iff $f(x) \in \mathcal{H}$.
- In this case, we write $\mathcal{L} \leq_p \mathcal{H}$.

# Reductions and Complexity

- If $\mathcal{L} \leq_p \mathcal{H}$, then $\mathcal{H}$ is at least as difficult as $\mathcal{L}$, at least as far as classes like $\mathbf{P}$ (or above it) are concerned.
  - If, e.g., $\mathcal{L} \leq_p \mathcal{H}$ and $\mathcal{H} \in \mathbf{P}$, then also $\mathcal{L} \in \mathbf{P}$: a way to decide if $x \in \mathcal{L}$ consists in traslating it into $f(x)$ (which can be done in polynomial time), then checking whether $f(x) \in \mathcal{H}$.

# Reductions and Complexity

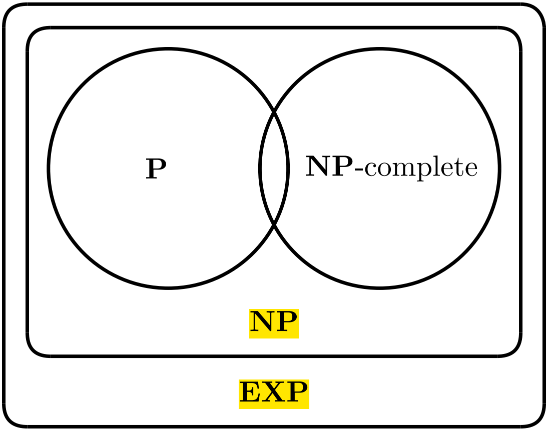
- If $\mathcal{L} \leq_p \mathcal{H}$, then $\mathcal{H}$ is at least as difficult as $\mathcal{L}$, at least as far as classes like $\mathbf{P}$ (or above it) are concerned.
  - If, e.g., $\mathcal{L} \leq_p \mathcal{H}$ and $\mathcal{H} \in \mathbf{P}$, then also $\mathcal{L} \in \mathbf{P}$: a way to decide if $x \in \mathcal{L}$ consists in traslating it into $f(x)$ (which can be done in polynomial time), then checking whether $f(x) \in \mathcal{H}$.
- A language $\mathcal{H} \subseteq \{0,1\}^*$ is said to be:
  - **NP**-hard if $\mathcal{L} \leq_p \mathcal{H}$ for every $\mathcal{L} \in \mathbf{NP}$.
  - **NP**-complete if $\mathcal{H}$ is **NP**-hard, and $\mathcal{H} \in \mathbf{NP}$.

# Reductions and Complexity

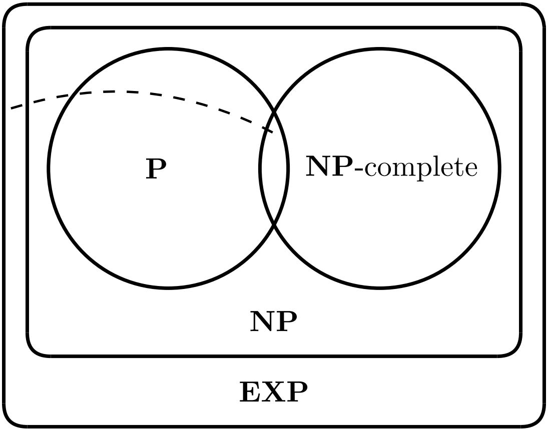
- If $\mathcal{L} \leq_p \mathcal{H}$, then $\mathcal{H}$ is at least as difficult as $\mathcal{L}$, at least as far as classes like $\mathbf{P}$ (or above it) are concerned.
  - If, e.g., $\mathcal{L} \leq_p \mathcal{H}$ and $\mathcal{H} \in \mathbf{P}$, then also $\mathcal{L} \in \mathbf{P}$: a way to decide if $x \in \mathcal{L}$ consists in traslating it into $f(x)$ (which can be done in polynomial time), then checking whether $f(x) \in \mathcal{H}$.
- A language $\mathcal{H} \subseteq \{0,1\}^*$ is said to be:
  - $\mathbf{NP}$-hard if $\mathcal{L} \leq_p \mathcal{H}$ for every $\mathcal{L} \in \mathbf{NP}$.
  - $\mathbf{NP}$-complete if $\mathcal{H}$ is $\mathbf{NP}$-hard, and $\mathcal{H} \in \mathbf{NP}$.

**Theorem**

1. *The relation $\leq_p$ is a pre-order (i.e. it is reflexive and transitive).*
2. *If $\mathcal{L}$ is $\mathbf{NP}$-hard and $\mathcal{L} \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$.*
3. *If $\mathcal{L}$ is $\mathbf{NP}$-complete, then $\mathcal{L} \in \mathbf{P}$ iff $\mathbf{P} = \mathbf{NP}$.*


P
NP-complete
NP
EXP

Empty
if and only if
$\mathbf{P} \neq \mathbf{NP}$



# The First Example of an **NP**-complete Problem

- One obvious way of building an **NP**-complete problem is to define it as the problem of *simulating* any Turing machine, more or less the way we have done it while proving the Hierarchy Theorem.

# The First Example of an **NP**-complete Problem

- One obvious way of building an **NP**-complete problem is to define it as the problem of *simulating* any Turing machine, more or less the way we have done it while proving the Hierarchy Theorem.
- Let TMSAT be the following language:

$$\texttt{TMSAT} = \{(\alpha, x, 1^n, 1^t) \mid \exists u \in \{0,1\}$$ [redacted] $$.\mathcal{M}_\alpha \text{ outputs } 1 \text{ on input } (x, u) \text{ within } t \text{ steps}\}$$

# The First Example of an **NP**-complete Problem

- One obvious way of building an **NP**-complete problem is to define it as the problem of *simulating* any Turing machine, more or less the way we have done it while proving the Hierarchy Theorem.
- Let `TMSAT` be the following language:

$$
\begin{aligned}
\texttt{TMSAT} = \{(\alpha, x, 1^n, 1^t) \mid \exists u \in \{0,1\}^*. \mathcal{M}_\alpha &\text{ outputs } 1\\
&\text{on input } (x, u) \text{ within } t \text{ steps}\}
\end{aligned}
$$

**Theorem**

`TMSAT` *is* **NP**-*complete.*

# The First Example of an **NP**-complete Problem

- One obvious way of building an **NP**-complete problem is to define it as the problem of *simulating* any Turing machine, more or less the way we have done it while proving the Hierarchy Theorem.
- Let `TMSAT` be the following language:

$$\mathtt{TMSAT} = \{(\alpha, x, 1^n, 1^t) \mid \exists u \in \{0,1\}^*. \mathcal{M}_\alpha \text{ outputs } 1 \text{ on input } (x, u) \text{ within } t \text{ steps}\}$$

**Theorem**

`TMSAT` *is* **NP***-complete.*

- Although interesting from a purely theoretical perspective, the language `TMSAT` is very specifically tied to Turing Machines, and thus of no practical importance.

# A Quick Recap on Propositional Logic

- Formulas of **propositional logic** are either:
  - Propositional variables, like $X, Y, Z, \ldots$;
  - Built from smaller formulas by way of the connective $\wedge$, $\vee$ and $\neg$.

  Formulas are indicated as $F, G, H, \ldots$,

# A Quick Recap on Propositional Logic

- Formulas of **propositional logic** are either:
  - Propositional variables, like $X, Y, Z, \ldots$;
  - Built from smaller formulas by way of the connective $\wedge$, $\vee$ and $\neg$.

  Formulas are indicated as $F, G, H, \ldots$,
- Examples: $X \vee \neg X$, $X \wedge (Y \vee \neg Z)$, etc.

# A Quick Recap on Propositional Logic

- Formulas of **propositional logic** are either:
  - Propositional variables, like $X, Y, Z, \ldots;$
  - Built from smaller formulas by way of the connective $\wedge$, $\vee$ and $\neg$.

  Formulas are indicated as $F, G, H, \ldots,$
- Examples: $X \vee \neg X$, $X \wedge (Y \vee \neg Z)$, etc.
- Given a formula $F$ and an assignment $\rho$ of elements from $\{0, 1\}$ to the propositional variables in $F$, one can define the **truth value** for $F$, indicated as $[\![F]\!]$, by induction on $F$:

$$
\begin{aligned}
[\![X]\!] &= \rho(X) & \qquad [\![F \vee G]\!] &= [\![F]\!] + [\![G]\!] \\
[\![F \wedge G]\!] &= [\![F]\!] \cdot [\![G]\!] & \qquad [\![\neg F]\!] &= 1 - [\![F]\!]
\end{aligned}
$$

# A Quick Recap on Propositional Logic

- Formulas of **propositional logic** are either:
  - Propositional variables, like $X, Y, Z, \ldots;$
  - Built from smaller formulas by way of the connective $\wedge$, $\vee$ and $\neg$.

  Formulas are indicated as $F, G, H, \ldots,$
- Examples: $X \vee \neg X$, $X \wedge (Y \vee \neg Z)$, etc.
- Given a formula $F$ and an assignment $\rho$ of elements from $\{0,1\}$ to the propositional variables in $F$, one can define the **truth value** for $F$, indicated as $[\![F]\!]$, by induction on $F$:

$$
\begin{aligned}
[\![X]\!] &= \rho(X) & \qquad [\![F \vee G]\!] &= [\![F]\!] + [\![G]\!] \\
[\![F \wedge G]\!] &= [\![F]\!] \cdot [\![G]\!] & \qquad [\![\neg F]\!] &= 1 - [\![F]\!]
\end{aligned}
$$

- Examples: $[\![X \vee \neg X]\!] = 1$ for every $\rho$, while the truth value $[\![X \wedge (Y \vee \neg Z)]\!]$ equals 1 only for some of the possible $\rho$.

# A Quick Recap on Propositional Logic

- Formulas of **propositional logic** are either:
  - Propositional variables, like $X, Y, Z, \ldots$;
  - Built from smaller formulas by way of the connective $\wedge$, $\vee$ and $\neg$.

  Formulas are indicated as $F, G, H, \ldots,$
- Examples: $X \vee \neg X$, $X \wedge (Y \vee \neg Z)$, etc.
- Given a formula $F$ and an assignment $\rho$ of elements from $\{0, 1\}$ to the propositional variables in $F$, one can define the **truth value** for $F$, indicated as $[\![F]\!]$, by induction on $F$:

$$
\begin{aligned}
[\![X]\!] &= \rho(X) & \qquad [\![F \vee G]\!] &= [\![F]\!] + [\![G]\!] \\
[\![F \wedge G]\!] &= [\![F]\!] \cdot [\![G]\!] & \qquad [\![\neg F]\!] &= 1 - [\![F]\!]
\end{aligned}
$$

- Examples: $[\![X \vee \neg X]\!] = 1$ for every $\rho$, while the truth value $[\![X \wedge (Y \vee \neg Z)]\!]$ equals 1 only for some of the possible $\rho$.
- A formula $F$ is **satisfiable** iff there is one $\rho$ such that $[\![F]\!] = 1$.

# The Cook-Levin Theorem

- A propositional formula $F$ is said to be in **conjunctive normal form** (or a **CNF**) when it is a conjuction of disjunctions of *literals* (a literal being a variable or its negation).
- Examples: $X \vee \neg X$ and $X \wedge (Y \vee \neg Z)$ are both CNFs, while a formula which is *not* a CNF is $X \vee (Y \wedge \neg Z)$.

# The Cook-Levin Theorem

- A propositional formula $F$ is said to be in **conjunctive normal form** (or a **CNF**) when it is a conjuction of disjunctions of *literals* (a literal being a variable or its negation).
- Examples: $X \vee \neg X$ and $X \wedge (Y \vee \neg Z)$ are both CNFs, while a formula which is *not* a CNF is $X \vee (Y \wedge \neg Z)$.
- The disjunctions in a CNF are said to be **clauses**, and a $k$**CNF** is a CNF whose clauses contains at most $k \in \mathbb{N}$ literals. Examples: the two formulas $X \vee \neg X$ and $X \wedge (Y \vee \neg Z)$ are 2CNFs, but not 1CNFs.

# The Cook-Levin Theorem

- A propositional formula $F$ is said to be in **conjunctive normal form** (or a **CNF**) when it is a conjuction of disjunctions of *literals* (a literal being a variable or its negation).
- Examples: $X \vee \neg X$ and $X \wedge (Y \vee \neg Z)$ are both CNFs, while a formula which is *not* a CNF is $X \vee (Y \wedge \neg Z)$.
- The disjunctions in a CNF are said to be **clauses**, and a $k$**CNF** is a CNF whose clauses contains at most $k \in \mathbb{N}$ literals. Examples: the two formulas $X \vee \neg X$ and $X \wedge (Y \vee \neg Z)$ are 2CNFs, but not 1CNFs.

**Theorem (Cook-Levin)**

*The following two languages are* **NP***-complete:*

$$\mathtt{SAT} = \{\llcorner F \lrcorner \mid F \text{ is a satisfiable CNF}\}$$
$$\mathtt{3SAT} = \{\llcorner F \lrcorner \mid F \text{ is a satisfiable 3CNF}\}$$

# Thank You!

Questions?