co-RE and Beyond
Friday Four Square!
Today at 4:15PM, Outside Gates
Announcements

• Problem Set 7 due right now.
  • With a late day, due this Monday at 2:15PM.
• Problem Set 8 out, due Friday, November 30.
  • Explore properties of \( R, \text{RE}, \) and co-\( \text{RE}. \)
  • Play around with mapping reductions.
  • Find problems far beyond the realm of computers.
  • **No checkpoint**, even though the syllabus says there is one.
• Most (but not all) Problem Set 6 graded; will be returned at end of lecture).
Recap From Last Time
Mapping Reducibility

- A **mapping reduction** from $A$ to $B$ is a function $f$ such that
  - $f$ is computable, and
  - For any $w$, $w \in A$ iff $f(w) \in B$.
- If there is a mapping reduction from $A$ to $B$, we say that $A$ is **mapping reducible** to $B$.
- Notation: $A \leq_M B$ iff $A$ is mapping reducible to $B$. 
Why Mapping Reducibility Matters

If this one is "easy" (R or RE)...

\[ A \leq_M B \]

... then this one is "easy" (R or RE) too.
Why Mapping Reducibility Matters

If this one is “hard” (not R or not RE)…

\[ A \leq_{M} B \]

… then this one is “hard” (not R or not RE) too.
Machine for

Compute $f$

$f(w)$

Machine for $B$

Machine $M$

$H = \text{"On input } w:\$

Compute $f(w)$.

Run $M$ on $f(w)$.

If $M$ accepts $f(w)$, accept $w$.

If $M$ rejects $f(w)$, reject $w$."

$H$ accepts $w$ iff $M$ accepts $f(w)$ iff $f(w) \in B$ iff $w \in A$
More Unsolvable Problems
A More Elaborate Reduction

• Since \( \text{HALT} \notin \mathbb{R} \), there is no algorithm for determining whether a TM will halt on some particular input.

• It seems, therefore, that we shouldn't be able to decide whether a TM halts on all possible inputs.

• Consider the language

\[
\text{DECIDER} = \{ \langle M \rangle \mid M \text{ is a decider} \}
\]

• How would we prove that \( \text{DECIDER} \) is, itself, undecidable?
\[ \text{HALT} \leq_{M} \text{DECIDER} \]

- We will prove that \text{DECIDER} is undecidable by reducing \text{HALT} to \text{DECIDER}.

- Want to find a function \( f \) such that
  \[ \langle M, w \rangle \in \text{HALT} \iff f(\langle M, w \rangle) \in \text{DECIDER}. \]

- Assuming that \( f(\langle M, w \rangle) = \langle M' \rangle \) for some TM \( M' \), we have that
  \[ \langle M, w \rangle \in \text{HALT} \iff \langle M' \rangle \in \text{DECIDER}. \]
  
  \( M \) halts on \( w \) \iff \( M' \) is a decider.
  
  \( M \) halts on \( w \) \iff \( M' \) halts on all inputs.
The Reduction

- Find a TM $M'$ such that $M'$ halts on all inputs iff $M$ halts on $w$.
- **Key idea:** Build $M'$ such that running $M'$ on any input runs $M$ on $w$.
- Here is one choice of $M'$:

  $$M' = \text{“On input } x:\text{ }
  \begin{align*}
  \text{Ignore } x. \\
  \text{Run } M \text{ on } w. \\
  \text{If } M \text{ accepts } w, \text{ accept.} \\
  \text{If } M \text{ rejects } w, \text{ reject.”}
  \end{align*}$$

- Notice that $M'$ “amplifies” what $M$ does on $w$:
  - If $M$ halts on $w$, $M'$ halts on every input.
  - If $M$ loops on $w$, $M'$ loops on every input.
DECIDER is Undecidable

Construct $M'$ from $\langle M, w \rangle$

Decider for DECIDER

This is a decider for HALT!
Justifying $M'$

- Notice that our machine $M'$ has the machine $M$ and string $w$ built into it!

- This is different from the machines we have constructed in the past.

- How do we justify that it's possible for some TM to construct a new TM at all?

$M' = \text{“On input } x:\text{ Ignore } x.\text{ Run } M \text{ on } w.\text{ If } M \text{ accepts } w, \text{ accept.}\text{ If } M \text{ rejects } w, \text{ reject.”}$
The Parameterization Theorem

**Theorem**: Let $M$ be a TM of the form

$$M = \text{"On input } \langle x_1, x_2, \ldots, x_n \rangle:\text{ Do something with } x_1, x_2, \ldots, x_n\text{"}$$

and any value $p$ for parameter $x_1$, then a TM can construct the following TM $M'$:

$$M' = \text{"On input } \langle x_2, \ldots, x_n \rangle:\text{ Do something with } p, x_2, \ldots, x_n\text{"}$$
Justifying $M'$

- Consider this machine $X$:

  $$X = \text{"On input } \langle N, z, x \rangle:\text{ Ignore } x.\text{ Run } N \text{ on } z.\text{ If } N \text{ accepts } z, \text{ accept. If } N \text{ rejects } z, \text{ reject."}$$

- Applying the parameterization theorem twice with the values $M$ and $w$ produces the machine $M' = \text{"On input } x:\text{ Ignore } x.\text{ Run } M \text{ on } w.\text{ If } M \text{ accepts } w, \text{ accept. If } M \text{ rejects } w, \text{ reject."}$
The Takeaway Point

• It is possible for a mapping reduction to take in a TM or TM/string pair and construct a new TM with that TM embedded within it.

• The parameterization theorem is just a formal way of justifying this.
**Theorem:** $\text{HALT} \leq^M \text{DECIDER}$.  

**Proof:** We exhibit a mapping reduction from $\text{HALT}$ to $\text{DECIDER}$. For any TM/string pair $\langle M, w \rangle$, let $f(\langle M, w \rangle) = \langle M' \rangle$, where $\langle M' \rangle$ is defined in terms of $M$ and $w$ as follows:

$$M' = \text{"On input } x:\text{ Ignore } x. \text{ Run } M \text{ on } w. \text{ If } M \text{ accepts } w, \text{ accept. If } M \text{ rejects } w, \text{ reject."}$$

By the parameterization theorem, $f$ is a computable function. We further claim that $\langle M, w \rangle \in \text{HALT}$ iff $f(\langle M, w \rangle) \in \text{DECIDER}$. To see this, note that $f(\langle M, w \rangle) = \langle M' \rangle \in \text{DECIDER}$ iff $M'$ halts on all inputs. We claim that $M'$ halts on all inputs iff $M$ halts on $w$. To see this, note that when $M'$ is run on any input, it halts iff $M$ halts on $w$. Thus if $M$ halts on $w$, then $M'$ halts on all inputs, and if $M$ loops on $w$, $M'$ loops on all inputs. Finally, note that $M$ halts on $w$ iff $\langle M, w \rangle \in \text{HALT}$. Thus $\langle M, w \rangle \in \text{HALT}$ iff $f(\langle M, w \rangle) \in \text{DECIDER}$. Therefore, $f$ is a mapping reduction from $\text{HALT}$ to $\text{DECIDER}$, so $\text{HALT} \leq^M \text{DECIDER}$. $\blacksquare$
Other Hard Languages

• We can't tell if a TM accepts a specific string.

• Could we determine whether or not a TM accepts one of many different strings with specific properties?

• For example, could we build a TM that determines whether some other TM accepts a string of all \(1s\)?

• Let \(\text{ONES}_{\text{TM}}\) be the following language:

\[
\text{ONES}_{\text{TM}} = \{ \langle M \rangle \mid M \text{ is a TM that accepts at least one string of the form } 1^n \}
\]

• Is \(\text{ONES}_{\text{TM}} \in \mathbb{R}\)? Is it \(\text{RE}\)?
ONES\textsuperscript{TM}

- Unfortunately, ONES\textsuperscript{TM} is undecidable.
- However, ONES\textsuperscript{TM} is recognizable.
  - Intuition: Nondeterministically guess the string of the form \(1^n\) that \(M\) will accept, then deterministically check that \(M\) accepts it.
- We'll show that ONES\textsuperscript{TM} is undecidable by showing that \(A_{\text{TM}} \leq_M \text{ONES}\).
\[ A_{\text{TM}} \leq_{M} \text{ONES}_{\text{TM}} \]

- As before, let's try to find a function \( f \) such that
  \[ \langle M, w \rangle \in A_{\text{TM}} \iff f(\langle M, w \rangle) \in \text{ONES}_{\text{TM}}. \]

- Let's let \( f(\langle M, w \rangle) = \langle M' \rangle \) for some TM \( M' \). Then we want to pick \( M' \) such that
  \[ \langle M, w \rangle \in A_{\text{TM}} \iff \langle M' \rangle \in \text{ONES}_{\text{TM}} \]
  \[ M \text{ accepts } w \iff M' \text{ accepts } 1^n \text{ for some } n \]
The Reduction

- Goal: construct $M'$ so $M'$ accepts $1^n$ for some $n$ iff $M$ accepts $w$.
- Here is one possible option:
  \[ M' = \text{"On input } x:\]
  
  Ignore $x$.
  
  Run $M$ on $w$.
  
  If $M$ accepts $w$, accept $x$.
  
  If $M$ rejects $w$, reject $x$.”
- As with before, we can justify the construction of $M'$ using the parameterization theorem.
- If $M$ accepts $w$, then $M'$ accepts all strings, including $1^n$ for any $n$.
- If $M$ does not accept $w$, then $M'$ does not accept any strings, so it certainly does not accept any strings of the form $1^n$. 
Theorem: $A_{TM} \leq_M ONES_{TM}$.

Proof: We exhibit a mapping reduction from $A_{TM}$ to $ONES_{TM}$. For any TM/string pair $\langle M, w \rangle$, let $f(\langle M, w \rangle) = \langle M' \rangle$, where $M'$ is defined in terms of $M$ and $w$ as follows:

$$M' = "On input x:
    Ignore x.
    Run M on w.
    If M accepts w, accept x.
    If M rejects w, reject x."
$$

By the parameterization theorem, $f$ is a computable function. We further claim that $\langle M, w \rangle \in A_{TM}$ iff $f(\langle M, w \rangle) \in ONES_{TM}$. To see this, note that $f(\langle M, w \rangle) = \langle M' \rangle \in ONES_{TM}$ iff $M'$ accepts at least one string of the form $1^n$. We claim that $M'$ accepts at least one string of the form $1^n$ iff $M$ accepts $w$. To see this, note that if $M$ accepts $w$, then $M'$ accepts 1, and if $M$ does not accept $w$, then $M'$ rejects all strings, including all strings of the form $1^n$. Finally, $M$ accepts $w$ iff $\langle M, w \rangle \in A_{TM}$. Thus $\langle M, w \rangle \in A_{TM}$ iff $f(\langle M, w \rangle) \in ONES_{TM}$. Consequently, $f$ is a mapping reduction from $A_{TM}$ to $ONES_{TM}$, so $A_{TM} \leq_M ONES_{TM}$ as required. ■
A Slightly Modified Question

• We cannot determine whether or not a TM will accept at least one string of all 1s.
• Can we determine whether a TM only accepts strings of all 1s?
• In other words, for a TM $M$, is $L(M) \subseteq 1^*$?
• Let $\text{ONLYONES}_{\text{TM}}$ be the language

$$\text{ONLYONES}_{\text{TM}} = \{ \langle M \rangle \mid L(M) \subseteq 1^* \}$$

• Is $\text{ONLYONES}_{\text{TM}} \in \mathcal{R}$? How about $\mathcal{RE}$?
\[
\text{ONLYONES}_{\text{TM}} \notin \text{RE}
\]

- It turns out that the language \(\text{ONLYONES}_{\text{TM}}\) is unrecognizable.
- We can prove this by reducing \(L_D\) to \(\text{ONLYONES}_{\text{TM}}\).
- If \(L_D \leq_M \text{ONLYONES}_{\text{TM}}\), then we have that \(\text{ONLYONES}_{\text{TM}} \notin \text{RE}\).
$L_D \leq_M \text{ONLYONES}_\text{TM}$

- We want to find a computable function $f$ such that
  \[ \langle M \rangle \in L_D \iff f(\langle M \rangle) \in \text{ONLYONES}_\text{TM}. \]
- We want to set $f(\langle M \rangle) = \langle M' \rangle$ for some suitable choice of $M'$. This means
  \[ \langle M \rangle \in L_D \iff \langle M' \rangle \in \text{ONLYONES}_\text{TM} \quad \langle M \rangle \notin L(M) \iff L(M') \subseteq 1^* \]
- How would we pick our machine $M'$?
One Possible Reduction

- We want to build $M'$ from $M$ such that $⟨M⟩ ∉ \mathcal{L}(M)$ iff $\mathcal{L}(M') ⊆ 1^*$.

- In other words, we construct $M'$ such that
  - If $⟨M⟩ ∈ \mathcal{L}(M)$, then $\mathcal{L}(M')$ is not a subset of $1^*$.
  - If $⟨M⟩ ∉ \mathcal{L}(M)$, then $\mathcal{L}(M')$ is a subset of $1^*$.

- One option: Come up with some languages with these properties, then construct our machine $M'$ such that its language changes based on whether $⟨M⟩ ∈ \mathcal{L}(M)$.
  - If $⟨M⟩ ∈ \mathcal{L}(M)$, then $\mathcal{L}(M') = \Sigma^*$, which isn't a subset of $1^*$.
  - If $⟨M⟩ ∉ \mathcal{L}(M)$, then $\mathcal{L}(M') = \emptyset$, which is a subset of $1^*$. 

One Possible Reduction

• We want
  • If $\langle M \rangle \in \mathcal{L}(M)$, then $\mathcal{L}(M') = \Sigma^*$
  • If $\langle M \rangle \notin \mathcal{L}(M)$, then $\mathcal{L}(M') = \emptyset$
• Here is one possible $M'$ that does this:
  
  $M' = \text{"On input } x:\$
  
  Ignore $x$.
  
  Run $M$ on $\langle M \rangle$.

  If $M$ accepts $\langle M \rangle$, accept $x$.
  
  If $M$ rejects $\langle M \rangle$, reject $x$.\"
Theorem: $L_D \leq_M \text{ONLYONES}_\text{TM}$.

Proof: We exhibit a mapping reduction from $L_D$ to $\text{ONLYONES}_\text{TM}$.

For any TM $M$, let $f(\langle M \rangle) = \langle M' \rangle$, where $M'$ is defined in terms of $M$ as follows:

$$M' = \text{"On input } x:\text{ Ignore } x.\text{ Run } M\text{ on } \langle M \rangle.\text{ If } M\text{ accepts } \langle M \rangle, \text{ accept } x.\text{ If } M\text{ rejects } \langle M \rangle, \text{ reject } x."$$

By the parameterization theorem, $f$ is a computable function. We further claim that $\langle M \rangle \in L_D$ iff $f(\langle M \rangle) \in \text{ONLYONES}_\text{TM}$. To see this, note that $f(\langle M \rangle) = \langle M' \rangle \in \text{ONLYONES}_\text{TM}$ iff $\mathcal{L}(M') \subseteq 1^*$. We claim that $\mathcal{L}(M') \subseteq 1^*$ iff $M$ does not accept $\langle M \rangle$. To see this, note that if $M$ does not accept $\langle M \rangle$, then $M'$ never accepts any strings, so $\mathcal{L}(M') = \emptyset \subseteq 1^*$. Otherwise, if $M$ accepts $\langle M \rangle$, then $M'$ accepts all strings, so $\mathcal{L}(M) = \Sigma^*$, which is not a subset of $1^*$. Finally, $M$ does not accept $\langle M \rangle$ iff $\langle M \rangle \in L_D$. Thus $\langle M \rangle \in L_D$ iff $f(\langle M \rangle) \in \text{ONLYONES}_\text{TM}$. Consequently, $f$ is a mapping reduction from $L_D$ to $\text{ONLYONES}_\text{TM}$, so $L_D \leq_M \text{ONLYONES}_\text{TM}$ as required. ■
• Although $\text{ONLYONES}^\text{TM}$ is not RE, its complement $(\text{ONLYONES}^\text{TM})$ is RE:

\[
\{ \langle M \rangle \mid \mathcal{L}(M) \text{ is not a subset of } 1^* \}
\]

• Intuition: Can nondeterministically guess a string in $\mathcal{L}(M)$ that is not of the form $1^n$, then check that $M$ accepts it.
The Limits of Computability

- Regular Languages
- DCFLs
- CFLs
- $R$
- $RE$
- $L_D$
- $HALT$
- $A_{TM}$
- $\overline{HALT}$
- $\overline{A_{TM}}$
- $ONES_{TM}$
- $ONLYONES_{TM}$

All Languages
RE and co-RE

- The class **RE** is the set of languages that are recognized by a TM.
- The class **co-RE** is the set of languages whose *complements* are recognized by a TM.
- In other words:
  \[ L \in \text{co-RE} \iff \overline{L} \in \text{RE} \]
  \[ \overline{L} \in \text{co-RE} \iff L \in \text{RE} \]
- Languages in co-RE are called **co-recognizable**. Languages not in co-RE are called **co-unrecognizable**.
Intuiting **RE** and **co-RE**

- A language $L$ is in **RE** iff there is a recognizer for it.
  - If $w \in L$, the recognizer accepts.
  - If $w \notin L$, the recognizer does not accept.
- A language $L$ is in **co-RE** iff there is a **refuter** for it.
  - If $w \notin L$, the refuter rejects.
  - If $w \in L$, the refuter does not reject.
RE, and co-RE

- **RE** and **co-RE** are fundamental classes of problems.
  - **RE** is the class of problems where a computer can always verify “yes” instances.
  - **co-RE** is the class of problems where a computer can always refute “no” instances.
- **RE** and **co-RE** are, in a sense, the weakest possible conditions for which a problem can be approached by computers.
R, RE, and co-RE

• Recall:

  If $L \in \text{RE}$ and $\overline{L} \in \text{RE}$, then $L \in \text{R}$

• Rewritten in terms of co-RE:

  If $L \in \text{RE}$ and $L \in \text{co-RE}$, then $L \in \text{R}$

• In other words:

  $\text{RE} \cap \text{co-RE} \subseteq \text{R}$

• We also know that $\text{R} \subseteq \text{RE}$ and $\text{R} \subseteq \text{co-RE}$, so

  $\text{R} = \text{RE} \cap \text{co-RE}$
$L_D$ Revisited

- The diagonalization language $L_D$ is the language
  $\quad L_D = \{\langle M \rangle \mid M \text{ is a TM and } M \notin \mathcal{L}(M)\}$
- As we saw before, $L_D \notin \text{RE}$.
- So where is $L_D$? Is it in $L_D \in \text{co-RE}$? Or is it someplace else?
To see whether $L_D \in \text{co-RE}$, we will see whether $\overline{L}_D \in \text{RE}$.

The language $\overline{L}_D$ is the language

$$\overline{L}_D = \{\langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \in \mathcal{L}(M)\}$$

Two questions:

- What is this language?
- Is this language RE?
This language is \( L_D \).
\( L_D \in \text{co-RE} \)

- Here's an TM for \( \overline{L}_D \):

\[
R = \text{"On input } \langle M \rangle: \\
\text{Run } M \text{ on } \langle M \rangle. \\
\text{If } M \text{ accepts } \langle M \rangle, \text{ accept.} \\
\text{If } M \text{ rejects } \langle M \rangle, \text{ reject."}
\]

- Then \( R \) accepts \( \langle M \rangle \) iff \( \langle M \rangle \in \mathcal{L}(M) \) iff \( \langle M \rangle \in \overline{L}_D \), so \( \mathcal{L}(R) = \overline{L}_D \).
The Limits of Computability

- \( \overline{\text{ONES}}^\text{TM} \)
- \( \overline{\text{ONLYONES}}^\text{TM} \)
- \( \overline{\text{HALT}} \)
- \( L_D \)
- \( A^\text{TM} \)
- \( 0^*1^* \)
- \( \text{DOGWALK} \)
- \( \text{ADD} \)
- \( \overline{L_D} \)
- \( \overline{A^\text{TM}} \)
- \( \overline{\overline{\text{ONES}}}^\text{TM} \)
- \( \overline{\text{ONLYONES}}^\text{TM} \)

\( \text{R} \), \( \text{RE} \), \( \text{co-RE} \)

All Languages
Theorem: If \( A \leq_{M} B \), then \( \overline{A} \leq_{M} \overline{B} \).

Proof: Suppose that \( A \leq_{M} B \). Then there exists a computable function \( f \) such that \( w \in A \) iff \( f(w) \in B \). Note that \( w \in A \) iff \( w \notin \overline{A} \) and \( f(w) \in B \) iff \( f(w) \notin \overline{B} \). Consequently, we have that \( w \notin \overline{A} \) iff \( f(w) \notin \overline{B} \). Thus \( w \in \overline{A} \) iff \( f(w) \in \overline{B} \). Since \( f \) is computable, \( \overline{A} \leq_{M} \overline{B} \). \( \square \)
co-RE Reductions

- **Corollary:** If $A \leq^M B$ and $B \in \text{co-RE}$, then $A \in \text{co-RE}$.

  *Proof:* Since $A \leq^M B$, $\overline{A} \leq^M \overline{B}$. Since $B \in \text{co-RE}$, $\overline{B} \in \text{RE}$. Thus $\overline{A} \in \text{RE}$, so $A \in \text{co-RE}$. ■

- **Corollary:** If $A \leq^M B$ and $A \notin \text{co-RE}$, then $B \notin \text{co-RE}$.

  *Proof:* Take the contrapositive of the above. ■
Why Mapping Reducibility Matters

If this one is “easy” (R or RE or co-RE) … then this one is “easy” (R or RE or co-RE) too.
Why Mapping Reducibility Matters

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

\[ A \leq_M B \]

... then this one is "hard" (not R or not RE or not co-RE) too.
The Limits of Computability

Is there anything out here?

All Languages
**RE ∪ co-RE is Not Everything**

- Using the same reasoning as the first day of lecture, we can show that there must be problems that are neither RE nor co-RE.
- There are more sets of strings than TMs.
- There are more sets of strings than twice the number of TMs.
- What do these languages look like?
An Extremely Hard Problem

- Recall: All regular languages are also $\text{RE}$. 
- This means that some TMs accept regular languages and some TMs do not. 
- Let $\text{REGULAR}_{\text{TM}}$ be the language of all TM descriptions that accept regular languages:

$$\text{REGULAR}_{\text{TM}} = \{ \langle M \rangle \mid \mathcal{L}(M) \text{ is regular} \}$$

- Is $\text{REGULAR}_{\text{TM}} \in \text{R}$? How about $\text{RE}$?
REGULAR$_{\text{TM}} \notin \text{RE}$

- It turns out that REGULAR$_{\text{TM}}$ is unrecognizable, meaning that there is no computer program that can even verify that another TM's language is regular!
- To do this, we'll do another reduction from $L_D$ and prove that $L_D \leq_M \text{REGULAR}_{\text{TM}}$. 
We want to find a computable function $f$ such that

$$\langle M \rangle \in L_D \iff f(\langle M \rangle) \in \text{REGULAR}_\text{TM}.$$  

We need to choose $M'$ such that $f(\langle M \rangle) = \langle M' \rangle$ for some TM $M'$. Then

$$\langle M \rangle \in L_D \iff f(\langle M \rangle) \in \text{REGULAR}_\text{TM}$$

$$\langle M \rangle \in L_D \iff \langle M' \rangle \in \text{REGULAR}_\text{TM}$$

$$\langle M \rangle \notin \mathcal{L}(M) \iff \mathcal{L}(M') \text{ is regular.}$$
\[ L_D \leq_M \text{REGULAR}_\text{TM} \]

- We want to construct some \( M' \) out of \( M \) such that
  - If \( \langle M \rangle \in \mathcal{L}(M) \), then \( \mathcal{L}(M') \) is not regular.
  - If \( \langle M \rangle \notin \mathcal{L}(M) \), then \( \mathcal{L}(M') \) is regular.

- One option: choose two languages, one regular and one nonregular, then construct \( M' \) so its language switches from regular to nonregular based on whether \( \langle M \rangle \notin \mathcal{L}(M) \).
  - If \( \langle M \rangle \in \mathcal{L}(M) \), then \( \mathcal{L}(M') = \{ 0^n1^n \mid n \in \mathbb{N} \} \)
  - If \( \langle M \rangle \notin \mathcal{L}(M) \), then \( \mathcal{L}(M') = \emptyset \)
The Reduction

• We want to build $M'$ from $M$ such that
  • If $\langle M \rangle \in \mathcal{L}(M)$, then $\mathcal{L}(M') = \{ \text{0}^n\text{1}^n \mid n \in \mathbb{N} \}$
  • If $\langle M \rangle \notin \mathcal{L}(M)$, then $\mathcal{L}(M') = \emptyset$
• Here is one way to do this:

  $M' = \text{“On input } x:\n    \text{If } x \text{ does not have the form } \text{0}^n\text{1}^n, \text{ reject.}n\text{Run } M \text{ on } \langle M \rangle.\n    \text{If } M \text{ accepts, accept } x.\n    \text{If } M \text{ rejects, reject } x.”$
Theorem: $L_D \leq_M \text{REGULAR}_{TM}$.

Proof: We exhibit a mapping reduction from $L_D$ to $\text{REGULAR}_{TM}$. For any TM $M$, let $f(\langle M \rangle) = \langle M' \rangle$, where $M'$ is defined in terms of $M$ as follows:

$$M' = \text{"On input } x:\text{"}$$

- If $x$ does not have the form $0^n1^n$, reject $x$.
- Run $M$ on $\langle M \rangle$.
- If $M$ accepts $\langle M \rangle$, accept $x$.
- If $M$ rejects $\langle M \rangle$, reject $x$.

By the parameterization theorem, $f$ is a computable function. We further claim that $\langle M \rangle \in L_D$ iff $f(\langle M \rangle) \in \text{REGULAR}_{TM}$. To see this, note that $f(\langle M \rangle) = \langle M' \rangle \in \text{REGULAR}_{TM}$ iff $\mathcal{L}(M')$ is regular. We claim that $\mathcal{L}(M')$ is regular iff $\langle M \rangle \notin \mathcal{L}(M)$. To see this, note that if $\langle M \rangle \notin \mathcal{L}(M)$, then $M'$ never accepts any strings. Thus $\mathcal{L}(M') = \emptyset$, which is regular. Otherwise, if $\langle M \rangle \in \mathcal{L}(M)$, then $M'$ accepts all strings of the form $0^n1^n$, so we have that $\mathcal{L}(M) = \{0^n1^n \mid n \in \mathbb{N}\}$, which is not regular. Finally, $\langle M \rangle \notin \mathcal{L}(\langle M \rangle)$ iff $\langle M \rangle \in L_D$. Thus $\langle M \rangle \in L_D$ iff $f(\langle M \rangle) \in \text{REGULAR}_{TM'}$, so $f$ is a mapping reduction from $L_D$ to $\text{REGULAR}_{TM}$. Therefore, $L_D \leq_M \text{REGULAR}_{TM}$. ■
REGULAR\textsuperscript{TM} ∉ \text{co-RE}

- Not only is REGULAR\textsuperscript{TM} ∉ \text{RE}, but REGULAR\textsuperscript{TM} ∉ \text{co-RE}.

- Before proving this, take a minute to think about just how ridiculously hard this problem is.
  
  - No computer can confirm that an arbitrary TM has a regular language.
  
  - No computer can confirm that an arbitrary TM has a nonregular language.

  - This is vastly beyond the limits of what computers could ever hope to solve.
\[ \overline{L}_D \leq_M \text{REGULAR}_{TM} \]

- To prove that \text{REGULAR}_{TM} is not co-\text{RE}, we will prove that \( \overline{L}_D \leq_M \text{REGULAR}_{TM} \).

- Since \( \overline{L}_D \) is not co-\text{RE}, this proves that \text{REGULAR}_{TM} is not co-\text{RE} either.

- Goal: Find a function \( f \) such that
  \[ \langle M \rangle \in \overline{L}_D \iff f(\langle M \rangle) \in \text{REGULAR}_{TM} \]

- Let \( f(\langle M \rangle) = \langle M' \rangle \) for some TM \( M' \). Then we want
  \[ \langle M \rangle \in \overline{L}_D \iff \langle M' \rangle \in \text{REGULAR}_{TM} \]
  \[ \langle M \rangle \in \mathcal{L}(M) \iff \mathcal{L}(M') \text{ is regular} \]
\( \overline{L_D} \leq_M \text{REGULAR}_TM \)

- We want to construct some \( M' \) out of \( M \) such that
  - If \( \langle M \rangle \in \mathcal{L}(M) \), then \( \mathcal{L}(M') \) is regular.
  - If \( \langle M \rangle \notin \mathcal{L}(M) \), then \( \mathcal{L}(M') \) is not regular.
- One option: choose two languages, one regular and one nonregular, then construct \( M' \) so its language switches from regular to nonregular based on whether \( \langle M \rangle \in \mathcal{L}(M) \).
  - If \( \langle M \rangle \in \mathcal{L}(M) \), then \( \mathcal{L}(M') = \Sigma^* \).
  - If \( \langle M \rangle \notin \mathcal{L}(M) \), then \( \mathcal{L}(M') = \{0^n1^n \mid n \in \mathbb{N}\} \)
\[ \overline{L}_D \leq_M \text{REGULAR}_{TM} \]

- We want to build \( M' \) from \( M \) such that
  - If \( \langle M \rangle \in \mathcal{L}(M) \), then \( \mathcal{L}(M') = \Sigma^* \)
  - If \( \langle M \rangle \notin \mathcal{L}(M) \), then \( \mathcal{L}(M') = \{ 0^n1^n \mid n \in \mathbb{N} \} \)
- Here is one way to do this:
  \[ M' = \text{"On input } x: \]
  - If \( x \) has the form \( 0^n1^n \), accept.
  - Run \( M \) on \( \langle M \rangle \).
  - If \( M \) accepts, accept \( x \).
  - If \( M \) rejects, reject \( x \)."
Theorem: $\overline{L}_D \leq_M \text{REGULAR}_{\text{TM}}$.

Proof: We exhibit a mapping reduction from $\overline{L}_D$ to $\text{REGULAR}_{\text{TM}}$. For any TM $M$, let $f(\langle M \rangle) = \langle M' \rangle$, where $M'$ is defined in terms of $M$ as follows:

$$M' = \text{"On input } x:\text{ }
\begin{align*}
\text{If } x \text{ has the form } 0^n1^n, & \text{ accept } x. \\
\text{Run } M \text{ on } \langle M \rangle. \\
\text{If } M \text{ accepts } \langle M \rangle, & \text{ accept } x. \\
\text{If } M \text{ rejects } \langle M \rangle, & \text{ reject } x.
\end{align*}$$

By the parameterization theorem, $f$ is a computable function. We further claim that $\langle M \rangle \in \overline{L}_D$ iff $f(\langle M \rangle) \in \text{REGULAR}_{\text{TM}}$. To see this, note that $f(\langle M \rangle) = \langle M' \rangle \in \text{REGULAR}_{\text{TM}}$ iff $\mathcal{L}(M')$ is regular. We claim that $\mathcal{L}(M')$ is regular iff $\langle M \rangle \in \mathcal{A}(M)$. To see this, note that if $\langle M \rangle \in \mathcal{A}(M)$, then $M'$ accepts all strings, either because that string is of the form $0^n1^n$ or because $M$ eventually accepts $\langle M \rangle$. Thus $\mathcal{A}(M') = \Sigma^*$, which is regular. Otherwise, if $\langle M \rangle \notin \mathcal{A}(M)$, then $M'$ only accepts strings of the form $0^n1^n$, so $\mathcal{A}(M) = \{ 0^n1^n \mid n \in \mathbb{N} \}$, which is not regular. Finally, $\langle M \rangle \in \mathcal{A}(\langle M \rangle)$ iff $\langle M \rangle \in \overline{L}_D$. Thus $\langle M \rangle \in \overline{L}_D$ iff $f(\langle M \rangle) \in \text{REGULAR}_{\text{TM}}$, so $f$ is a mapping reduction from $\overline{L}_D$ to $\text{REGULAR}_{\text{TM}}$. Therefore, $\overline{L}_D \leq_M \text{REGULAR}_{\text{TM}}$. ■
The Limits of Computability

- \(\text{REGULAR}_{TM}\)
- \(\text{HALT}\)
- \(\overline{\text{ONES}}_{TM}\)
- \(\text{ONLYONES}_{TM}\)
- \(\overline{L}_{D}\)
- \(\overline{A}_{TM}\)
- \(0^*1^*\)
- \(\text{DOGWALK}\)
- \(\text{ADD}\)
- \(\overline{\text{L}}_{D}\)
- \(A_{TM}\)
- \(\text{ONES}_{TM}\)
- \(\text{ONLYONES}_{TM}\)

All Languages
Beyond \textit{RE} and co-\textit{RE}

- The most famous problem that is neither \textit{RE} nor co-\textit{RE} is the TM equality problem:
  \[
  \text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2) \}
  \]

- This is why we have to write testing code; there's no way to have a computer prove or disprove that two programs always have the same output.

- This is related to Q6.ii from Problem Set 7.
Why All This Matters
The Limits of Computability

\[ \text{RE} \]

\[ \text{co-RE} \]

\[ \text{ONES}_{TM} \]

\[ \text{ONLYONES}_{TM} \]

\[ \text{EQ}^{TM} \]

\[ \text{REGULAR}^{TM} \]

\[ \text{HALT} \]

\[ L_D \]

\[ A_{TM} \]

\[ 0^*1^* \]

\[ \text{DOGWALK} \]

\[ \text{ADD} \]

\[ \text{ONES}^{TM} \]

\[ \text{ONLYONES}^{TM} \]

\[ \text{EQ}^{TM} \]

\[ \text{REGULAR}^{TM} \]

All Languages
What problems can be solved **efficiently** a computer?
Where We're Going

• The class $\mathbf{P}$ represents problems that can be solved \textit{efficiently} by a computer.

• The class $\mathbf{NP}$ represents problems where answers can be verified \textit{efficiently} by a computer.

• The class $\text{co-}\mathbf{NP}$ represents problems where answers can be \textit{efficiently} refuted by a computer.

• The \textit{polynomial-time} mapping reduction can be used to find connections between problems.
Next Time

- **Introduction to Complexity Theory**
  - How do you define efficiency?
  - How do you measure it?
  - What tools will we need?

- **Complexity Class P**
  - What problems can be solved efficiently?
  - How do we reason about them?
Have a wonderful Thanksgiving!