21.2

(Polynomial Time) Reductions: Overview
Reductions

A reduction from Problem $\mathbf{X}$ to Problem $\mathbf{Y}$ means (informally) that if we have an algorithm for Problem $\mathbf{Y}$, we can use it to find an algorithm for Problem $\mathbf{X}$.

Using Reductions

- We use reductions to find algorithms to solve problems.
Reductions

A reduction from Problem $\text{X}$ to Problem $\text{Y}$ means (informally) that if we have an algorithm for Problem $\text{Y}$, we can use it to find an algorithm for Problem $\text{X}$.

Using Reductions

1. We use reductions to find algorithms to solve problems.
Reductions

A reduction from Problem $X$ to Problem $Y$ means (informally) that if we have an algorithm for Problem $Y$, we can use it to find an algorithm for Problem $X$.

Using Reductions

1. We use reductions to find algorithms to solve problems.
2. We also use reductions to show that we can’t find algorithms for some problems. (We say that these problems are hard.)
Reductions for decision problems/languages

For languages $L_X, L_Y$, a reduction from $L_X$ to $L_Y$ is:

1. An algorithm ...
2. Input: $w \in \Sigma^*$
3. Output: $w' \in \Sigma^*$
4. Such that:

   \[
   w \in L_X \iff w' \in L_Y
   \]

(Actually, this is only one type of reduction, but this is the one we’ll use most often.) There are other kinds of reductions.
Reductions for decision problems/languages

For languages $L_X, L_Y$, a **reduction from $L_X$ to $L_Y$** is:

1. An algorithm ...
2. Input: $w \in \Sigma^*$
3. Output: $w' \in \Sigma^*$
4. Such that:

   $w \in L_X \iff w' \in L_Y$

(Actually, this is only one type of reduction, but this is the one we'll use most often.)

There are other kinds of reductions.
Reductions for decision problems/languages

For decision problems $X, Y$, a **reduction from $X$ to $Y$** is:

1. An algorithm ...
2. Input: $I_X$, an instance of $X$.
4. Such that:

   $I_Y$ is YES instance of $Y \iff I_X$ is YES instance of $X$
Using reductions to solve problems

1. \( \mathcal{R} \): Reduction \( X \rightarrow Y \)

2. \( \mathcal{A}_Y \): algorithm for \( Y \):

\[ \Rightarrow \quad \text{New algorithm for } X: \]

\[ \mathcal{A}_X(I_X): \]

\[ \quad \text{// } I_X: \text{ instance of } X. \]
\[ I_Y \leftarrow \mathcal{R}(I_X) \]
\[ \text{return } \mathcal{A}_Y(I_Y) \]

If \( \mathcal{R} \) and \( \mathcal{A}_Y \) polynomial-time \( \Rightarrow \) \( \mathcal{A}_X \) polynomial-time.
Using reductions to solve problems

1. \( \mathcal{R} \): Reduction \( X \rightarrow Y \)
2. \( \mathcal{A}_Y \): algorithm for \( Y \):
3. \( \implies \) New algorithm for \( X \):

\[
\mathcal{A}_X(l_X):
\]

// \( l_X \): instance of \( X \).

\[
l_Y \leftarrow \mathcal{R}(l_X)
\]

return \( \mathcal{A}_Y(l_Y) \)

If \( \mathcal{R} \) and \( \mathcal{A}_Y \) polynomial-time \( \implies \) \( \mathcal{A}_X \) polynomial-time.
Using reductions to solve problems

1. \( R \): Reduction \( X \rightarrow Y \)

2. \( A_Y \): algorithm for \( Y \):

3. \( \Rightarrow \) New algorithm for \( X \):

   \[ A_X(l_X) : \]
   
   \[ \begin{align*}
   // & \quad l_X \text{: instance of } X. \\
   l_Y & \leftarrow R(l_X) \\
   \text{return } A_Y(l_Y)
   \end{align*} \]

If \( R \) and \( A_Y \) polynomial-time \( \Rightarrow \) \( A_X \) polynomial-time.
Comparing Problems

1. “Problem $X$ is no harder to solve than Problem $Y$”.

2. If Problem $X$ reduces to Problem $Y$ (we write $X \leq Y$), then $X$ cannot be harder to solve than $Y$.

3. $X \leq Y$:
   - $X$ is no harder than $Y$, or
   - $Y$ is at least as hard as $X$. 
THE END

... 

(for now)