21.2
(Polynomial Time) Reductions: Overview

Reductions

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

Using Reductions

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

Reductions

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

Using Reductions

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

Reductions

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

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:

$$\boxed{w \in L_X} \iff \boxed{w' \in L_Y}$$

(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:

$$\boxed{w \in L_X} \iff \boxed{w' \in L_Y}$$

(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.
3. Output: I_Y an instance of Y.
4. Such that:

$$\boxed{I_Y \text{ is YES instance of } Y} \iff \boxed{I_X \text{ is YES instance of } X}$$

Using reductions to solve problems

1. $\mathcal{R}$: Reduction $\boldsymbol{X} \rightarrow \boldsymbol{Y}$
2. $\mathcal{A}_{\boldsymbol{Y}}$: algorithm for $\boldsymbol{Y}$:
3. $\implies$ New algorithm for $\boldsymbol{X}$:

```
A_X(I_X):
        // I_X: instance of X.
        I_Y ⇐ R(I_X)
        return A_Y(I_Y)
```

If $\mathcal{R}$ and $\mathcal{A}_{\boldsymbol{Y}}$ polynomial-time $\implies$ $\mathcal{A}_{\boldsymbol{X}}$ 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:

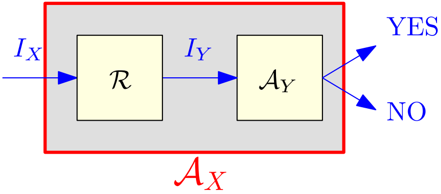
```
A_X(I_X):
        // I_X: instance of X.
        I_Y ⇐ R(I_X)
        return A_Y(I_Y)
```

If $\mathcal{R}$ and $\mathcal{A}_Y$ polynomial-time $\implies$ $\mathcal{A}_X$ polynomial-time.

Using reductions to solve problems

1. $\mathcal{R}$: Reduction $X \to Y$
2. $\mathcal{A}_Y$: algorithm for Y:
3. $\implies$ New algorithm for X:

```
𝒜_X(I_X):
        // I_X: instance of X.
        I_Y ⇐ ℛ(I_X)
        return 𝒜_Y(I_Y)
```



If $\mathcal{R}$ and $\mathcal{A}_Y$ polynomial-time $\implies$ $\mathcal{A}_X$ polynomial-time.

Comparing Problems

1. "Problem $\boldsymbol{X}$ is no harder to solve than Problem $\boldsymbol{Y}$".
2. If Problem $\boldsymbol{X}$ reduces to Problem $\boldsymbol{Y}$ (we write $\boldsymbol{X} \leq \boldsymbol{Y}$), then $\boldsymbol{X}$ cannot be harder to solve than $\boldsymbol{Y}$.
3. $\boldsymbol{X} \leq \boldsymbol{Y}$:
    1. $\boldsymbol{X}$ is no harder than $\boldsymbol{Y}$, or
    2. $\boldsymbol{Y}$ is at least as hard as $\boldsymbol{X}$.

THE END

...

(for now)