P:

- The complexity class **P** contains all problems that can be solved in polynomial time. Polynomial time means $O(n^k)$, where k is a constant.
- Formally: P = {L | There is a TM that decides L in polynomial time.}
- **Theorem 1:** L ∈ P iff there is a polynomial-time TM for it. Essentially, a problem is in P iff you could solve it using a TM in polynomial time.
- **Note:** If a language L is in P, then $\overline{L}$ is also in P. I.e. P is closed under complementation.
- Here's how you can prove that a language, L, is in P:
  1. Construct a TM that decides L in polynomial time.
  2. Use P's closure properties.
  3. Reduce the language to a language in P. If $A \leq_P B$ and B ∈ P, then A ∈ P.

NP:

- The complexity class **NP** contains all problems that can be solved in polynomial time by an NTM.
- Formally: NP = {L | There is a NTM that decides L in polynomial time.}
- The NTMs we have seen so far always follow this pattern:
  1. M = On input w:
     a. Nondeterministically guess some object.
     b. Deterministically check whether this was the right guess.
     c. If so, accept. Otherwise, reject.
- **Theorem 2:** L ∈ NP iff there is a deterministic TM V with the following properties:
  1. w ∈ L iff there is some c ∈ Σ* such that V accepts ⟨w, c⟩.
  2. V runs in time polynomial in |w|.
- A TM V with the above property is called a **polytime verifie**r for L.
- The string c is called a **certificate** for w.
- You can think of V as checking the certificate that proves w ∈ L.
- Important properties of V:
  1. If V accepts ⟨w, c⟩, then we're guaranteed w ∈ L.
  2. If V does not accept ⟨w, c⟩, then either
     a. w ∈ L, but you gave the wrong c, or
     b. w ∉ L, so no possible c will work.
- Important properties of the certificate, c:
  1. c must be comprehensive, meaning that all yes-instances have one.
  2. c must be sound, meaning that all no-instances do not have one.
  3. c must be short.
  4. c must be efficiently checkable.
- Here's how you can prove that a language, L, is in NP:
  1. If there is a verifier V for L, we can build a poly-time NTM for L by nondeterministically guessing a certificate c, then running V on w.
  2. If there is a poly-time NTM for L, we can build a verifier for it. The certificate is the sequence of choices the NTM should make, and V checks that this sequence accepts.

  Note that the above 2 ways are the same.

  3. If $L1 \leq_P L2$ and L2 ∈ NP, then L1 ∈ NP.

Here are 4 steps we can do to prove that a language, L, is in NP:

1. Show how to generate certificates for L.
   We can use a NTM to generate all of the strings and say that each one is a certificate.
2. Argue that each certificate is short in size.
3. Explain how the verifier works to validate input.
4. Argue that the verifier works in polytime.

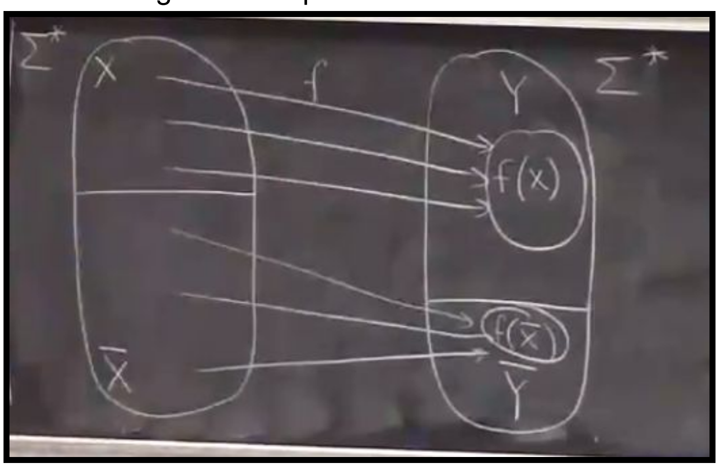
- **Theorem 3:** P $\subseteq$ NP
  Conjecture 3.1: P $\neq$ NP and hence, P $\subset$ NP.

Cook Reduction:

- Let X, Y be problems (not necessarily decision problems). X **cook reduces** to Y , denoted as $X \rightarrow_P Y$ if there exists a polynomial time algorithm A that solves X given an oracle (blackbox subroutine) for Y where each use of the Y-oracle counts as 1 step.
  Note: While using the Y-oracle counts as 1 step per use, A has to prepare input(s) to the Y-oracle. This preparation is not part of the 1 step and is counted separately.
- **Theorem 4:** If $X \rightarrow_P Y$ and there exists a polynomial time algorithm for Y, then there exists a polynomial time algorithm for X.

Karp Reduction/Polytime Reduction:

- Let X, Y $\subseteq \Sigma^*$ be languages. X **karp-reduces/polytime reduces** to Y, denoted as $X \leq_P Y$, iff there exists a function f: $\Sigma^* \rightarrow \Sigma^*$ that can be completed in polynomial time s.t. $x \in X$ iff $f(x) \in Y$.
- Here's a diagram to help with the definition.



  f maps all the Yes-instances of X to a subset of the Yes-instances of Y.
  Similarly, f maps all the No-instances of X to a subset of the No-instances of Y.
- **Theorem 5:** $\leq_P$ is transitive.
  I.e. If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.

<u>NPC:</u>

- Y is a NP-Complete (NPC) language iff
    a. $Y \in NP$
       I.e. Y is in NP.
    b. $\forall X \in NP, X \leq_P Y$
       I.e. Every problem, X, in NP must be polytime reducible to Y.
       This means that Y is the “hardest” in NP.
- **Theorem 6:** If Y is NPC and $Y \in P$ then P=NP.
- **Theorem 7:** If Y is NPC, $Z \in NP$ and $Y \leq_P Z$ then Z is NPC.
- How to Prove $B \in NPC$:
    1. Prove that $B \in NP$.
       You can do this with an NTM or certificate.
    2. Choose problem A, which is known to be NPC.
    3. Describe a polytime reduction of A to B, $A \leq_p B$.
       - Show how to transform any instance of A into an equivalent instance of B.
       - Argue that the transformation is polytime.