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 \( \mathcal{R} \) is the set of all decidable languages.
  \[ L \in \mathcal{R} \iff L \text{ is decidable} \]
The Limits of Computability

- Regular Languages
- CFLs
- RE
- R
- All Languages

HALT
\overline{\text{A}_{\text{TM}}}
\overline{L_D}
\text{A}_{\text{TM}}
L_D
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

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

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

- $ADD$
- $0^*1^*$
- $HALT$
- $\overline{L_D}$
- $A_{TM}$

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

- A language \( L \) is in \textbf{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 \textbf{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}.
\textbf{RE and co-RE}

- Intuitively, \textbf{RE} consists of all problems where a TM can exhaustively search for \textbf{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, \textbf{co-RE} consists of all problems where a TM can exhaustively search for a \textbf{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

- $L_D$ 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 $M$ accepts $w$ iff $w \in L$.

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

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

\[
\text{\( M' \) 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}. 
\]

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 $w \notin L$.
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?

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 $w \notin L$.
Which Picture is Correct?

CO-RE

- $L_D$
- $\overline{HALT}$
- $A_{TM}$

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

R

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

RE

- $\overline{L_D}$
- $HALT$
- $A_{TM}$

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 $L \in \text{R}$.

- **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 $L$:

$$D = "\text{On input } w: \text{ Run } M \text{ on } w \text{ and } \overline{M} \text{ on } w \text{ in parallel. If } M \text{ accepts } w, \text{ accept. If } \overline{M} \text{ rejects } w, \text{ reject.}"$$
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 $\mathbf{R}$, $\mathbf{RE}$, or $\mathbf{co-RE}$.

• **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
\[ L = \{ \langle M \rangle \mid M \text{ is a TM that accepts } \varepsilon \} \]

\[ H = \text{"On input } \langle M \rangle \text{:} \]
\[ \begin{align*}
&\cdot \text{ Construct the string } \langle M, \varepsilon \rangle. \\
&\cdot \text{ Run } R \text{ on } \langle M, \varepsilon \rangle. \\
&\cdot \text{ If } R \text{ accepts } \langle M, \varepsilon \rangle, \text{ then } H \text{ accepts } \langle M, \varepsilon \rangle. \\
&\cdot \text{ If } R \text{ rejects } \langle M, \varepsilon \rangle, \text{ then } H \text{ rejects } \langle M, \varepsilon \rangle. 
\end{align*} \]
From $\overline{A_{TM}}$ to $L_D$

$H$ = “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 \textit{HALT} to $A_{TM}$

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

- Build $M$ into $M'$ so $M'$ loops when $M$ rejects.
- Run $D$ on $\langle M', w \rangle$.
- If $D$ accepts $\langle M', w \rangle$, then $H$ accepts $\langle M, w \rangle$.
- If $D$ rejects $\langle M', w \rangle$, then $H$ rejects $\langle M, w \rangle.$
The General Pattern

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

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

Is $\mathcal{L}(D) = \Sigma^*$?

Can be converted to

Can be used to solve

Is $\mathcal{L}(D_1)$ equal to $\Sigma^* - \mathcal{L}(D_2)$?

Problem $A$

Problem $B$
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

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

Problem $A$

Problem $B$
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 $A$.

\[ A_{TM} \quad \text{Can be converted to} \quad \text{HALT} \]

Problem $A$  |  Can be used to solve  |  Problem $B$
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** $f(w) \in B$

- Every $w \in A$ maps to some $f(w) \in B$.
- Every $w \notin A$ maps to some $f(w) \notin B$.
- $f$ 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?
$w \in A$ iff $f(w) \in B$

$H$ accepts $w$ iff:
- 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 $w$.

$R$ accepts $f(w)$ iff $f(w) \in B$ iff $w \in A$
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 $f : \Sigma_1^* \rightarrow \Sigma_2^*$ 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 $w \in L_D$. 
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 $f : \Sigma_1^* \to \Sigma_2^*$ 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.”
Mapping Reductions

- A function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ is called a 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*. 
$A \leq^M_B$

$H = \text{"On input } w:\”$

- 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

\[ A \leq_{M} B \]

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

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

If this one is "hard" (not R, not RE, or not co-RE)...

\[ A \leq_M B \]

... then this one is "hard" (not R, not RE, or not co-RE) too.