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**, **RE**, and co-**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: $\boldsymbol{A \leq_M B}$ 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.

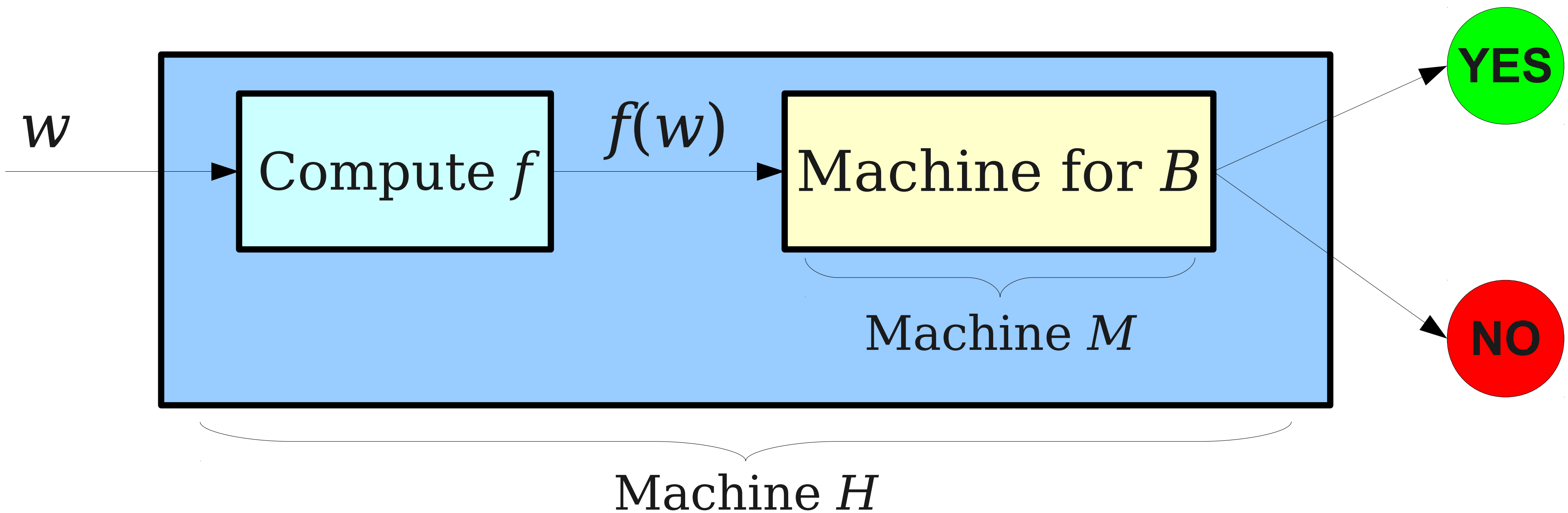
Why Mapping Reducibility Matters

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.

Sketch of the Proof



H = "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 *HALT* ∉ **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

$$\boldsymbol{DECIDER} = \{\ \langle \boldsymbol{M} \rangle \mid \boldsymbol{M} \textbf{ is a decider } \}$$

- How would we prove that *DECIDER* is, itself, undecidable?

$HALT \leq_M DECIDER$

- We will prove that *DECIDER* is undecidable by reducing *HALT* to *DECIDER*.
- Want to find a function *f* such that

$\langle M, w \rangle \in HALT$ iff $f(\langle M, w \rangle) \in DECIDER$.

- Assuming that $f(\langle M, w \rangle) = \langle M' \rangle$ for some TM M', we have that

$\langle M, w \rangle \in HALT$ iff $\langle M' \rangle \in DECIDER$.

***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' = "On input x:

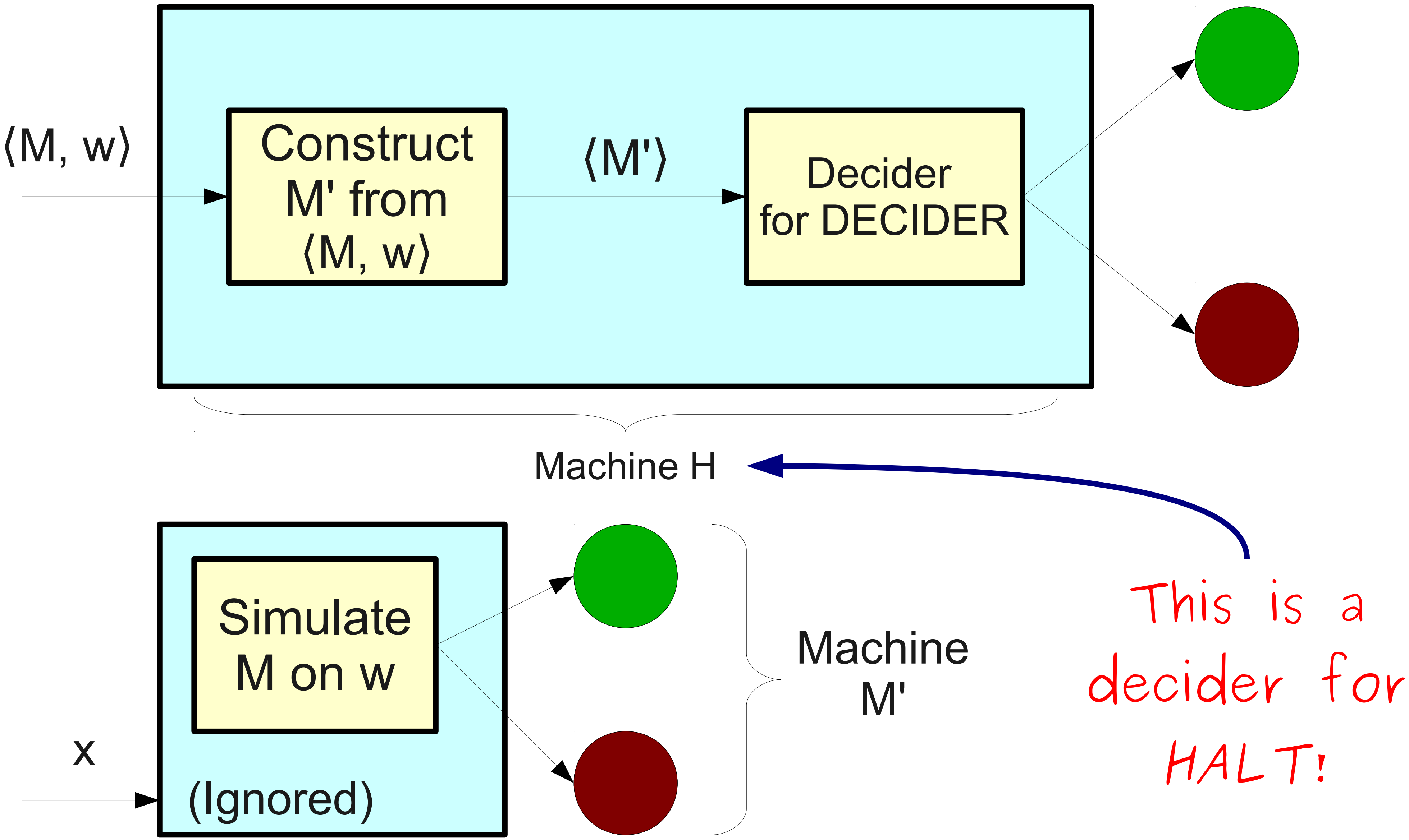
  Ignore x.

  Run M on w.

  If M accepts w, accept.

  If M rejects w, reject."
- 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



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' = "On input x:

Ignore x.

Run M on w.

If M accepts w, accept.

If M rejects w, reject."

The Parameterization Theorem

Theorem: Let M be a TM of the form

M = "On input $\langle x_1, x_2, \ldots, x_n \rangle$:

Do something with $x_1, x_2, \ldots, x_n$"

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

M' = "On input $\langle x_2, \ldots, x_n \rangle$:

Do something with $p, x_2, \ldots, x_n$"

Justifying M'

- Consider this machine X:

  X = "On input $\langle N, z, x\rangle$:

  Ignore x.

  Run N on z.

  If N accepts z, accept.

  If N rejects z, reject."

- Applying the parameterization theorem twice with the values M and w produces the machine

  M' = "On input x:

  Ignore x.

  Run M on w.

  If M accepts w, accept.

  If M rejects w, 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: $HALT \leq_M DECIDER$.

Proof: We exhibit a mapping reduction from *HALT* to *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' = "On input x:
Ignore x.
Run M on w.
If M accepts w, accept.
If M rejects w, reject."

By the parameterization theorem, f is a computable function. We further claim that $\langle M, w\rangle \in HALT$ iff $f(\langle M, w\rangle) \in DECIDER$. To see this, note that $f(\langle M, w\rangle) = \langle M'\rangle \in DECIDER$ 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 $\langle M, w\rangle \in HALT$. Thus $\langle M, w\rangle \in HALT$ iff $f(\langle M, w\rangle) \in DECIDER$. Therefore, f is a mapping reduction from *HALT* to *DECIDER*, so $HALT \leq_M DECIDER$. ■

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 `1`s?
- Let ONES_{TM} be the following language:

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

- Is $\text{ONES}_{\text{TM}} \in \mathbf{R}$? Is it **RE**?

ONES_{TM}

- Unfortunately, ONES_{TM} is undecidable.
- However, ONES_{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_{TM} is undecidable by showing that $A_{\text{TM}} \leq_M$ ONES.

$A_{TM} \leq_M ONES_{TM}$

- As before, let's try to find a function *f* such that

  $\langle M, w \rangle \in A_{TM}$ iff $f(\langle M, w \rangle) \in ONES_{TM}$.

- Let's let $f(\langle M, w \rangle) = \langle M' \rangle$ for some TM *M*'. Then we want to pick *M*' such that

  $\langle M, w \rangle \in A_{TM}$ iff $f(\langle M, w \rangle) \in ONES_{TM}$

  $\langle M, w \rangle \in A_{TM}$ iff $\langle M' \rangle \in ONES_{TM}$

  M* accepts *w* iff *M*' accepts 1^n for some *n

The Reduction

- Goal: construct M' so M' accepts $\mathtt{1}^n$ for some n iff M accepts w.
- Here is one possible option:

  M' = "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 $\mathtt{1}^n$ 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 $\mathtt{1}^n$.

Theorem: $A_{TM} \leq_M ONES_{TM}$.

Proof: We exhibit a mapping reduction from A_{TM} to $ONES_{TM}$. For any TM/string pair ⟨*M*, *w*⟩, let *f*(⟨*M*, *w*⟩) = ⟨*M*'⟩, 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 ⟨*M*, *w*⟩ ∈ A_{TM} iff *f*(⟨*M*, *w*⟩) ∈ $ONES_{TM}$. To see this, note that f(⟨*M*, *w*⟩) = ⟨*M*'⟩ ∈ $ONES_{TM}$ 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 ⟨*M*, *w*⟩ ∈ A_{TM}. Thus ⟨M, w⟩ ∈ A_{TM} iff *f*(⟨*M*, *w*⟩) ∈ $ONES_{TM}$. 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 `1`s.
- Can we determine whether a TM *only* accepts strings of all `1`s?
- In other words, for a TM M, is $\mathscr{L}(M) \subseteq \mathtt{1}^*$?
- Let $\text{ONLYONES}_{\text{TM}}$ be the language

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

- Is $\text{ONLYONES}_{\text{TM}} \in \mathbf{R}$? How about $\mathbf{RE}$?

$\text{ONLYONES}_{\text{TM}} \notin \textbf{RE}$

- 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 $L_{\text{D}} \leq_{\text{M}} \text{ONLYONES}_{\text{TM}}$, then we have that $\text{ONLYONES}_{\text{TM}} \notin \textbf{RE}$.

$L_D \leq_M \text{ONLYONES}_{TM}$

- We want to find a computable function f such that

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

- We want to set $f(\langle M \rangle) = \langle M' \rangle$ for some suitable choice of M'. This means

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

$$\langle M \rangle \notin \mathscr{L}(M) \quad \text{iff} \quad \mathscr{L}(M') \subseteq 1^*$$

- How would we pick our machine M'?

One Possible Reduction

- We want to build M' from M such that $\langle M \rangle \notin \mathscr{L}(M)$ iff $\mathscr{L}(M') \subseteq$ `1`*.
- In other words, we construct M' such that
  - If $\langle M \rangle \in \mathscr{L}(M)$, then $\mathscr{L}(M')$ is not a subset of `1`*.
  - If $\langle M \rangle \notin \mathscr{L}(M)$, then $\mathscr{L}(M')$ 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 $\langle M \rangle \in \mathscr{L}(M)$.
  - If $\langle M \rangle \in \mathscr{L}(M)$, then $\mathscr{L}(M') = \Sigma^*$, which isn't a subset of `1`*.
  - If $\langle M \rangle \notin \mathscr{L}(M)$, then $\mathscr{L}(M') = \emptyset$, which is a subset of `1`*.

One Possible Reduction

- We want
  - If $\langle M \rangle \in \mathscr{L}(M)$, then $\mathscr{L}(M') = \Sigma^*$
  - If $\langle M \rangle \notin \mathscr{L}(M)$, then $\mathscr{L}(M') = \emptyset$
- Here is one possible M' that does this:

M' = "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: $L_D \leq_M$ ONLYONES$_{TM}$.

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

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

M' = "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."

By the parameterization theorem, f is a computable function. We further claim that $\langle M \rangle \in L_D$ iff $f(\langle M \rangle) \in$ ONLYONES$_{TM}$. To see this, note that $f(\langle M \rangle) = \langle M' \rangle \in$ ONLYONES$_{TM}$ iff $\mathscr{L}(M') \subseteq \mathbf{1}^*$. We claim that $\mathscr{L}(M') \subseteq \mathbf{1}^*$ 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 $\mathscr{L}(M') = \emptyset \subseteq \mathbf{1}^*$. Otherwise, if M accepts $\langle M \rangle$, then M' accepts all strings, so $\mathscr{L}(M) = \Sigma^*$, which is not a subset of $\mathbf{1}^*$. Finally, M does not accept $\langle M \rangle$ iff $\langle M \rangle \in L_D$. Thus $\langle M \rangle \in L_D$ iff $f(\langle M \rangle) \in$ ONLYONES$_{TM}$. Consequently, f is a mapping reduction from L_D to ONLYONES$_{TM}$, so $L_D \leq_M$ ONLYONES$_{TM}$ as required. ■

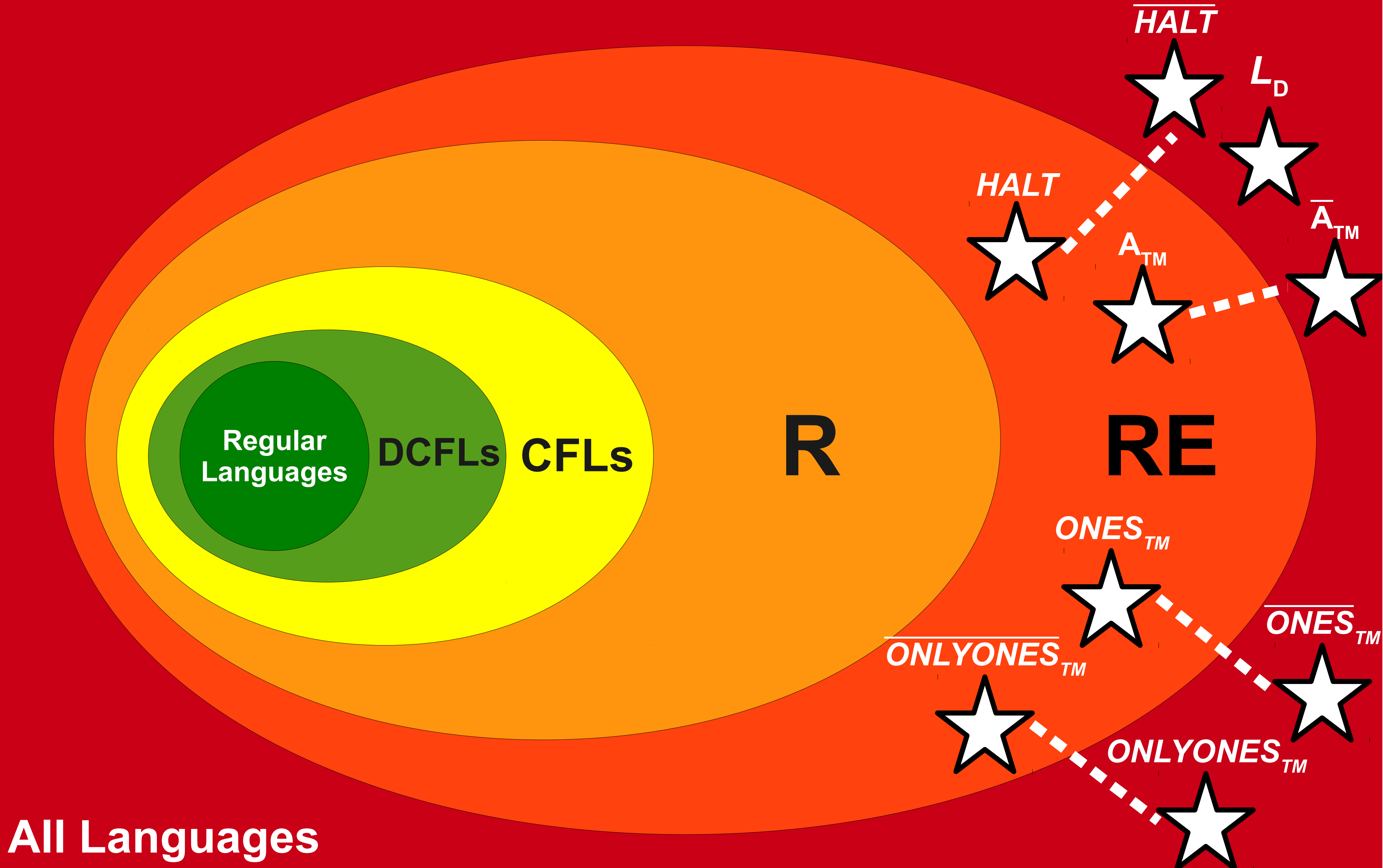
$\overline{\text{ONLYONES}_{\text{TM}}}$

- Although $\text{ONLYONES}_{\text{TM}}$ is not **RE**, its complement ($\overline{\text{ONLYONES}_{\text{TM}}}$) *is* **RE**:

$$\{ \langle M \rangle \mid \mathscr{L}(M) \text{ is not a subset of } 1^* \}$$

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

The Limits of Computability



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:

$$L \in \text{co-}\mathbf{RE} \quad \text{iff} \quad \overline{L} \in \mathbf{RE}$$

$$\overline{L} \in \text{co-}\mathbf{RE} \quad \text{iff} \quad L \in \mathbf{RE}$$

- 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.

$\mathbf{R}$, $\mathbf{RE}$, and co-$\mathbf{RE}$

- Recall:

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

- Rewritten in terms of co-$\mathbf{RE}$:

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

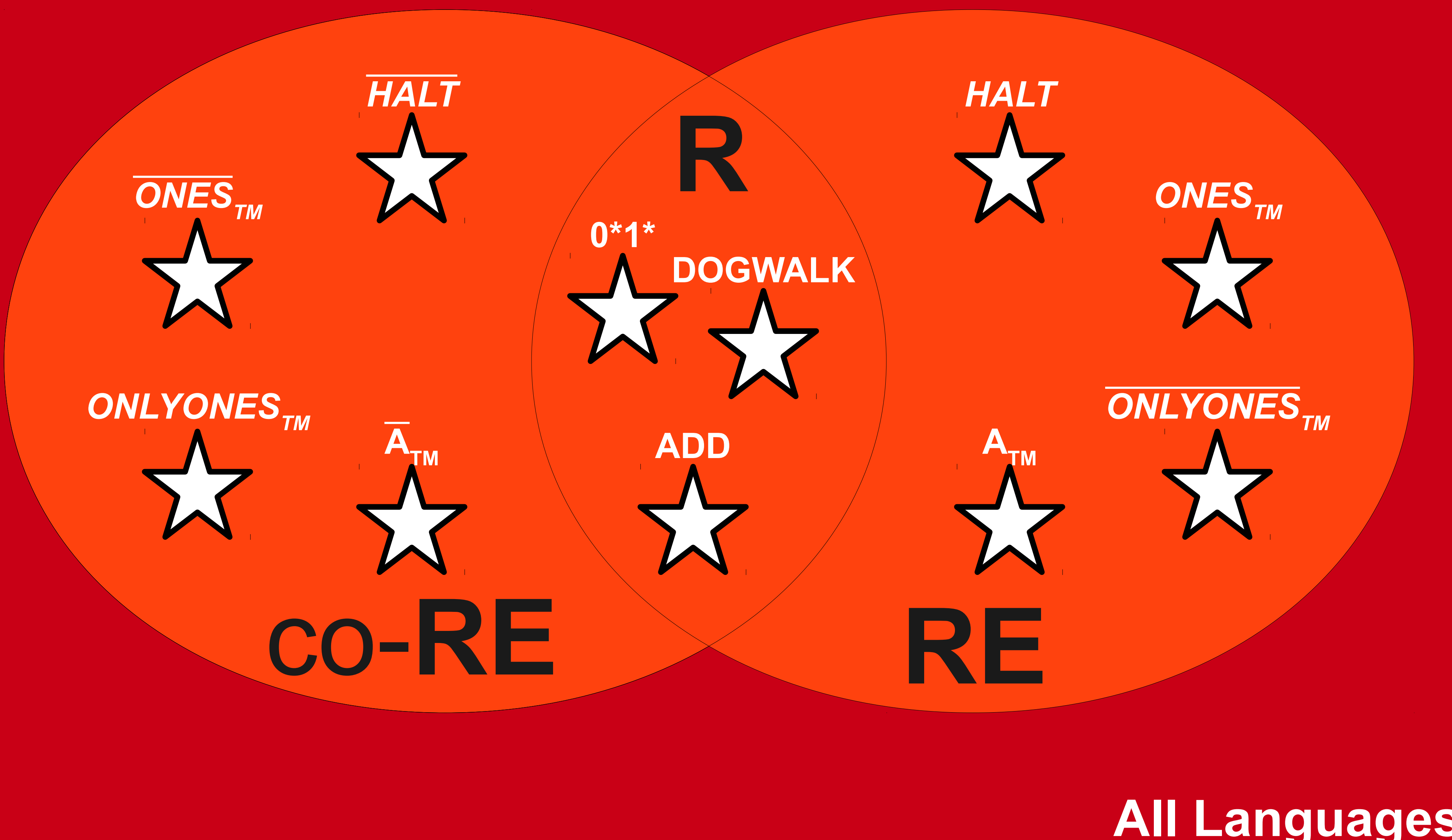
- In other words:

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

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

$$\mathbf{R} = \mathbf{RE} \cap \text{co-}\mathbf{RE}$$

The Limits of Computability



L_D Revisited

- The diagonalization language L_D is the language

  $\boldsymbol{L_D = \{\langle M \rangle \mid M \text{ is a TM and } M \notin \mathscr{L}(M)\}}$

- As we saw before, $L_D \notin$ **RE**.
- So where is L_D? Is it in $L_D \in$ co-**RE**? Or is it someplace else?

$\overline{L}_D$

- To see whether $L_D \in$ co-**RE**, we will see whether $\overline{L}_D \in$ **RE**.
- The language $\overline{L}_D$ is the language

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

- Two questions:
  - What is this language?
  - Is this language **RE**?

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

Acc	Acc	Acc	No	Acc	No	…

$\{ \langle M \rangle \mid M$ is a TM and $\langle M \rangle \in \mathscr{L}(M) \}$

This language is $\overline{L_D}$.

$L_D \in$ co-**RE**

- Here's an TM for $\overline{L_D}$:

  R = "On input $\langle M \rangle$:

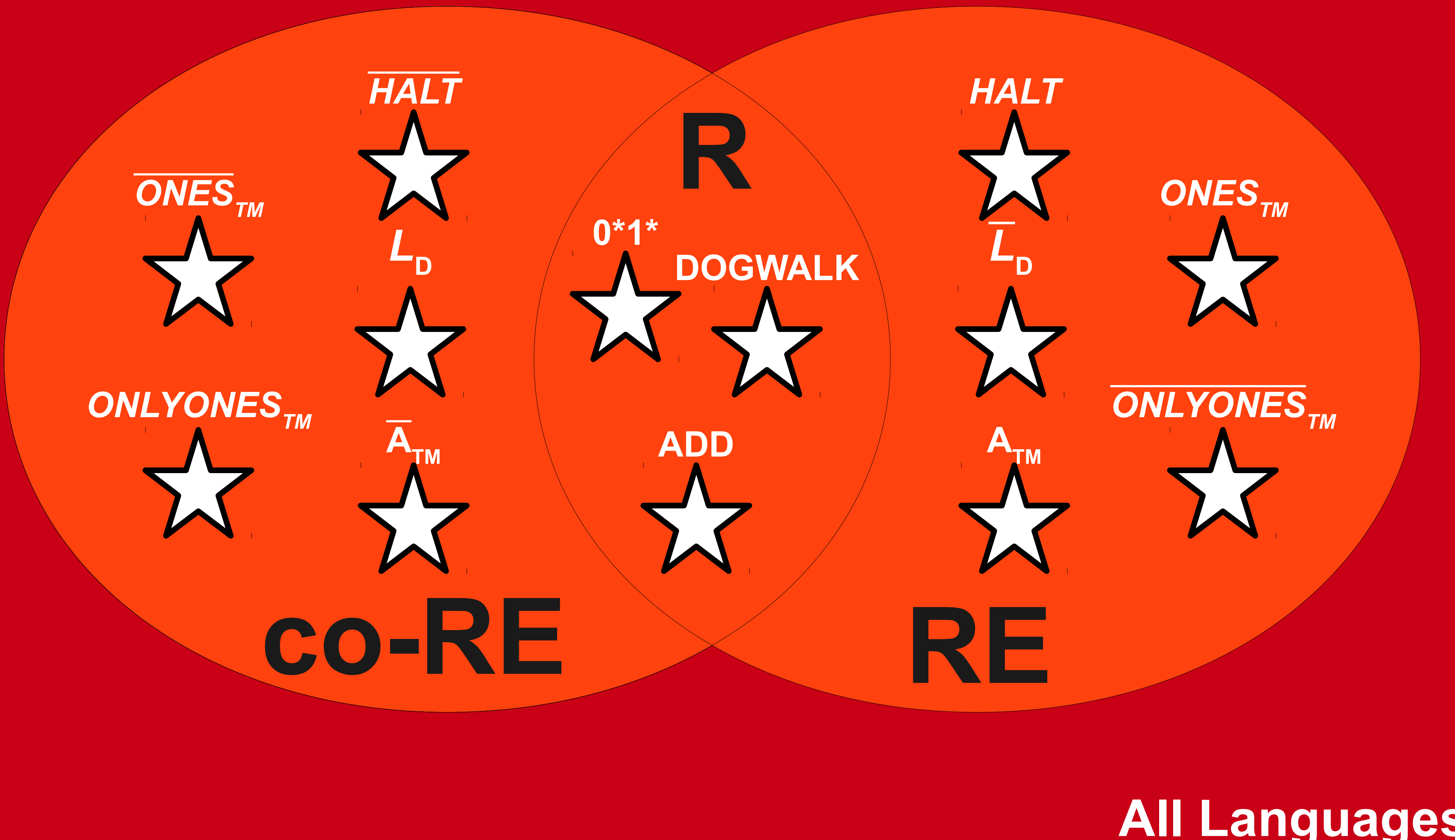
  Run M on $\langle M \rangle$.

  If M accepts $\langle M \rangle$, accept.

  If M rejects $\langle M \rangle$, reject."

- Then R accepts $\langle M \rangle$ iff $\langle M \rangle \in \mathscr{L}(M)$ iff $\langle \mathrm{M} \rangle \in \overline{L_D}$, so $\mathscr{L}(R) = \overline{L_D}$.

The Limits of Computability



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, $\overline{A} \leq_M \overline{B}$. ■

co-**RE** Reductions

- **Corollary:** If $A \leq_M B$ and $B \in$ co-**RE**, then $A \in$ co-**RE**.

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

- **Corollary:** If $A \leq_M B$ and $A \notin$ co-**RE**, then $B \notin$ co-**RE**.

  Proof: Take the contrapositive of the above. ■

Why Mapping Reducibility Matters

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

$$A \leq_M B$$

... then this one is "easy" (R or RE or co-RE) too.

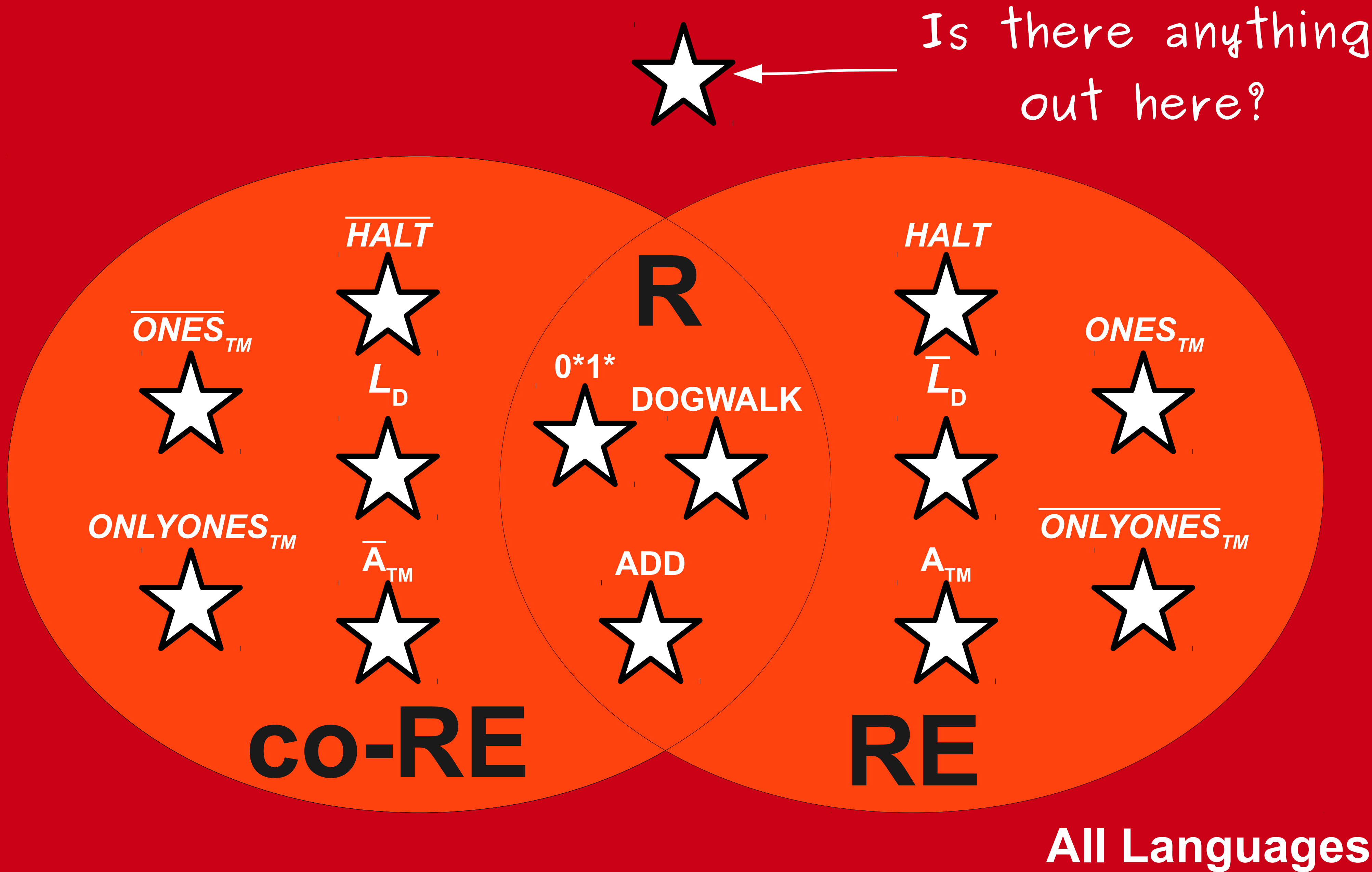
Why Mapping Reducibility Matters

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



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 **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:

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

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

$\text{REGULAR}_{\text{TM}} \notin \textbf{RE}$

- It turns out that $\text{REGULAR}_{\text{TM}}$ 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 $L_D \leq_M \text{REGULAR}_{\text{TM}}$.

$L_D \leq_M$ REGULAR$_{TM}$

- We want to find a computable function f such that

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

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

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

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

$$\langle M \rangle \notin \mathscr{L}(M) \quad \text{iff} \quad \mathscr{L}(M') \text{ is regular.}$$

$L_D \leq_M \text{REGULAR}_{TM}$

- We want to construct some M' out of M such that
  - If $\langle M \rangle \in \mathscr{L}(M)$, then $\mathscr{L}(M')$ is not regular.
  - If $\langle M \rangle \notin \mathscr{L}(M)$, then $\mathscr{L}(M')$ 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 $\langle M \rangle \notin \mathscr{L}(M)$.
  - If $\langle M \rangle \in \mathscr{L}(M)$, then $\mathscr{L}(M') = \{ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N} \}$
  - If $\langle M \rangle \notin \mathscr{L}(M)$, then $\mathscr{L}(M') = \emptyset$

The Reduction

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

M' = "On input x:

If x does not have the form $\mathtt{0}^n\mathtt{1}^n$, reject.

Run M on $\langle M \rangle$.

If M accepts, accept x.

If M rejects, reject x."

Theorem: $L_D \leq_M \text{REGULAR}_{TM}$.

Proof: We exhibit a mapping reduction from L_D to 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' = "On input x:

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 $\mathscr{L}(M')$ is regular. We claim that $\mathscr{L}(M')$ is regular iff $\langle M \rangle \notin \mathscr{L}(M)$. To see this, note that if $\langle M \rangle \notin \mathscr{L}(M)$, then M' never accepts any strings. Thus $\mathscr{L}(M') = \emptyset$, which is regular. Otherwise, if $\langle M \rangle \in \mathscr{L}(M)$, then M' accepts all strings of the form 0^n1^n, so we have that $\mathscr{L}(M) = \{ 0^n1^n \mid n \in \mathbb{N} \}$, which is not regular. Finally, $\langle M \rangle \notin \mathscr{L}(\langle M \rangle)$ 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 REGULAR_{TM}. Therefore, $L_D \leq_M \text{REGULAR}_{TM}$. ■

$\text{REGULAR}_{\text{TM}} \notin$ co-**RE**

- Not only is $\text{REGULAR}_{\text{TM}} \notin$ **RE**, but $\text{REGULAR}_{\text{TM}} \notin$ 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.

$\overline{L}_D \leq_M$ REGULAR$_{TM}$

- To prove that REGULAR$_{TM}$ is not co-**RE**, we will prove that $\overline{L}_D \leq_M$ REGULAR$_{TM}$.
- Since $\overline{L}_D$ is not co-**RE**, this proves that REGULAR$_{TM}$ is not co-**RE** either.
- Goal: Find a function f such that

$$\langle M \rangle \in \overline{L}_D \quad \textbf{iff} \quad f(\langle M \rangle) \in \text{REGULAR}_{TM}$$

- Let $f(\langle M \rangle) = \langle M' \rangle$ for some TM M'. Then we want

$$\langle M \rangle \in \overline{L}_D \quad \textbf{iff} \quad \langle M' \rangle \in \text{REGULAR}_{TM}$$

$$\langle M \rangle \in \mathscr{L}(M) \quad \textbf{iff} \quad \mathscr{L}(M') \textbf{ is regular}$$

$\overline{L}_D \leq_M \text{REGULAR}_{TM}$

- We want to construct some M' out of M such that
  - If $\langle M \rangle \in \mathscr{L}(M)$, then $\mathscr{L}(M')$ is regular.
  - If $\langle M \rangle \notin \mathscr{L}(M)$, then $\mathscr{L}(M')$ 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 $\langle M \rangle \in \mathscr{L}(M)$.
  - If $\langle M \rangle \in \mathscr{L}(M)$, then $\mathscr{L}(M') = \Sigma^*$.
  - If $\langle M \rangle \notin \mathscr{L}(M)$, then $\mathscr{L}(M') = \{\texttt{0}^n\texttt{1}^n \mid n \in \mathbb{N}\}$

$\overline{L}_D \leq_M \text{REGULAR}_{TM}$

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

M' = "On input x:

If x has the form $\mathtt{0}^n\mathtt{1}^n$, accept.

Run M on $\langle M \rangle$.

If M accepts, accept x.

If M rejects, reject x."

Theorem: $\overline{L}_D \leq_M$ REGULAR$_{TM}$.

Proof: We exhibit a mapping reduction from $\overline{L}_D$ to 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' = "On input x:

If x has the form 0^n1^n, accept 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 $\langle M \rangle \in \overline{L}_D$ iff $f(\langle M \rangle) \in$ REGULAR$_{TM}$. To see this, note that $f(\langle M \rangle) = \langle M' \rangle \in$ REGULAR$_{TM}$ iff $\mathscr{L}(M')$ is regular. We claim that $\mathscr{L}(M')$ is regular iff $\langle M \rangle \in \mathscr{L}(M)$. To see this, note that if $\langle M \rangle \in \mathscr{L}(M)$, 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 $\mathscr{L}(M') = \Sigma^*$, which is regular. Otherwise, if $\langle M \rangle \notin \mathscr{L}(M)$, then M' only accepts strings of the form 0^n1^n, so $\mathscr{L}(M) = \{ 0^n1^n \mid n \in \mathbb{N} \}$, which is not regular. Finally, $\langle M \rangle \in \mathscr{L}(\langle M \rangle)$ iff $\langle M \rangle \in \overline{L}_D$. Thus $\langle M \rangle \in \overline{L}_D$ iff $f(\langle M \rangle) \in$ REGULAR$_{TM}$, so f is a mapping reduction from $\overline{L}_D$ to REGULAR$_{TM}$. Therefore, $\overline{L}_D \leq_M$ REGULAR$_{TM}$. ■

The Limits of Computability

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

- The most famous problem that is neither **RE** nor co-**RE** is the TM equality problem:

$$\mathrm{EQ}_{\mathrm{TM}} = \{ \langle M_1, M_2 \rangle \mid \mathscr{L}(M_1) = \mathscr{L}(M_2) \}$$

- 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

What problems can be
solved **efficiently** a computer?

Where We're Going

- The class **P** represents problems that can be solved *efficiently* by a computer.
- The class **NP** represents problems where answers can be verified *efficiently* by a computer.
- The class co-**NP** represents problems where answers can be *efficiently* refuted by a computer.
- The *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!