• DISCLAIMER: These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

• Last time:
  – What is a verifier?
  – Definition of NP: text book introduces them as search problems on page 249. Our verifier is the same as the algorithm $C$.

Most of these are problems that we don’t know algorithms that provably do better than brute force. However, we can define reductions between problems: The idea is to directly convert the inputs of the problems. Since we’re dealing with decision problems, these problems have the same output: true and false (equivalent to ‘yes’ and ‘no’ from the previous lecture). This means one strategy that we can take is to create a routine that converts an input of $\Pi_1$ to an input of $\Pi_2$.

**Definition 0.1.** A decision problem $\Pi_1$ is **polynomial time reducible** to a decision problem $\Pi_2$ if there exists a polynomial time computable function $f$ such that for any input $x$, $\Pi_1(x) = \Pi_2(f(x))$.

We denote this with $\Pi_1 \rightarrow \Pi_2$. It implies that

• if $\Pi_2$ can be solved in polynomial time, then $\Pi_1$ can also be solved in polynomial time.

• if $\Pi_1$ is hard, then so is $\Pi_2$.

Reductions play a key role complexity theory. The Cook-Levin theorem states that every problem in NP can be reduced to 3-SAT. This problem takes a set of variables $x_1, \ldots, x_n$, and defines:

• a **literal** is either an atom $x_i$ or its negation $\neg x_i$.

• A **clause** is the disjunction (“or”) of three literals.

The 3-SAT problem asks, given a propositional formula $\varphi(x_1, \ldots, x_n)$ which is the “and” of finitely many clauses of length 3, does there exist an assignment of either TRUE or FALSE to each $x_i$ which makes $\varphi(x_1, \ldots, x_n)$ evaluate to TRUE?
**Theorem 0.2** (Cook-Levin Theorem). *For any problem \( \Pi \) in NP, we have \( \Pi \rightarrow 3-SAT \).*

This motivated the definition of NP-hard problems:

**Definition 0.3.** A problem is NP-hard if every problem in NP can be reduced to it.

This then leads to the existence of NP-complete problems:

**Definition 0.4.** A problem \( \Pi \) is NP complete if:

1. It is in \( NP \).

2. Every problem in NP can be reduced to it. This is usually shown by exhibiting a NP-hard problem \( \Pi' \) such that \( \Pi' \rightarrow \Pi \).

Our first NP-hard problem is then 3-SAT: it is in NP because we can just exhibit a satisfying set of variable assignments. It’s NP hard by the Cook-Levin theorem.