Reducibility
Part I
Deciders

- Some Turing machines always halt; they never go into an infinite loop.
- Turing machines of this sort are called **deciders**.
- For deciders, accepting is the same as not rejecting and rejecting is the same as not accepting.
Decidable Languages

- A language $L$ is called **decidable** iff there is a decider $M$ such that $\mathcal{L}(M) = L$.

- Given a decider $M$, you *can* learn whether or not a string $w \in \mathcal{L}(M)$.
  - Run $M$ on $w$.
  - Although it might take a staggeringly long time, $M$ will eventually accept or reject $w$.

- The set $\mathbb{R}$ is the set of all decidable languages.
  $$L \in \mathbb{R} \text{ iff } L \text{ is decidable}$$
The Limits of Computability

- Regular Languages
- CFLs
- \( R \)
- RE
- \( \overline{L}_D \)
- \( \overline{A}_{TM} \)
- \( L_D \)
- \( A_{TM} \)

All Languages
\( A_{TM} \) and \( HALT \)

- Both \( A_{TM} \) and \( HALT \) are undecidable.
  - There is no way to decide whether a TM will accept or eventually terminate.
- However, both \( A_{TM} \) and \( HALT \) are recognizable.
  - We can always run a TM on a string \( w \) and accept if that TM accepts or halts.
- **Intuition**: The only general way to learn what a TM will do on a given string is to run it and see what happens.
Resolving an Asymmetry
The Limits of Computability
The Limits of Computability

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$
The Limits of Computability

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$
There is a TM $M$ where $M$ accepts $w$ iff $w \in L$. 

The Limits of Computability
There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.
There is a TM $M$ where $M$ accepts $w$ iff $w \in L$. 

The Limits of Computability
The Limits of Computability

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$

There is a TM $M$ where $M$ rejects $w$ iff $w \notin L$

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$

There is a TM $M$ where $M$ rejects $w$ iff $w \notin L$
The Limits of Computability

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.

There is a TM $M$ where $M$ rejects $w$ iff $w \notin L$.

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.

There is a TM $M$ where $M$ accepts $w$ iff $w \notin L$. 

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.

There is a TM $M$ where $M$ accepts $w$ iff $w \notin L$. 

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.

There is a TM $M$ where $M$ accepts $w$ iff $w \notin L$. 

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.

There is a TM $M$ where $M$ accepts $w$ iff $w \notin L$. 

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.

There is a TM $M$ where $M$ accepts $w$ iff $w \notin L$. 

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.

There is a TM $M$ where $M$ accepts $w$ iff $w \notin L$. 

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.

There is a TM $M$ where $M$ accepts $w$ iff $w \notin L$. 

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.
A New Complexity Class

- A language $L$ is in $\text{RE}$ iff there is a TM $M$ such that
  - if $w \in L$, then $M$ accepts $w$.
  - if $w \notin L$, then $M$ does not accept $w$.
- A TM $M$ of this sort is called a \textit{recognizer}, and $L$ is called \textit{recognizable}.

- A language $L$ is in $\text{co-RE}$ iff there is a TM $M$ such that
  - if $w \in L$, then $M$ does not reject $w$.
  - if $w \notin L$, then $M$ rejects $w$.
- A TM $M$ of this sort is called a \textit{co-recognizer}, and $L$ is called \textit{co-recognizable}.
RE and co-RE

• Intuitively, **RE** consists of all problems where a TM can exhaustively search for **proof** that \( w \in L \).
  
  • If \( w \in L \), the TM will find the proof.
  • If \( w \notin L \), the TM cannot find a proof.

• Intuitively, **co-RE** consists of all problems where a TM can exhaustively search for a **disproof** that \( w \in L \).
  
  • If \( w \in L \), the TM cannot find the disproof.
  • If \( w \notin L \), the TM will find the disproof.
RE and co-RE Languages

- $A_{TM}$ is an RE language:
  - Simulate the TM $M$ on the string $w$.
  - If you find that $M$ accepts $w$, accept.
  - If you find that $M$ rejects $w$, reject.
  - (If $M$ loops, we implicitly loop forever)

- $\overline{A}_{TM}$ is a co-RE language:
  - Simulate the TM $M$ on the string $w$.
  - If you find that $M$ accepts $w$, reject.
  - If you find that $M$ rejects $w$, accept.
  - (If $M$ loops, we implicitly loop forever)
RE and co-RE Languages

- $\overline{L_D}$ is an RE language.
  - Simulate $M$ on $\langle M \rangle$.
  - If you find that $M$ accepts $\langle M \rangle$, accept.
  - If you find that $M$ rejects $\langle M \rangle$, reject.
  - (If $M$ loops, we implicitly loop forever)

- $L_D$ is a co-RE language.
  - Simulate $M$ on $\langle M \rangle$.
  - If you find that $M$ accepts $\langle M \rangle$, reject.
  - If you find that $M$ rejects $\langle M \rangle$, accept.
  - (If $M$ loops, we implicitly loop forever)
The Limits of Computability

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$

There is a TM $M$ where $M$ rejects $w$ iff $w \notin L$

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$

There is a TM $M$ where $M$ rejects $w$ iff $w \notin L$

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$

There is a TM $M$ where $M$ rejects $w$ iff $w \notin L$

$A_{TM}$

$\overline{A}_{TM}$

$L_D$

$\overline{L}_D$

$ADD$

$0^*1^*$

$HALT$

The diagram illustrates the relationships between different classes of problems in computability theory.
Theorem: \( L \in \text{RE} \) iff \( \overline{L} \in \text{co-RE} \).
RE and co-RE

*Theorem:* \( L \in \text{RE} \) iff \( \overline{L} \in \text{co-RE} \).

*Proof Sketch:* Start with a recognizer \( M \) for \( L \).
RE and co-RE

**Theorem:** $L \in \text{RE}$ iff $\overline{L} \in \text{co-RE}$.

**Proof Sketch:** Start with a recognizer $M$ for $L$. Then, flip its accepting and rejecting states to make machine $M'$. Then, flip the accept and reject states of a co-recognizer for $L$. ■
**RE and co-RE**

**Theorem:** \( L \in \text{RE} \iff \overline{L} \in \text{co-RE}. \)

**Proof Sketch:** Start with a recognizer \( M \) for \( L \). Then, flip its accepting and rejecting states to make machine \( M' \). Then

\[ M' \text{ rejects } w \iff M \text{ accepts } w \iff w \in L \iff w \notin L. \]
RE and co-RE

**Theorem:** $L \in \text{RE}$ iff $\overline{L} \in \text{co-RE}$.

**Proof Sketch:** Start with a recognizer $M$ for $L$. Then, flip its accepting and rejecting states to make machine $M'$. Then

\[
M' \text{ rejects } w \text{ iff } M \text{ accepts } w
\]
**RE and co-RE**

**Theorem:** \( L \in \text{RE} \) iff \( \overline{L} \in \text{co-RE} \).

**Proof Sketch:** Start with a recognizer \( M \) for \( L \). Then, flip its accepting and rejecting states to make machine \( M' \). Then

\[
M' \text{ rejects } w \quad \iff \quad M \text{ accepts } w \quad \iff \quad w \in L \quad \iff \quad w \not\in L
\]
RE and co-RE

*Theorem:* \( L \in \text{RE} \) iff \( \overline{L} \in \text{co-RE} \).

*Proof Sketch:* Start with a recognizer \( M \) for \( L \).

Then, flip its accepting and rejecting states to make machine \( M' \). Then

\[
M' \text{ rejects } w \iff M \text{ accepts } w \iff w \in L \iff w \notin \overline{L}.
\]
RE and co-RE

**Theorem:** $L \in \text{RE}$ iff $\overline{L} \in \text{co-RE}$.

**Proof Sketch:** Start with a recognizer $M$ for $L$. Then, flip its accepting and rejecting states to make machine $M'$. Then

$M'$ rejects $w$ iff $M$ accepts $w$ iff $w \in L$ iff $w \notin \overline{L}$.

$M'$ does not reject $w$ iff $M'$ accepts $w$ or $M'$ loops on $w$ iff $M$ rejects $w$ or $M$ loops on $w$ iff $w \notin L$ iff $w \in L$. 
**Theorem:** \( L \in \text{RE} \) iff \( \overline{L} \in \text{co-RE} \).

**Proof Sketch:** Start with a recognizer \( M \) for \( L \). Then, flip its accepting and rejecting states to make machine \( M' \). Then

\[
M' \text{ rejects } w \text{ iff } M \text{ accepts } w \text{ iff } w \in L \text{ iff } w \notin \overline{L}.
\]

\[
M' \text{ does not reject } w \text{ iff } M' \text{ accepts } w \text{ or } M' \text{ loops on } w.
\]
**Theorem:** $L \in \text{RE}$ iff $\overline{L} \in \text{co-RE}$.

**Proof Sketch:** Start with a recognizer $M$ for $L$. Then, flip its accepting and rejecting states to make machine $M'$. Then

$M'$ rejects $w$ iff $M$ accepts $w$ iff $w \in L$ iff $w \notin \overline{L}$.

$M'$ does not reject $w$ iff $M'$ accepts $w$ or $M'$ loops on $w$ iff $M$ rejects $w$ or $M$ loops on $w$. $w \notin L$ iff $w \in L$. 

**RE and co-RE**
**RE and co-RE**

**Theorem:** \( L \in \text{RE} \iff \overline{L} \in \text{co-RE}. \)

**Proof Sketch:** Start with a recognizer \( M \) for \( L \). Then, flip its accepting and rejecting states to make machine \( M' \). Then

\[
\begin{align*}
M' \text{ rejects } w & \iff M \text{ accepts } w \\
& \iff w \in L \\
& \iff w \notin \overline{L}.
\end{align*}
\]

\[
\begin{align*}
M' \text{ does not reject } w & \iff M' \text{ accepts } w \text{ or } M' \text{ loops on } w \\
& \iff M \text{ rejects } w \text{ or } M \text{ loops on } w \\
& \iff w \notin L.
\end{align*}
\]
**RE and co-RE**

**Theorem:** $L \in \text{RE}$ iff $\overline{L} \in \text{co-RE}$.

**Proof Sketch:** Start with a recognizer $M$ for $L$. Then, flip its accepting and rejecting states to make machine $M'$. Then

- $M'$ rejects $w$ iff $M$ accepts $w$ iff $w \in L$ iff $w \notin \overline{L}$.

- $M'$ does not reject $w$ iff $M'$ accepts $w$ or $M'$ loops on $w$ iff $M$ rejects $w$ or $M$ loops on $w$ iff $w \notin L$ iff $w \in \overline{L}$. 

$\blacksquare$
RE and co-RE

Theorem: $L \in \text{RE}$ iff $\overline{L} \in \text{co-RE}$.

Proof Sketch: Start with a recognizer $M$ for $L$. Then, flip its accepting and rejecting states to make machine $M'$. Then

- $M' \text{ rejects } w$ iff $M \text{ accepts } w$ iff $w \in L$ iff $w \notin \overline{L}$.
- $M' \text{ does not reject } w$ iff $M' \text{ accepts } w$ or $M' \text{ loops on } w$ iff $M \text{ rejects } w$ or $M \text{ loops on } w$ iff $w \notin L$ iff $w \in \overline{L}$.
RE and co-RE

**Theorem:** \( L \in \text{RE} \) iff \( \overline{L} \in \text{co-RE} \).

**Proof Sketch:** Start with a recognizer \( M \) for \( L \).
Then, flip its accepting and rejecting states to make machine \( M' \). Then

\begin{align*}
M' \text{ rejects } w & \iff M \text{ accepts } w \\
& \iff w \in L \\
& \iff w \notin \overline{L}.
\end{align*}

\begin{align*}
M' \text{ does not reject } w & \iff M' \text{ accepts } w \text{ or } M' \text{ loops on } w \\
& \iff M \text{ rejects } w \text{ or } M \text{ loops on } w \\
& \iff w \notin L \\
& \iff w \in \overline{L}.
\end{align*}
Theorem: \( L \in \text{RE} \) iff \( \overline{L} \in \text{co-RE} \).

Proof Sketch: Start with a recognizer \( M \) for \( L \). Then, flip its accepting and rejecting states to make machine \( M' \). Then

\[
M' \text{ rejects } w \quad \text{iff } M \text{ accepts } w \\
\quad \text{iff } w \in L \\
\quad \text{iff } w \notin \overline{L}.
\]

\[
M' \text{ does not reject } w \\
\quad \text{iff } M' \text{ accepts } w \text{ or } M' \text{ loops on } w \\
\quad \text{iff } M \text{ rejects } w \text{ or } M \text{ loops on } w \\
\quad \text{iff } w \notin L \\
\quad \text{iff } w \in \overline{L}.
\]

The same approach works if we flip the accept and reject states of a co-recognizer for \( \overline{L} \).
RE and co-RE

\textbf{Theorem:} \( L \in \text{RE} \) iff \( \overline{L} \in \text{co-RE} \).

\textbf{Proof Sketch:} Start with a recognizer \( M \) for \( L \). Then, flip its accepting and rejecting states to make machine \( M' \). Then

\begin{align*}
M' & \text{ rejects } w \\
& \text{ iff } M \text{ accepts } w \\
& \text{ iff } w \in L \\
& \text{ iff } w \notin \overline{L}.
\end{align*}

\begin{align*}
M' & \text{ does not reject } w \\
& \text{ iff } M' \text{ accepts } w \text{ or } M' \text{ loops on } w \\
& \text{ iff } M \text{ rejects } w \text{ or } M \text{ loops on } w \\
& \text{ iff } w \notin L \\
& \text{ iff } w \in \overline{L}.
\end{align*}

The same approach works if we flip the accept and reject states of a co-recognizer for \( \overline{L} \). ■
There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.

There is a TM $M$ where $M$ rejects $w$ iff $w \notin L$.

The diagram illustrates the relationships between the classes $\text{RE}$, $\text{co-RE}$, $\text{HALT}$, and $\overline{\text{HALT}}$. The acceptance and rejection of inputs $w$ are marked by stars within the respective classes.
R, RE, and co-RE

• Every language in R is in both RE and co-RE.

• Why?
  • A decider for L accepts all \( w \in L \) and rejects all \( w \notin L \).

• In other words, \( R \subseteq RE \cap co-RE \).

• **Question:** Does \( R = RE \cap co-RE \)?
Which Picture is Correct?

**CO-RE**
- $L_D$
- $\overline{\text{HALT}}$
- $A_{TM}$

There is a TM $M$ where $M$ rejects $w$ iff $w \notin L$

**RE**
- $\overline{L_D}$
- $\text{HALT}$
- $A_{TM}$

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$

**RE**
- $ADD$
- $0^*1^*$

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$
Which Picture is Correct?

CO-RE

- $L_D$
- $A_{TM}$
- $\overline{HALT}$
- There is a TM $M$ where $M$ rejects $w$ iff $w \notin L$

R

- $ADD$
- $0^*1^*$

RE

- $L_D$
- $A_{TM}$
- $\overline{HALT}$
- There is a TM $M$ where $M$ accepts $w$ iff $w \in L$
R, RE, and co-RE

• *Theorem:* If $L \in \text{RE}$ and $L \in \text{co-RE}$, then $L \in \text{R}$. 
R, RE, and co-RE

- **Theorem:** If $L \in \text{RE}$ and $L \in \text{co-RE}$, then $L \in \text{R}$.

- **Proof sketch:** Since $L \in \text{RE}$, there is a recognizer $M$ for it.
\textbf{R, RE, and co-RE}

- \textit{Theorem:} If $L \in \text{RE}$ and $L \in \text{co-RE}$, then $L \in \text{R}$.

- \textit{Proof sketch:} Since $L \in \text{RE}$, there is a recognizer $M$ for it. Since $L \in \text{co-RE}$, there is a co-recognizer $\overline{M}$ for it.
Theorem: If $L \in \text{RE}$ and $L \in \text{co-RE}$, then $L \in \text{R}$.

Proof sketch: Since $L \in \text{RE}$, there is a recognizer $M$ for it. Since $L \in \text{co-RE}$, there is a co-recognizer $\overline{M}$ for it.

This TM $D$ is a decider for $L$: 
R, RE, and co-RE

• **Theorem:** If $L \in \text{RE}$ and $L \in \text{co-RE}$, then $L \in \text{R}$.

• **Proof sketch:** Since $L \in \text{RE}$, there is a recognizer $M$ for it. Since $L \in \text{co-RE}$, there is a co-recognizer $\overline{M}$ for it.

This TM $D$ is a decider for $L$:

$$D = "\text{On input } w:\n\text{Run } M \text{ on } w \text{ and } \overline{M} \text{ on } w \text{ in parallel.}\n\text{If } M \text{ accepts } w, \text{ accept.}\n\text{If } \overline{M} \text{ rejects } w, \text{ reject.}"$$
The Limits of Computability

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.

There is a TM $M$ where $M$ rejects $w$ iff $w \notin L$.

There is a TM $M$ where $M$ accepts $w$ iff $w \in L$.

There is a TM $M$ where $M$ rejects $w$ iff $w \notin L$.

What's out here?
Time-Out For Announcements!
Friday Four Square!
Today at 4:15PM outside Gates
Two Handouts Online

• **24: Additional Proofs on TMs**
  • See alternate proofs of why various languages are or are not \( R \), \( \text{RE} \), or \( \text{co-RE} \).

• **25: Extra Practice Problems**
  • By popular demand, extra questions on topics you'd like some more practice with!
  • Solutions released Monday.
Picking up Problem Sets

• If you pick up problem sets from the filing cabinet,

   please put all other papers back into the filing cabinet when you're done!

• If you don't:
  • they get mixed with problem sets from other classes and lost,
  • it causes a fire hazard, and
  • I get flak from the building managers about making a mess.
Your Questions
“Can you recommend software for designing and / or simulating Turing machines?”

http://www.jflap.org/
“Is there a difference between when a TM “runs” another TM as a subroutine vs. when it “simulates running” another TM?”
“Sometime my brain is stuck and I make silly and stupid mistakes [...]. What [do] you do when you are stuck on a problem?”
Back to CS103!
A Repeating Pattern
$L = \{ \langle M \rangle \mid M \text{ is a TM that accepts } \varepsilon \}$

$H = \text{"On input } \langle M \rangle \text{:} $

- Construct the string $\langle M, \varepsilon \rangle$.
- Run $R$ on $\langle M, \varepsilon \rangle$.
- If $R$ accepts $\langle M, \varepsilon \rangle$, then $H$ accepts $\langle M, \varepsilon \rangle$.
- If $R$ rejects $\langle M, \varepsilon \rangle$, then $H$ rejects $\langle M, \varepsilon \rangle$.\"
$H$ = “On input $\langle M \rangle$:

- Construct the string $\langle M, \langle M \rangle \rangle$.
- Run $R$ on $\langle M, \langle M \rangle \rangle$.
- If $R$ accepts $\langle M, \langle M \rangle \rangle$, then $H$ accepts $\langle M, \langle M \rangle \rangle$.
- If $R$ rejects $\langle M, \langle M \rangle \rangle$, then $H$ rejects $\langle M, \langle M \rangle \rangle$.”
From **HALT** to $A_{TM}$

$H = "On input \langle M, w \rangle:\n\begin{itemize}
\item Build $M$ into $M'$ so $M'$ loops when $M$ rejects.
\item Run $D$ on $\langle M', w \rangle$.
\item If $D$ accepts $\langle M', w \rangle$, then $H$ accepts $\langle M, w \rangle$.
\item If $D$ rejects $\langle M', w \rangle$, then $H$ rejects $\langle M, w \rangle."
\end{itemize}$
The General Pattern

Compute $f$  

$f(w)$  

Subroutine TM

Machine $R$

Machine $H$

$w$  

YES

NO
The General Pattern

$H =$ “On input $w$: 
· Transform the input $w$ into $f(w)$.
· Run machine $R$ on $f(w)$.
· If $R$ accepts $f(w)$, then $H$ accepts $w$.
· If $R$ rejects $f(w)$, then $H$ rejects $w$.”
Reductions

- Intuitively, problem $A$ reduces to problem $B$ iff a solver for $B$ can be used to solve problem $A$.

\[ \mathcal{L}(D) = \Sigma^* \? \]
\[ \mathcal{L}(D_1) \text{ equal to } \Sigma^* - \mathcal{L}(D_2) \? \]
Reductions

- Intuitively, problem $A$ reduces to problem $B$ iff a solver for $B$ can be used to solve problem $A$.

Is $\mathcal{L}(G) = \emptyset$?

Can be converted to

Can be used to solve

Problem $A$

Problem $B$

Is $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$?
Reductions

- Intuitively, problem $A$ reduces to problem $B$ iff a solver for $B$ can be used to solve problem $A$.
Reductions

• Intuitively, problem $A$ reduces to problem $B$ iff a solver for $B$ can be used to solve problem $A$.

$A_{TM}$  
Problem $A$  

Can be converted to

$HALT$  
Problem $B$  

Can be used to solve
Reductions

● Intuitively, problem $A$ reduces to problem $B$ iff a solver for $B$ can be used to solve problem $A$.

● Reductions can be used to show certain problems are “solvable:”

   If $A$ reduces to $B$ and $B$ is “solvable,” then $A$ is “solvable.”
Formalizing Reductions

- In order to make the previous intuition more rigorous, we need to formally define reductions.
- There are many ways to do this; we'll explore two:
  - **Mapping reducibility** (today / Monday), and
  - **Polynomial-time reducibility** (next week).
Defining Reductions

- A **reduction** from $A$ to $B$ is a function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that

For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$
Defining Reductions

- A **reduction** from $A$ to $B$ is a function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that

  For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$
Defining Reductions

- A **reduction** from $A$ to $B$ is a function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that

  For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$
Defining Reductions

- A reduction from $A$ to $B$ is a function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that

  \[
  \text{For any } w \in \Sigma_1^*, w \in A \text{ iff } f(w) \in B
  \]

- Every $w \in A$ maps to some $f(w) \in B$.
- Every $w \not\in A$ maps to some $f(w) \not\in B$.
- $f$ does not have to be injective or surjective.
Why Reductions Matter

• If language $A$ reduces to language $B$, we can use a recognizer / co-recognizer / decider for $B$ to recognize / co-recognize / decide problem $A$.
  • (There's a slight catch – we'll talk about this in a second).

• How is this possible?
\( w \in A \quad \text{iff} \quad f(w) \in B \)
\( w \in A \iff f(w) \in B \)
\[ w \in A \quad \text{iff} \quad f(w) \in B \]
\( w \in A \iff f(w) \in B \)
$w \in A$  iff  $f(w) \in B$
$w \in A \text{ iff } f(w) \in B$

Machine $H$ is used to compute $f$ on input $w$. The result, $f(w)$, is then given to Machine $R$ to determine if it is in language $B$. The output of Machine $R$ is used to determine whether $w$ is in language $A$. If the output is YES, then $w \in A$, otherwise, if the output is NO, then $w \notin A$. This diagram illustrates the relationship between the languages $A$ and $B$ through the function $f$. 

Machine $R$ operates on inputs that are the result of $f(w)$, which is computed by Machine $H$. The YES or NO output from Machine $R$ decides whether $w$ is a member of language $A$. 

Diagram Components:
- **Compute $f$**: Computes the function $f$ on input $w$.
- **TM for language $B$**: Determines if $f(w)$ is in language $B$.
- **Machine $R$**: Determines the output (YES or NO) based on the input from Machine $H$.
- **Machine $H$**: Computes $f(w)$ for input $w$. 

The diagram visually represents the process of determining whether an input $w$ belongs to language $A$ by first computing $f(w)$ and then using Machine $R$ to check if $f(w)$ is in language $B$. 

**Mathematical Representation**:

Given $w \in A$, we have:

$$w \in A \iff f(w) \in B$$
$w \in A$ iff $f(w) \in B$

Machine $H$

\[ w \rightarrow \text{Compute } f \rightarrow f(w) \rightarrow \text{TM for language } B \rightarrow \text{Machine } R \rightarrow \begin{cases} \text{YES} & \text{if } R \text{ accepts } f(w) \\ \text{NO} & \text{if } R \text{ rejects } f(w) \end{cases} \]

$H = \text{“On input } w:\$
- Transform the input $w$ into $f(w)$.
- Run machine $R$ on $f(w)$.
- If $R$ accepts $f(w)$, then $H$ accepts $w$.
- If $R$ rejects $f(w)$, then $H$ rejects $w.”$
$w \in A$ iff $f(w) \in B$

$H$ accepts $w$

$H = \text{“On input } w: \text{  
  \begin{itemize}
    \item Transform the input } w \text{ into } f(w).
    \item Run machine } R \text{ on } f(w).
    \item If } R \text{ accepts } f(w), \text{ then } H \text{ accepts } w.
    \item If } R \text{ rejects } f(w), \text{ then } H \text{ rejects } w.\text{”}$
$w \in A$ \iff $f(w) \in B$

$H = \text{"On input } w:\n\begin{itemize}
  \item Transform the input } w \text{ into } f(w).
  \item Run machine } R \text{ on } f(w).
  \item If } R \text{ accepts } f(w), \text{ then } H \text{ accepts } w.
  \item If } R \text{ rejects } f(w), \text{ then } H \text{ rejects } w.
\end{itemize}$

$R$ accepts $f(w)$ \iff $H$ accepts $w$
Compute \( f \)

\[ w \in A \quad \text{iff} \quad f(w) \in B \]

Machine \( H \) = “On input \( w \):
- Transform the input \( w \) into \( f(w) \).
- Run machine \( R \) on \( f(w) \).
- If \( R \) accepts \( f(w) \), then \( H \) accepts \( w \).
- If \( R \) rejects \( f(w) \), then \( H \) rejects \( w \).”
$w \in A \iff f(w) \in B$

Machine $H$

$H = \text{“On input } w:\$
\begin{itemize}
  \item Transform the input $w$ into $f(w)$.
  \item Run machine $R$ on $f(w)$.
  \item If $R$ accepts $f(w)$, then $H$ accepts $w$.
  \item If $R$ rejects $f(w)$, then $H$ rejects $w$.
\end{itemize}

$R$ accepts $f(w)$ iff $f(w) \in B$ iff $w \in A$
\( w \in A \iff f(w) \in B \)

\[ L(H) = A \]

\( H = \) “On input \( w \):

- Transform the input \( w \) into \( f(w) \).
- Run machine \( R \) on \( f(w) \).
- If \( R \) accepts \( f(w) \), then \( H \) accepts \( w \).
- If \( R \) rejects \( f(w) \), then \( H \) rejects \( w \).”
$w \in A \iff f(w) \in B$

$H = \text{"On input $w$:}
\begin{itemize}
  \item Transform the input $w$ into $f(w)$.
  \item Run machine $R$ on $f(w)$.
  \item If $R$ accepts $f(w)$, then $H$ accepts $w$.
  \item If $R$ rejects $f(w)$, then $H$ rejects $w$.
\end{itemize}$

\[\mathcal{L}(H) = A\]
A Problem

• Recall: $f$ is a reduction from $A$ to $B$ iff

$$w \in A \iff f(w) \in B$$

• Under this definition, *any* language $A$ reduces to *any* language $B$ unless $B = \emptyset$ or $\Sigma^*$.

• Since $B \neq \emptyset$ and $B \neq \Sigma^*$, there is some $w_{yes} \in B$ and some $w_{no} \notin B$.

• Define $f : \Sigma_1^* \to \Sigma_2^*$ as follows:

$$f(w) = \begin{cases} 
   w_{yes} & \text{if } w \in A \\
   w_{no} & \text{if } w \notin A
\end{cases}$$

• Then $f$ is a reduction from $A$ to $B$. 
A Problem

- Example: let's reduce $L_D$ to $0^*1^*$.
- Take $w_{yes} = 01$, $w_{no} = 10$.
- Then $f(w)$ is defined as

$$f(w) = \begin{cases} 
01 & \text{if } w \in L_D \\
10 & \text{if } w \notin L_D 
\end{cases}$$

- There is no TM that can actually evaluate the function $f(w)$ on all inputs, since no TM can decide whether or not $w \in L_D$. 
Example: let's reduce $L_D$ to 0.

Take $w_{\text{yes}} = 01$, $w_{\text{no}} = 10$.

Then $f(w)$ is defined as:

$\begin{align*}
01 & \quad \text{if } w \in L_D \\
10 & \quad \text{if } w \notin L_D
\end{align*}$

That's bad!

There is no TM that can actually evaluate the function $f(w)$ on all inputs, since no TM can decide whether or not $w \in L_D$. 
Computable Functions

- This general reduction is mathematically well-defined, but might be impossible to actually compute!
- To fix our definition, we need to introduce the idea of a computable function.
- A function \( f : \Sigma_1^* \rightarrow \Sigma_2^* \) is called a **computable function** if there is some TM \( M \) with the following behavior:

  “On input \( w \):
  
  Compute \( f(w) \) and write it on the tape.
  
  Move the tape head to the start of \( f(w) \).
  
  Halt.”
Computable Functions

\[ f(1^n) = 1^{3n+1} \]
Computable Functions

\[ f(1^n) = 1^{3n+1} \]
Computable Functions

\[ f(w) = \begin{cases} 1^{mn} & \text{if } w = 1^{n \times 1^m} \\ \varepsilon & \text{otherwise} \end{cases} \]
Computable Functions

\[ f(w) = \begin{cases} 
1^{mn} & \text{if } w = 1^n \times 1^m \\
\varepsilon & \text{otherwise}
\end{cases} \]
Computable Functions

\[ f(\langle M \rangle) = \langle M, \langle M \rangle \rangle \]
Computable Functions

\[ f(\langle M \rangle) = \langle M, \langle M \rangle \rangle \]
Mapping Reductions

• A function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ is called a \textbf{mapping reduction} from $A$ to $B$ iff
  • For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$.
  • $f$ is a computable function.
• Intuitively, a mapping reduction from $A$ to $B$ says that a computer can transform any instance of $A$ into an instance of $B$ such that the answer to $B$ is the answer to $A$. 
Mapping Reducibility

• If there is a mapping reduction from language $A$ to language $B$, we say that language $A$ is **mapping reducible** to language $B$.

• Notation: $A \leq_{m} B$ iff language $A$ is mapping reducible to language $B$.

• Note that we reduce **languages**, not **machines**.
$A \leq_M B$

Machine $H$:

- On input $w$:
  - Compute $f(w)$.
  - Run machine $R$ on $f(w)$.
  - If $R$ accepts $f(w)$, then $H$ accepts $w$.
  - If $R$ rejects $f(w)$, then $H$ rejects $w$.
\( A \leq^M_B \)

Machine \( H \)

- Compute \( f \)
- Run machine \( R \) on \( f(w) \)
- If \( R \) accepts \( f(w) \), then \( H \) accepts \( w \).
- If \( R \) rejects \( f(w) \), then \( H \) rejects \( w \).

If \( R \) is a decider for \( B \), then \( H \) is a decider for \( A \).
$A \leq^M B$

$H = "$On input $w$:
  - Compute $f(w)$.
  - Run machine $R$ on $f(w)$.
  - If $R$ accepts $f(w)$, then $H$ accepts $w$.
  - If $R$ rejects $f(w)$, then $H$ rejects $w$.$"

If $R$ is a decider for $B$, then $H$ is a decider for $A$.

If $R$ is a recognizer for $B$, then $H$ is a recognizer for $A$. 
$A \leq_M B$

Machine $H$

$H = \text{“On input } w: \!
\begin{align*}
&\cdot \text{ Compute } f(w). \\
&\cdot \text{ Run machine } R \text{ on } f(w). \\
&\cdot \text{ If } R \text{ accepts } f(w), \text{ then } H \text{ accepts } w. \\
&\cdot \text{ If } R \text{ rejects } f(w), \text{ then } H \text{ rejects } w.\!
\end{align*}$

If $R$ is a decider for $B$, then $H$ is a decider for $A$.

If $R$ is a recognizer for $B$, then $H$ is a recognizer for $A$.

If $R$ is a co-recognizer for $B$, then $H$ is a co-recognizer for $A$. 
H = “On input w:
- Compute $f(w)$.
- Run machine $R$ on $f(w)$.
- If $R$ accepts $f(w)$, then $H$ accepts $w$.
- If $R$ rejects $f(w)$, then $H$ rejects $w$.”

If $R$ is a decider for $B$, then $H$ is a decider for $A$.

If $R$ is a recognizer for $B$, then $H$ is a recognizer for $A$.

If $R$ is a co-recognizer for $B$, then $H$ is a co-recognizer for $A$. 
Why Mapping Reducibility Matters

- **Theorem**: If $B \in \mathbb{R}$ and $A \leq_{M} B$, then $A \in \mathbb{R}$.

- **Theorem**: If $B \in \text{RE}$ and $A \leq_{M} B$, then $A \in \text{RE}$.

- **Theorem**: If $B \in \text{co-RE}$ and $A \leq_{M} B$, then $A \in \text{co-RE}$.

- *Intuitively*: $A \leq_{M} B$ means “$A$ is not harder than $B$.”
Why Mapping Reducibility Matters

- **Theorem**: If $A \notin R$ and $A \leq_M B$, then $B \notin R$.

- **Theorem**: If $A \notin \text{RE}$ and $A \leq_M B$, then $B \notin \text{RE}$.

- **Theorem**: If $A \notin \text{co-RE}$ and $A \leq_M B$, then $B \notin \text{co-RE}$.

- **Intuitively**: $A \leq_M B$ means “$B$ is at least as hard as $A$. “
Why Mapping Reducibility Matters

If this one is "easy" \((R, \text{RE}, \text{co-RE})\)...

\[ A \leq_M B \]

... then this one is "easy" \((R, \text{RE}, \text{co-RE})\) too.
Why Mapping Reducibility Matters

$A \leq_M B$

If this one is "hard"
(not $R$, not $RE$, or not $co\text{-}RE$)...

... then this one is "hard" (not $R$, not $RE$, or not $co\text{-}RE$) too.