The Cook-Levin Theorem

- The proof of the Cook-Levin Theorem is outside the scope of this course.
  - It would require at least one lecture, alone!.

The Cook-Levin Theorem

- The proof of the Cook-Levin Theorem is outside the scope of this course.
  - It would require at least one lecture, alone!.
- The **structure** of the proof is as follows:
  - We consider any language $\mathcal{L} \in \mathbf{NP}$, and we show that $\mathcal{L} \leq_p \texttt{SAT}$.

The Cook-Levin Theorem

- The proof of the Cook-Levin Theorem is outside the scope of this course.
  - It would require at least one lecture, alone!.
- The **structure** of the proof is as follows:
  - We consider any language $\mathcal{L} \in \mathbf{NP}$, and we show that $\mathcal{L} \leq_p \texttt{SAT}$.
  - To do that, we consider any possible polynomial p and any polytime deterministic TM $\mathcal{M}$.
    - Intuitively, these exist because $\mathcal{L} \in \mathbf{NP}$

The Cook-Levin Theorem

- The proof of the Cook-Levin Theorem is outside the scope of this course.
  - It would require at least one lecture, alone!.
- The **structure** of the proof is as follows:
  - We consider any language $\mathcal{L} \in \mathbf{NP}$, and we show that $\mathcal{L} \leq_p \texttt{SAT}$.
  - To do that, we consider any possible polynomial p and any polytime deterministic TM $\mathcal{M}$.
    - Intuitively, these exist because $\mathcal{L} \in \mathbf{NP}$
  - We define a polynomial-time transformation $x \mapsto \varphi_x$ from strings to CNFs such that

$$\varphi_x \in \texttt{SAT} \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(x,y) = 1$$

    - This is the crucial step, and requires quite some work. It amounts to showing that computation in TM is *inherently local*.

The Cook-Levin Theorem

- The proof of the Cook-Levin Theorem is outside the scope of this course.
  - It would require at least one lecture, alone!.
- The **structure** of the proof is as follows:
  - We consider any language $\mathcal{L} \in \mathbf{NP}$, and we show that $\mathcal{L} \leq_p \texttt{SAT}$.
  - To do that, we consider any possible polynomial p and any polytime deterministic TM $\mathcal{M}$.
    - Intuitively, these exist because $\mathcal{L} \in \mathbf{NP}$
  - We define a polynomial-time transformation $x \mapsto \varphi_x$ from strings to CNFs such that

$$\varphi_x \in \texttt{SAT} \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(x,y) = 1$$

    - This is the crucial step, and requires quite some work. It amounts to showing that computation in TM is *inherently local*.
  - Finally, we need to show that $\texttt{SAT} \leq_p \texttt{3SAT}$.

Proving a Problem Hard

- Suppose you are studying the complexity of a language $\mathcal{L}$, and you are convinced that the underlying problem is **hard**. How should you back your claim?
- By proving that $\mathcal{L} \in \mathbf{P}$?
  - **Of course not**, this way you rather prove that $\mathcal{L}$ is easy.

Proving a Problem Hard

- Suppose you are studying the complexity of a language $\mathcal{L}$, and you are convinced that the underlying problem is **hard**. How should you back your claim?
- By proving that $\mathcal{L} \in \mathbf{P}$?
  - **Of course not**, this way you rather prove that $\mathcal{L}$ is easy.
- By proving that $\mathcal{L} \in \mathbf{EXP}$?
  - **No**, the fact that there is an exponential-time algorithm deciding $\mathcal{L}$ does **not** mean that no polynomial-time algorithm for $\mathcal{L}$ exist.

Proving a Problem Hard

- Suppose you are studying the complexity of a language $\mathcal{L}$, and you are convinced that the underlying problem is **hard**. How should you back your claim?
- By proving that $\mathcal{L} \in \mathbf{P}$?
  - **Of course not**, this way you rather prove that $\mathcal{L}$ is easy.
- By proving that $\mathcal{L} \in \mathbf{EXP}$?
  - **No**, the fact that there is an exponential-time algorithm deciding $\mathcal{L}$ does **not** mean that no polynomial-time algorithm for $\mathcal{L}$ exist.
- By proving that $\mathcal{L} \in \mathbf{NP}$?
  - Again, this **does not** mean much.

Proving a Problem Hard

- Suppose you are studying the complexity of a language $\mathcal{L}$, and you are convinced that the underlying problem is **hard**. How should you back your claim?
- By proving that $\mathcal{L} \in \mathbf{P}$?
  - **Of course not**, this way you rather prove that $\mathcal{L}$ is easy.
- By proving that $\mathcal{L} \in \mathbf{EXP}$?
  - **No**, the fact that there is an exponential-time algorithm deciding $\mathcal{L}$ does **not** mean that no polynomial-time algorithm for $\mathcal{L}$ exist.
- By proving that $\mathcal{L} \in \mathbf{NP}$?
  - Again, this **does not** mean much.
- By proving that $\mathcal{L}$ is $\mathbf{NP}$-complete?
  - **Yes**, this way you prove that the problem is not so hard (being in $\mathbf{NP}$), but not so easy either (unless $\mathbf{P} = \mathbf{NP}$).

Proving a Problem **NP**-complete

- If we want to prove $\mathcal{L}$ to be **NP**-complete, we have to prove two statements:

Proving a Problem **NP**-complete

- If we want to prove $\mathcal{L}$ to be **NP**-complete, we have to prove two statements:
  1. That $\mathcal{L}$ is in **NP**.
     - This amounts to showing that there are p polynomial and $\mathcal{M}$ polytime TM such that $\mathcal{L}$ can be written as

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(x,y) = 1\}$$

     - This is typically rather easy.

Proving a Problem **NP**-complete

- If we want to prove $\mathcal{L}$ to be **NP**-complete, we have to prove two statements:
  1. That $\mathcal{L}$ is in **NP**.
     - This amounts to showing that there are p polynomial and $\mathcal{M}$ polytime TM such that $\mathcal{L}$ can be written as

       $$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(x,y) = 1\}$$

     - This is typically rather easy.
  2. That any other language $\mathcal{H} \in \mathbf{NP}$ is such that $\mathcal{H} \leq_p \mathcal{L}$.
     - We can of course prove the statement directly.
     - More often (e.g. when showing `3SAT` **NP**-complete), one rather proves that $\mathcal{J} \leq_p \mathcal{L}$ for a language $\mathcal{J}$ which is already known to be **NP**-complete.
     - This is correct, simply because $\leq_p$ is transitive:

$$\begin{array}{lllll} \vdots & & & & \\ \mathcal{H} & \xrightarrow{\leq_p} & \mathcal{J} & \xrightarrow{\leq_p} & \mathcal{L} \\ \vdots & & & & \end{array}$$

NP-complete Problems: Examples

Theorem

The Maximum Independent Set Problem `INDSET` *is* **NP***-complete.*

NP-complete Problems: Examples

Theorem

The Maximum Independent Set Problem `INDSET` *is* **NP**-*complete.*

- There is a polytime reduction from `SAT` to `INDSET`.

NP-complete Problems: Examples

Theorem

The Maximum Independent Set Problem `INDSET` *is* **NP***-complete.*

- There is a polytime reduction from `SAT` to `INDSET`.

Theorem

The Subset Sum Problem `SUBSETSUM` *is* **NP***-complete.*

- There are polytime reductions from `3SAT` to a variation `OL3SAT` of it, and from the latter to `SUBSETSUM`.

NP-complete Problems: Examples

Theorem

The Maximum Independent Set Problem `INDSET` *is* **NP**-*complete.*

- There is a polytime reduction from `SAT` to `INDSET`.

Theorem

The Subset Sum Problem `SUBSETSUM` *is* **NP**-*complete.*

- There are polytime reductions from `3SAT` to a variation `OL3SAT` of it, and from the latter to `SUBSETSUM`.

Theorem

The Decisional 0/1 Linear Programming Problem `ILP` *is* **NP**-*complete*

- There is an easy polytime reduction from `SAT` to `ILP`.

The Graph of **NP**-complete Problems

- For any pair $\mathcal{L}, \mathcal{H}$ of **NP**-complete problems, we have that

$$\mathcal{L} \leq_p \mathcal{H} \qquad \mathcal{H} \leq_p \mathcal{L}$$

  - In other words, $\mathcal{L}$ and $\mathcal{H}$ are equivalent.

The Graph of **NP**-complete Problems

- For any pair $\mathcal{L}, \mathcal{H}$ of **NP**-complete problems, we have that

$$\mathcal{L} \leq_p \mathcal{H} \qquad \mathcal{H} \leq_p \mathcal{L}$$

  - In other words, $\mathcal{L}$ and $\mathcal{H}$ are equivalent.
- This being said, defining a reduction from $\mathcal{L}$ to $\mathcal{H}$ is sometimes easy, and sometimes very difficult, and it is instructive to think at **NP**-complete problems as forming a graph, where edges are *natural* polytime reductions
  - A fragment, the one we have encountered so far is the following

$$\begin{tikzcd}
& \mathtt{ILP} \\
\mathtt{SAT} \arrow[ur] \arrow[r] \arrow[dr] & \mathtt{INDSET} \\
& \mathtt{3SAT} \arrow[r] & \mathtt{OL3SAT} \arrow[r] & \mathtt{SUBSETSUM}
\end{tikzcd}$$

The Graph of **NP**-complete Problems

- For any pair $\mathcal{L}, \mathcal{H}$ of **NP**-complete problems, we have that

$$\mathcal{L} \leq_p \mathcal{H} \qquad \mathcal{H} \leq_p \mathcal{L}$$

  - In other words, $\mathcal{L}$ and $\mathcal{H}$ are equivalent.
- This being said, defining a reduction from $\mathcal{L}$ to $\mathcal{H}$ is sometimes easy, and sometimes very difficult, and it is instructive to think at **NP**-complete problems as forming a graph, where edges are *natural* polytime reductions
  - A fragment, the one we have encountered so far is the following

$$\begin{tikzcd}
 & \mathtt{ILP} & & \\
\mathtt{SAT} \arrow[ur] \arrow[r] \arrow[dr] & \mathtt{INDSET} & & \\
 & \mathtt{3SAT} \arrow[r] & \mathtt{OL3SAT} \arrow[r] & \mathtt{SUBSETSUM}
\end{tikzcd}$$

  - In fact, this graph is **huge**: thousands of different problems are known to be **NP**-complete

Mitigating the Computational Difficulty of **NP**-complete Problems

- What if we actually prove a problem $\mathcal{L}$ we are interesting in solving to actually *be* **NP**-complete?
  - Is the hope to solve it for inputs of nontrivial length lost?

Mitigating the Computational Difficulty of **NP**-complete Problems

- What if we actually prove a problem $\mathcal{L}$ we are interesting in solving to actually *be* **NP**-complete?
  - Is the hope to solve it for inputs of nontrivial length lost?
- We know that this implies that, given the state-of-the-art in computational complexity, no polynomial time algorithm for $\mathcal{L}$ is known.

Mitigating the Computational Difficulty of **NP**-complete Problems

- What if we actually prove a problem $\mathcal{L}$ we are interesting in solving to actually *be* **NP**-complete?
  - Is the hope to solve it for inputs of nontrivial length lost?
- We know that this implies that, given the state-of-the-art in computational complexity, no polynomial time algorithm for $\mathcal{L}$ is known.
- On the other hand, given that $\mathcal{L}$ is in **NP**, and that `SAT` is **NP**-complete, we know that $\mathcal{L} \leq_p \texttt{SAT}$, i.e. that instances of $\mathcal{L}$ can be efficiently translated into instances of `SAT`.

Mitigating the Computational Difficulty of **NP**-complete Problems

- What if we actually prove a problem $\mathcal{L}$ we are interesting in solving to actually *be* **NP**-complete?
  - Is the hope to solve it for inputs of nontrivial length lost?
- We know that this implies that, given the state-of-the-art in computational complexity, no polynomial time algorithm for $\mathcal{L}$ is known.
- On the other hand, given that $\mathcal{L}$ is in **NP**, and that `SAT` is **NP**-complete, we know that $\mathcal{L} \leq_p$ `SAT`, i.e. that instances of $\mathcal{L}$ can be efficiently translated into instances of `SAT`.
- This is often **very useful**, because specialised tools for `SAT`, called SAT-solvers do exist.
  - They do not work in polynomial time.
  - Concretely, they work extremely well on a relatively large class of formulas.

Thank You!

Questions?