CS 302.1 - Automata Theory

Lecture 13

Shantanav Chakraborty

Center for Quantum Science and Technology (CQST)

Center for Security, Theory and Algorithms (CSTAR)

IIIT Hyderabad

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Quick Recap

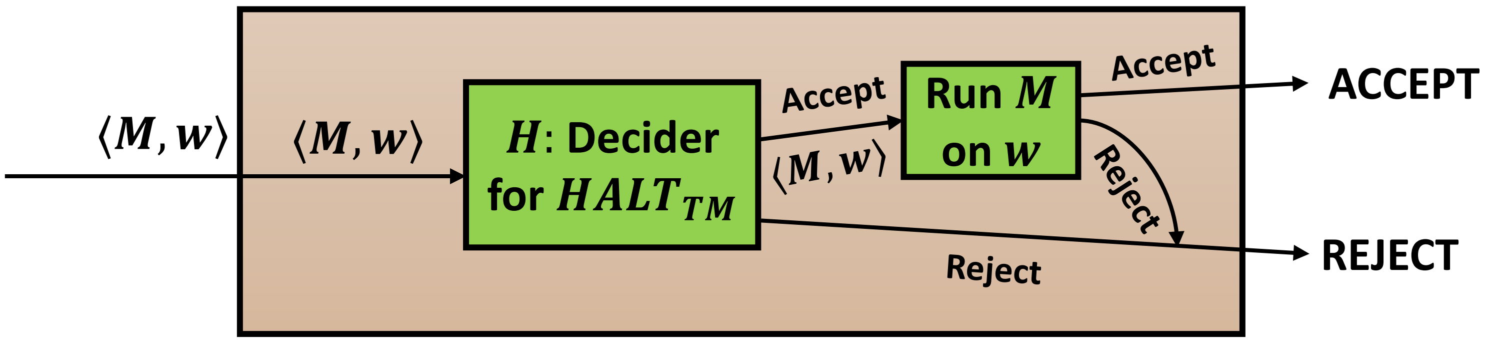
$HALT_{TM} = \{\langle M, w\rangle | M$ **halts on input** $w\}$. **Is** $HALT_{TM}$ **decidable?**

The Halting Problem: Does there exist a Total Turing Machine H that accepts as input a Turing Machine M and an input string w and outputs YES, if $M(w)$ halts (accepts or rejects) and NO, if $M(w)$ does not halt (loops forever), i.e.

$$H(\langle M, w\rangle) = \begin{cases} \textbf{ACCEPTS, if } M(w) \textbf{ HALTS, i.e. accepts or rejects} \\ \textbf{REJECTS, if } M(w) \textbf{ does not HALT, i.e. loops infinitely} \end{cases}$$

NO! $A_{TM} \preccurlyeq HALT_{TM}$

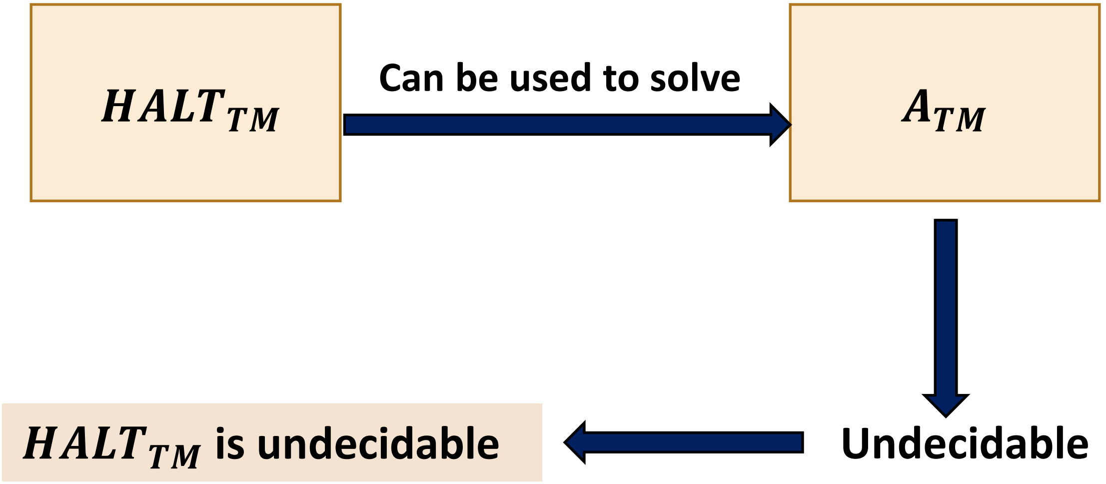
$A =$ On input $\langle M, w\rangle$
- Run $H(\langle M, w\rangle)$
- If H rejects, output $REJECT$
- If H accepts,
  - Run $M(w)$
  - If $M(w)$ accepts, output $ACCEPT$
  - If $M(w)$ rejects, output $REJECT$



Quick Recap

Generally,

- A language $\boldsymbol{A}$ **reduces to** another language $\mathbf{B}$ ($\boldsymbol{A} \preccurlyeq \boldsymbol{B}$) iff we can build a **solver for $\boldsymbol{A}$** using a **solver for $\boldsymbol{B}$**
- In terms of computability, suppose using B we can compute A. Then, if **$\boldsymbol{A}$ is undecidable then so is $\boldsymbol{B}$**.
- We showed: $\boldsymbol{A_{TM}} \preccurlyeq \boldsymbol{HALT_{TM}}$ to prove that $\boldsymbol{HALT_{TM}}$ is undecidable.



Suppose, $\boldsymbol{A} \preccurlyeq \boldsymbol{B}$

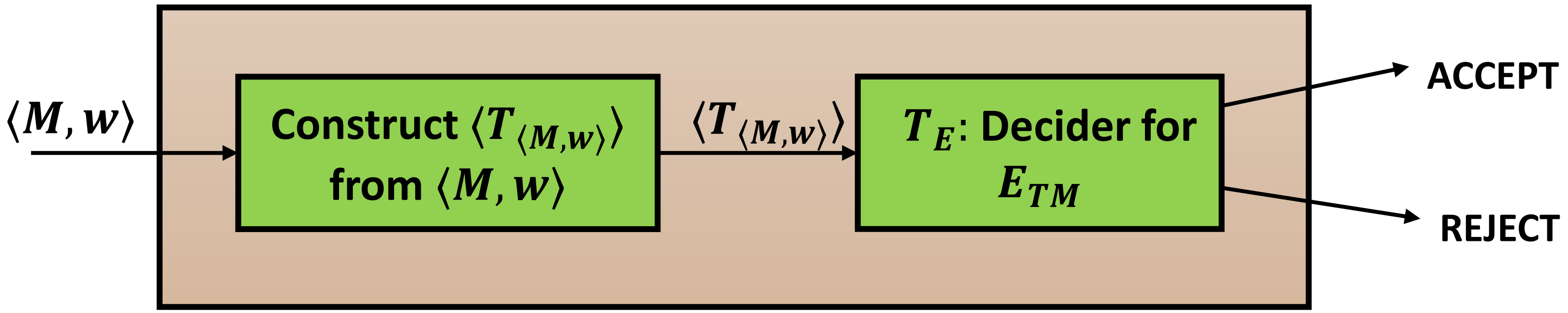
- If A is undecidable then B is undecidable.
- **If $\boldsymbol{B}$ is decidable then $\boldsymbol{A}$ is decidable.**

Quick Recap

$E_{TM} = \{\langle M \rangle | M$ is a Turing Machine and $L(M) = \Phi\}$. **Is E_{TM} decidable?**

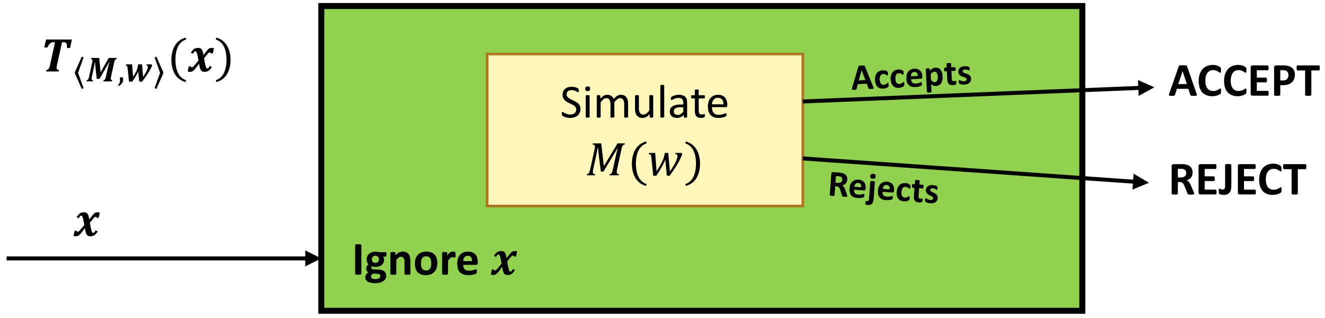
NO! $\overline{A}_{TM} \preccurlyeq E_{TM}$

Proof: Let T_E be the Turing Machine that decides E_{TM}. We shall prove that $\overline{A_{TM}} \preccurlyeq E_{TM}$ by constructing a Turing Machine N for $\overline{A_{TM}}$ using T_E.



$T_{\langle M,w \rangle}(x) =$

- Ignore x.
- Run M on w
  - If M accepts w, accept x
  - If M rejects w, reject x



$$L(T) = \begin{cases} \Phi, \text{ if } M(w) \text{ doesn't accept} \\ \Sigma^*, \text{ if } M(w) \text{ accepts} \end{cases}$$

Quick Recap

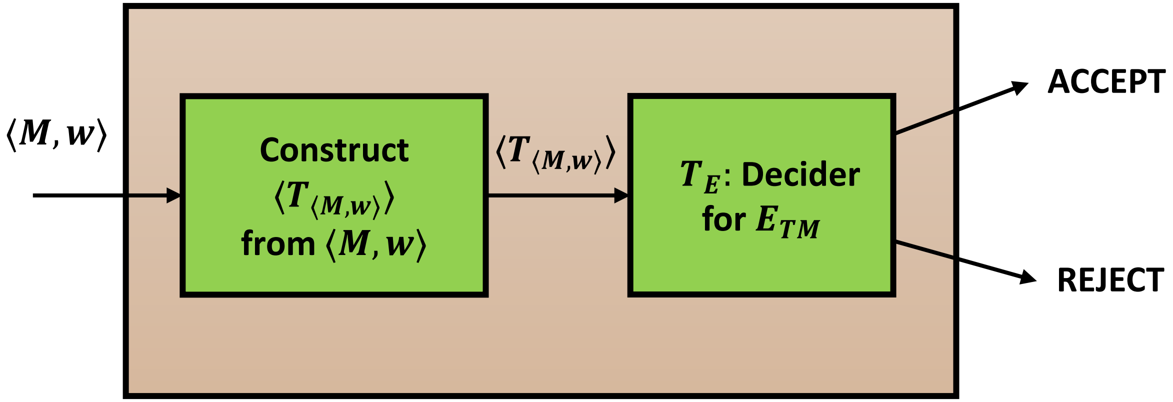
$\boldsymbol{E_{TM}} = \{\langle M\rangle | M$ is a Turing Machine and $L(M) = \Phi\}$

Proof: Let T_E be the Turing Machine that decides E_{TM}. We shall prove that $\overline{\boldsymbol{A_{TM}}} \preccurlyeq \boldsymbol{E_{TM}}$ by constructing a Turing Machine N that decides $\overline{\boldsymbol{A_{TM}}}$ using $\boldsymbol{T_E}$.

$N(\langle M, w\rangle) =$

- Construct $\langle T_{\langle M,w\rangle} \rangle$, the encoding of $T_{\langle M,w\rangle}$ such that for any input x it works as follows:
  - Ignore x.
  - Run M on w
    - If M accepts w, accept x
    - If M rejects w, reject x

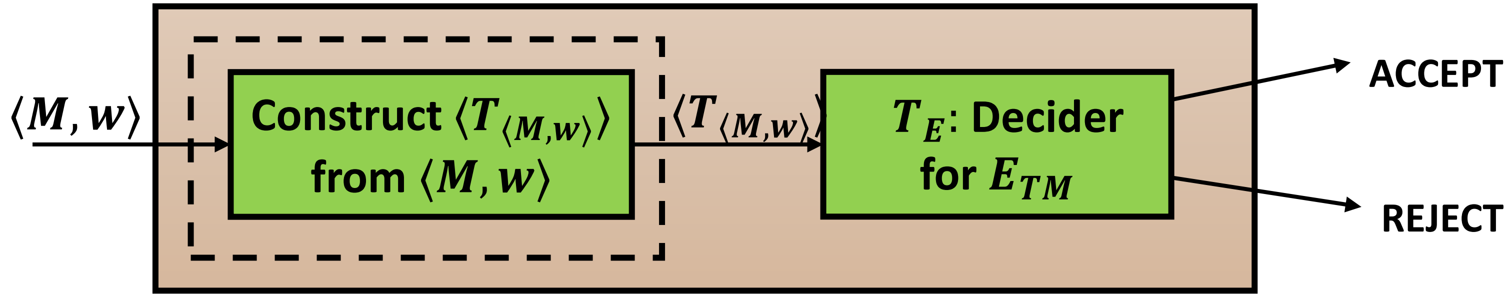
- Send $\langle T_{\langle M,w\rangle}\rangle$ to T_E and Output
  ACCEPT if T_E accepts
  REJECT if T_E rejects



- $\overline{\boldsymbol{A_{TM}}} \preccurlyeq \boldsymbol{E_{TM}}$
- $\overline{\boldsymbol{A_{TM}}}$ **is undecidable**
- $\boldsymbol{E_{TM}}$ **is undecidable!**

Quick Recap

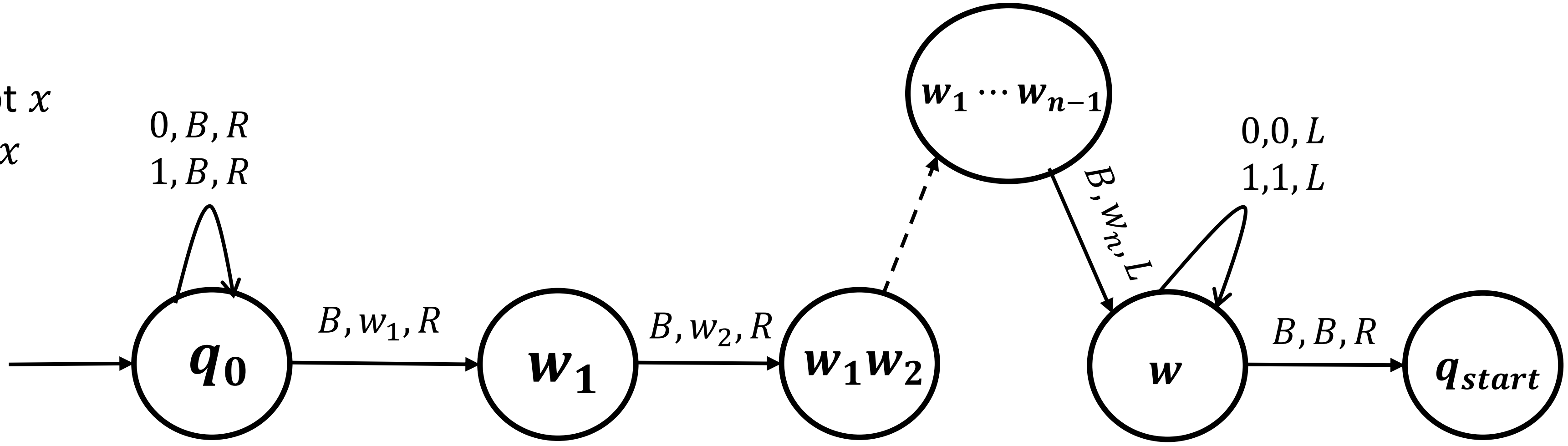
$\boldsymbol{E_{TM}} = \{\langle M\rangle | M$ is a Turing Machine and $L(M) = \Phi\}$



$N(\langle M, w\rangle) =$

- Construct $\langle T_{\langle M,w\rangle}\rangle$, the encoding of $T_{\langle M,w\rangle}$ such that for any input x it works as follows:
  - Ignore x.
  - Run M on w
    - If M accepts w, accept x
    - If M rejects w, reject x

- Send $\langle T_{\langle M,w\rangle}\rangle$ to T_E and Output
  ACCEPT if T_E accepts
  REJECT if T_E rejects



Undecidability

$\boldsymbol{E_{TM}} = \{\langle M \rangle | M$ is a Turing Machine and $L(M) = \Phi\}$

Proof: Let T_E be the Turing Machine that decides E_{TM}. We shall prove that $\overline{\boldsymbol{A_{TM}}} \preccurlyeq \boldsymbol{E_{TM}}$ by constructing a Turing Machine N that decides $\overline{\boldsymbol{A_{TM}}}$ using $\boldsymbol{T_E}$.

$N(\langle M, w \rangle) =$

- Construct $\langle T_{\langle M,w \rangle} \rangle$, the encoding of $T_{\langle M,w \rangle}$ such that for any input x it works as follows:
  - Ignore x.
  - Run M on w
    - If M accepts w, accept x
    - If M rejects w, reject x

- Send $\langle T_{\langle M,w \rangle} \rangle$ to T_E and Output
  ACCEPT if T_E accepts
  REJECT if T_E rejects

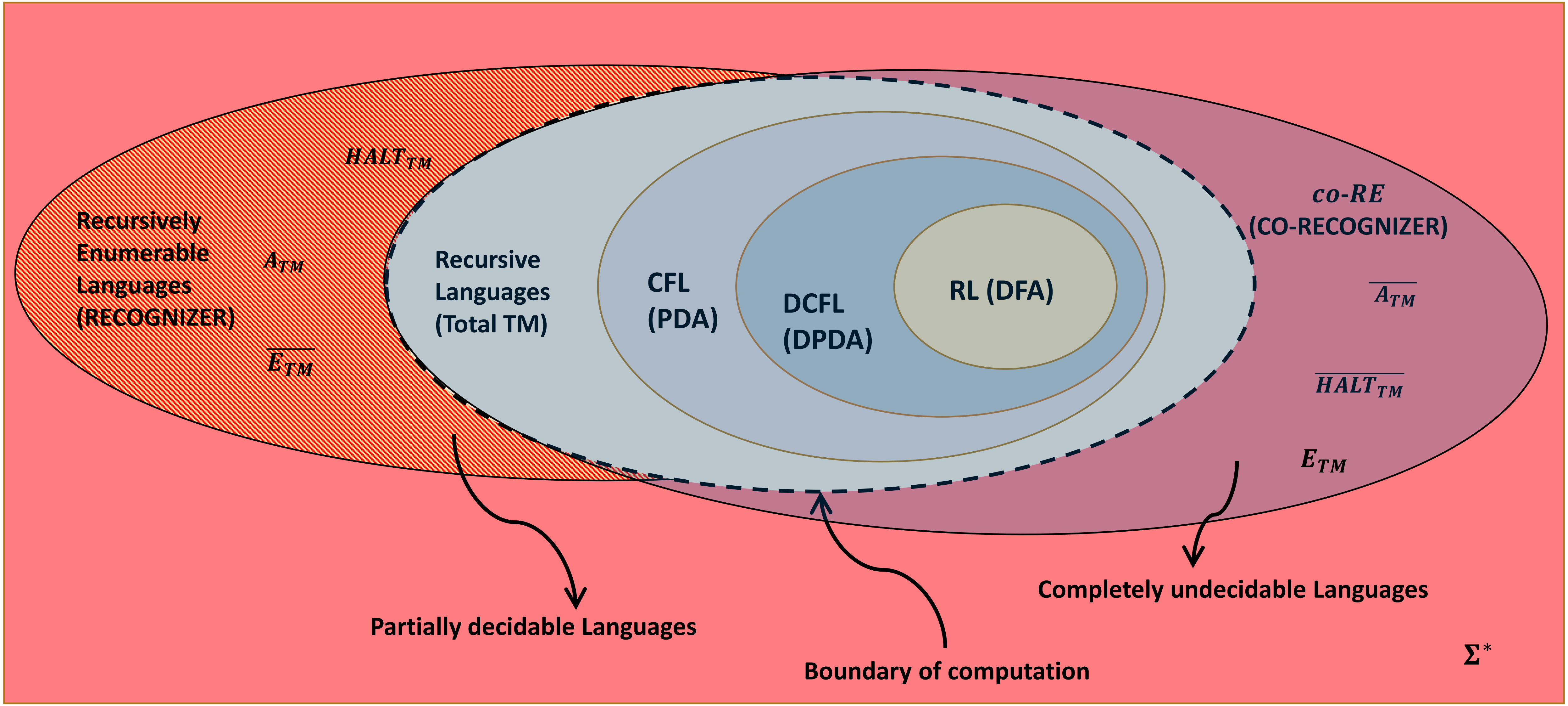
- $\overline{\boldsymbol{A_{TM}}} \preccurlyeq \boldsymbol{E_{TM}}$
- $\overline{\boldsymbol{A_{TM}}}$ **is undecidable**
- $\boldsymbol{E_{TM}}$ **is undecidable!**

$\boldsymbol{E_{TM}} \in$ **co-RE**$-\boldsymbol{R}$

Proof idea: We can build a co-recognizer for E_{TM}.

C = On input $\langle M \rangle$
For $i = 1, 2, 3, \cdots$
For $j = 1, 2, 3, \cdots i$
Run M on s_j for i steps.
If M accepts s_j, REJECT.

Everything in one slide



Closure properties

Are Recursive languages **closed under Union**? If R_1 and R_2 are recursive, is $R_1 \cup R_2$ recursive?

Proof:

- Let M_1 and M_2 be the Total Turing Machines corresponding to R_1 and R_2 respectively.
- Using them, we construct a Total Turing Machine M' such that M' decides $R_1 \cup R_2$.

Closure properties

Are Recursive languages **closed under Union**? If R_1 and R_2 are recursive, is $R_1 \cup R_2$ recursive?

Proof:

- Let M_1 and M_2 be the Total Turing Machines corresponding to R_1 and R_2 respectively.
- Using them, we construct a Total Turing Machine M' such that M' decides $R_1 \cup R_2$.

$M' =$ On input w

Run $M_1(w)$

Run $M_2(w)$

If either of them accept, $ACCEPT$

If both rejects, $REJECT$

Recursive languages are **CLOSED under Union.**

Closure properties

Are Recursive languages **closed under intersection**? If R_1 and R_2 are recursive, is $R_1 \cap R_2$ recursive?

Proof:

- Let M_1 and M_2 be the Total Turing Machines corresponding to R_1 and R_2 respectively.
- Using M, we construct a Total Turing Machine M' such that M' decides $R_1 \cap R_2$.

$M' =$ On input w

Run $M_1(w)$

Run $M_2(w)$

If both of them accept, $ACCEPT$

If either rejects, $REJECT$

Recursive languages are **CLOSED under Intersection.**

Closure properties

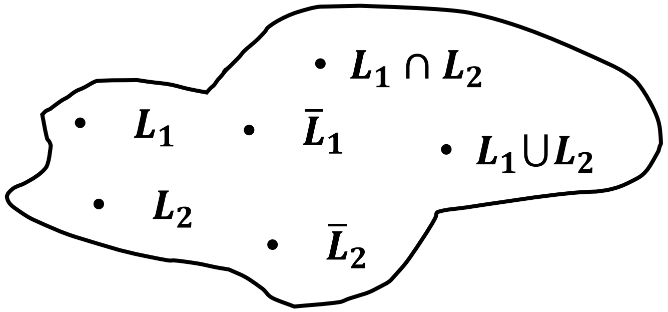
Are Recursive languages **closed under complementation**? If some language R is recursive, is $\bar{R}$ also recursive?

Proof:

- If R is recursive then there exists a total TM M for R.
- Using M, we construct a Total Turing Machine $\bar{M}$ such that $\bar{M}$ decides $\bar{R}$.

$\bar{M}$ $=$ On input w
Run $M(w)$.
If M accepts, $REJECT$
If M rejects, $ACCEPT$

Recursive languages are **CLOSED under complementation**



Set of all recursive (decidable) languages

Closure properties

L and $\overline{L}$ are both Recursively Enumerable if and only if L is Recursive.

Closure properties

L and $\overline{L}$ are both Recursively Enumerable if and only if L is Recursive.

Proof: In one direction, the proof is trivial. That is, if L is Recursive then so is $\overline{L}$. As Recursive Languages $R \subseteq RE$, we have that both L and $\overline{L}$ are in RE.

Closure properties

$\boldsymbol{L}$ and $\boldsymbol{\overline{L}}$ are both Recursively Enumerable if and only if $\boldsymbol{L}$ is Recursive.

Proof: In one direction, the proof is trivial. That is, if L is Recursive then so is $\overline{L}$. As Recursive Languages $R \subseteq RE$, we have that both L and $\overline{L}$ are in RE.

For the other direction: Let M_1 be the TM for L and M_2 be the TM for $\overline{L}$. Then, if $w \in L$, $M_1(w)$ accepts and if $w \notin L$, $M_2(w)$ accepts.

Closure properties

L and $\overline{L}$ are both Recursively Enumerable if and only if L is Recursive.

Proof: In one direction, the proof is trivial. That is, if L is Recursive then so is $\overline{L}$. As Recursive Languages $R \subseteq RE$, we have that both L and $\overline{L}$ are in RE.

For the other direction: Let M_1 be the TM for L and M_2 be the TM for $\overline{L}$. Then, if $w \in L$, $M_1(w)$ accepts and if $w \notin L$, $M_2(w)$ accepts.

How do we build a Total Turing Machine for L? There is one problem: We can't run M_1 and M_2 one after the other as if some input M_1 gets stuck in an infinite loop and M_2 never gets control.

Closure properties

L and $\overline{L}$ are both Recursively Enumerable if and only if L is Recursive.

Proof: In one direction, the proof is trivial. That is, if L is Recursive then so is $\overline{L}$. As Recursive Languages $R \subseteq RE$, we have that both L and $\overline{L}$ are in RE.

For the other direction: Let M_1 be the TM for L and M_2 be the TM for $\overline{L}$. Then, if $w \in L$, $M_1(w)$ accepts and if $w \notin L$, $M_2(w)$ accepts.

How do we build a Total Turing Machine for L? There is one problem: We can't run M_1 and M_2 one after the other as if some input M_1 gets stuck in an infinite loop and M_2 never gets control.

Idea: We use a time sharing technique – also known as **Dovetailing** to build a Total TM M' for L**.**

Closure properties

L and $\bar{L}$ are both Recursively Enumerable if and only if L is Recursive.

Proof: In one direction, the proof is trivial. That is, if L is Recursive then so is $\bar{L}$. As Recursive Languages $R \subseteq RE$, we have that both L and $\bar{L}$ are in RE.

For the other direction: Let M_1 be the TM for L and M_2 be the TM for $\bar{L}$. Then, if $w \in L$, $M_1(w)$ accepts and if $w \notin L$, $M_2(w)$ accepts.

How do we build a Total Turing Machine for L? There is one problem: We can't run M_1 and M_2 one after the other as if some input M_1 gets stuck in an infinite loop and M_2 never gets control.

Idea: We use a time sharing technique – also known as **Dovetailing** to build a Total TM M' for L**.**

$M' =$

For $i = 1, 2, \cdots$

Run $M_1(w)$ for i steps.

Run $M_2(w)$ for i steps.

If M_1 accepts, M' outputs $ACCEPT$.

If M_2 accepts, M' outputs $REJECT$.

Closure properties

L and $\bar{L}$ are both Recursively Enumerable if and only if L is Recursive.

Proof: In one direction, the proof is trivial. That is, if L is Recursive then so is $\bar{L}$. As Recursive Languages $R \subseteq RE$, we have that both L and $\bar{L}$ are in RE.

For the other direction: Let M_1 be the TM for L and M_2 be the TM for $\bar{L}$. Then, if $w \in L$, $M_1(w)$ accepts and if $w \notin L$, $M_2(w)$ accepts.

How do we build a Total Turing Machine for L? There is one problem: We can't run M_1 and M_2 one after the other as if some input M_1 gets stuck in an infinite loop and M_2 never gets control.

Idea: We use a time sharing technique – also known as **Dovetailing** to build a Total TM M' for L**.**

$M' =$

For $i = 1, 2, \cdots$

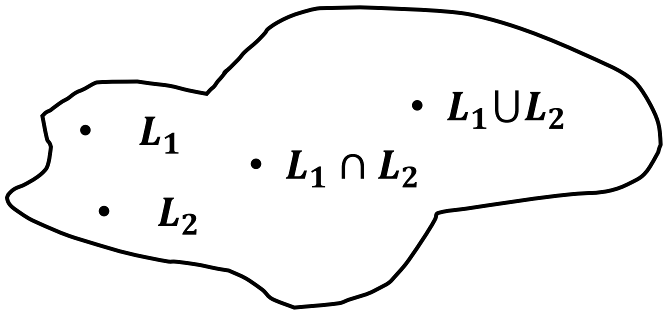
Run $M_1(w)$ for i steps.
Run $M_2(w)$ for i steps.
If M_1 accepts, M' outputs $ACCEPT$.
If M_2 accepts, M' outputs $REJECT$.

L is Recursive

Closure properties

Using Dovetailing it is easy to prove that:

- RE languages are closed under **union and intersection**
    - On input w, run $M_1(w)$ and $M_2(w)$ in parallel using dovetailing
    - **For union**: If either $M_1(w)$ or $M_2(w)$ accepts, $ACCEPT$
    - **For intersection**: If both $M_1(w)$ and $M_2(w)$ accept, $ACCEPT$

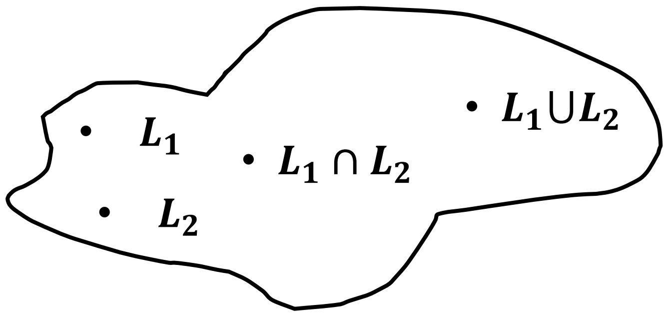


Set of all recursively enumerable Languages

Closure properties

Using Dovetailing it is easy to prove that:

- RE languages are closed under **union and intersection**
    - On input w, run $M_1(w)$ and $M_2(w)$ in parallel using dovetailing
    - **For union**: If either $M_1(w)$ or $M_2(w)$ accepts, $ACCEPT$
    - **For intersection**: If both $M_1(w)$ and $M_2(w)$ accept, $ACCEPT$



Set of all recursively enumerable Languages

RE languages are **NOT closed** under complementation. Why?

Closure properties

Using Dovetailing it is easy to prove that:

- RE languages are **closed under union and intersection**
  - On input w, run $M_1(w)$ and $M_2(w)$ in parallel using dovetailing
  - **For union**: If either $M_1(w)$ or $M_2(w)$ accepts, $ACCEPT$
  - **For intersection**: If both $M_1(w)$ and $M_2(w)$ accept, $ACCEPT$

RE languages are **NOT closed** under complementation. Why?

- Well, we just proved that L and $\overline{L}$ are both Recursively Enumerable, **iff** L is Recursive.
- But we know that there exists problems that are in RE but are not Recursive (e.g: A_{TM}, $HALT_{TM}$,⋯).
- So the complement of such problems are not in RE (e.g.: $\overline{A_{TM}}$, $\overline{HALT_{TM}}$, ⋯).

Closure properties

RE languages are **NOT closed** under complementation. Why?

- Well, we just proved that L and $\overline{L}$ are both Recursively Enumerable, iff L is Recursive.
- But we know that there exists problems that are in RE but not Recursive (e.g.: A_{TM}, $HALT_{TM}, \cdots$).
- So the complement of such problems are not in RE (e.g.: $\overline{HALT}_{TM}, \cdots$).

Suppose $L \in RE$ and M be the TM which recognizes L, i.e. $\mathcal{L}(M) = L$.

What if we try to build a TM $\overline{M}$ that outputs the opposite of M?

Closure properties

RE languages are **NOT closed** under complementation. Why?

- Well, we just proved that L and $\overline{L}$ are both Recursively Enumerable, iff L is Recursive.
- But we know that there exists problems that are in RE but not Recursive (e.g.: A_{TM}, $HALT_{TM}$,⋯).
- So the complement of such problems are not in RE (e.g.: $\overline{HALT_{TM}}$, ⋯).

Suppose $L \in RE$ and M be the TM which recognizes L, i.e. $\mathcal{L}(M) = L$.

What if we try to build a TM $\overline{M}$ that outputs the opposite of M?

$\overline{M}$ = On input w

Run $M(w)$.

If $M(w)$ accepts, output *REJECT*

If $M(w)$ rejects, output $ACCEPT$

If $M(w)$ loops, ……………………….

Closure properties

RE languages are **NOT closed** under complementation. Why?

- Well, we just proved that L and $\overline{L}$ are both Recursively Enumerable, iff L is Recursive.
- But we know that there exists problems that are in RE but not Recursive (e.g.: A_{TM}, $HALT_{TM}$,⋯).
- So the complement of such problems are not in RE (e.g.: $\overline{HALT_{TM}}$, ⋯).

Suppose $L \in RE$ and M be the TM which recognizes L, i.e. $\mathcal{L}(M) = L$.

What if we try to build a TM $\overline{M}$ that outputs the opposite of M?

$\overline{M}$ = On input w
Run $M(w)$.
If $M(w)$ accepts, output $REJECT$
If $M(w)$ rejects, output $ACCEPT$
If $M(w)$ loops, ……………………….

So if,
w $\notin L$, $i.e.$, $w \in \overline{L}$, $\overline{M}$ outputs $ACCEPT$ or loops forever
$w \in L$, $i.e.$, $w \notin \overline{L}$, $\overline{M}$ outputs $REJECT$

Closure properties

RE languages are **NOT closed** under complementation. Why?

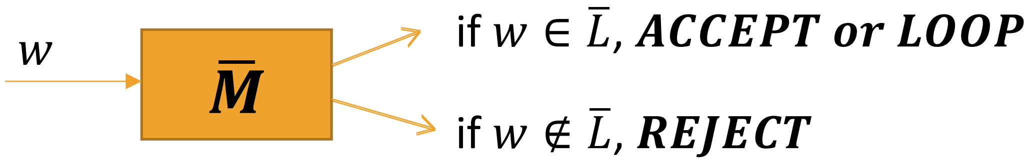
$\bar{M}$ = On input w

Run $M(w)$.

If $M(w)$ accepts, output REJECT

If $M(w)$ rejects, output $ACCEPT$

If $M(w)$ loops, ……………………….



Closure properties

RE languages are **NOT closed** under complementation. Why?

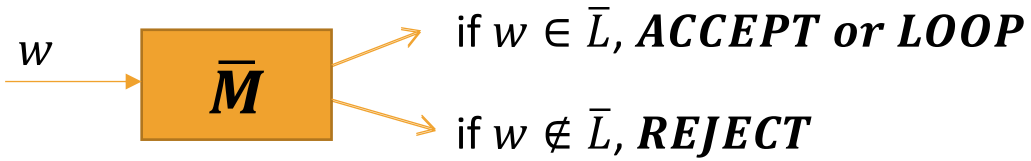
$\bar{M}$ = On input w

Run $M(w)$.

If $M(w)$ accepts, output REJECT

If $M(w)$ rejects, output $ACCEPT$

If $M(w)$ loops, ……………………….



Co-Recursively Enumerable Language/co-Turing Recognizable (Co-RE/$\overline{RE}$/nRE): A language C is **Co-Recursively Enumerable (co-RE/$\overline{RE}$/nRE) or Co-Turing Recognizable** if

$\forall \omega \in C, M(\omega)$ **doesn't reject** (accepts or loops forever)

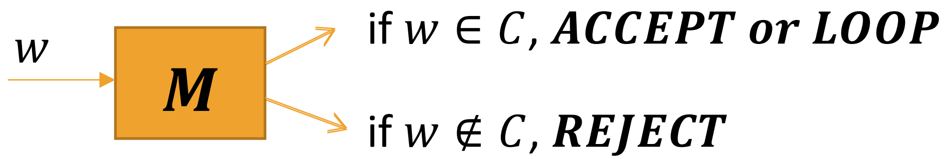
$\forall \omega \notin C, M(\omega)$ **rejects**

If $L \in RE$, $\bar{L} \in co\text{-}RE$ and vice versa

$co\text{-}RE$

Co-Recursively Enumerable Language (Co-RE/$\overline{RE}$/nRE): C ∈ **Co-RE** if

$\forall \omega \in C, M(\omega)$ **doesn't reject**
$\forall \omega \notin C, M(\omega)$ **rejects**

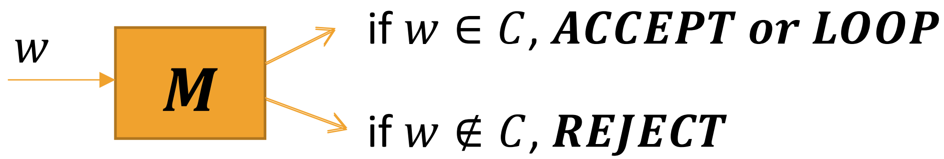


- $\boldsymbol{RE}$: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- $\boldsymbol{co\text{-}RE}$: May loop when $w \in L$. Halts and rejects if $w \notin L$.

$co\text{-}RE$

Co-Recursively Enumerable Language (Co-RE/$\overline{RE}$/nRE): C ∈ **Co-RE** if

$$\forall\omega \in C, M(\omega) \textbf{ doesn't reject}$$
$$\forall\omega \notin C, M(\omega) \textbf{ rejects}$$



- $\boldsymbol{RE}$: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- $\boldsymbol{co\text{-}RE}$: May loop when $w \in L$. Halts and rejects if $w \notin L$.

Note: Every Recursive Language $\boldsymbol{R}$, is both in $\boldsymbol{RE}$ and $\boldsymbol{co\text{-}RE}$, i.e. $\boldsymbol{R \subseteq RE \cap co\text{-}RE}$

$co\text{-}RE$

Co-Recursively Enumerable Language (Co-RE/$\overline{RE}$/nRE): C $\in$ **Co-RE** if

$$\forall \omega \in C, M(\omega) \textbf{ doesn't reject}$$
$$\forall \omega \notin C, M(\omega) \textbf{ rejects}$$

if $w \in C$, $\boldsymbol{ACCEPT\ or\ LOOP}$

if $w \notin C$, $\boldsymbol{REJECT}$

- $\boldsymbol{RE}$: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- $\boldsymbol{co\text{-}RE}$: May loop when $w \in L$. Halts and rejects if $w \notin L$.

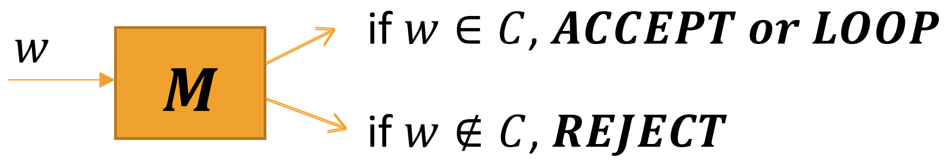
Note: Every Recursive Language R, is both in RE and $co\text{-}RE$, i.e. $R \subseteq RE \cap co\text{-}RE$

Is $R = RE \cap co\text{-}RE$?

$co\text{-}RE$

Co-Recursively Enumerable Language (Co-RE/$\overline{RE}$/nRE): C ∈ **Co-RE** if

$\forall \omega \in C, M(\omega)$ **doesn't reject**
$\forall \omega \notin C, M(\omega)$ **rejects**



- $\boldsymbol{RE}$: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- $\boldsymbol{co\text{-}RE}$: May loop when $w \in L$. Halts and rejects if $w \notin L$.

Note: Every Recursive Language $\boldsymbol{R}$, is both in $\boldsymbol{RE}$ and $\boldsymbol{co\text{-}RE}$, i.e. $\boldsymbol{R \subseteq RE \cap co\text{-}RE}$

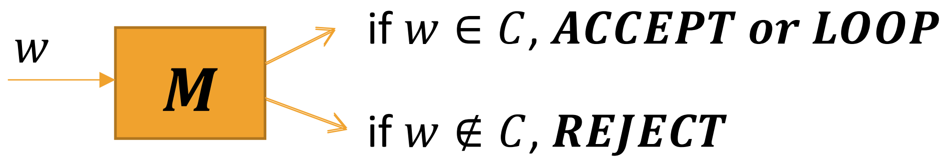
Is $\boldsymbol{R = RE \cap co\text{-}RE}$?

We have to prove the following: **If $\boldsymbol{L \in RE}$ and $\boldsymbol{L \in co\text{-}RE}$, then $\boldsymbol{L}$ is Recursive**

$co\text{-}RE$

Co-Recursively Enumerable Language (Co-RE/$\overline{RE}$/nRE): C $\in$ **Co-RE** if

$\forall \omega \in C, M(\omega)$ **doesn't reject**
$\forall \omega \notin C, M(\omega)$ **rejects**



- $\boldsymbol{RE}$: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- $\boldsymbol{co\text{-}RE}$: May loop when $w \in L$. Halts and rejects if $w \notin L$.

Note: Every Recursive Language $\boldsymbol{R}$, is both in $\boldsymbol{RE}$ and $\boldsymbol{co\text{-}RE}$, i.e. $\boldsymbol{R \subseteq RE \cap co\text{-}RE}$

Is $\boldsymbol{R = RE \cap co\text{-}RE}$?

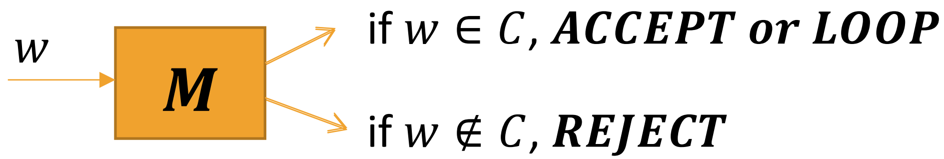
We have to prove the following: **If $\boldsymbol{L \in RE}$ and $\boldsymbol{L \in co\text{-}RE}$, then $\boldsymbol{L}$ is Recursive**

Proof: Let M be a TM such that $L(M) = L$. As $L \in co\text{-}RE$, there also exists a $\overline{M}$ that halts and outputs reject whenever $w \notin L$.

$co\text{-}RE$

Co-Recursively Enumerable Language (Co-RE/$\overline{RE}$/nRE): C ∈ **Co-RE** if

$\forall \omega \in C, M(\omega)$ **doesn't reject**
$\forall \omega \notin C, M(\omega)$ **rejects**



- $\boldsymbol{RE}$: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- $\boldsymbol{co\text{-}RE}$: May loop when $w \in L$. Halts and rejects if $w \notin L$.

Note: Every Recursive Language $\boldsymbol{R}$, is both in $\boldsymbol{RE}$ and $\boldsymbol{co\text{-}RE}$, i.e. $\boldsymbol{R \subseteq RE \cap co\text{-}RE}$

Is $\boldsymbol{R = RE \cap co\text{-}RE}$?

We have to prove the following: **If $\boldsymbol{L \in RE}$ and $\boldsymbol{L \in co\text{-}RE}$, then $\boldsymbol{L}$ is Recursive**

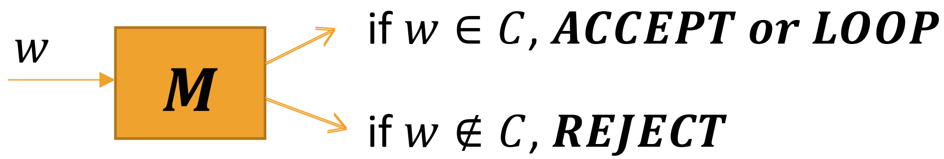
Proof: Let M be a TM such that $L(M) = L$. As $L \in co\text{-}RE$, there also exists a $\bar{M}$ that halts and outputs reject whenever $w \notin L$. We shall construct a Total Turing Machine D by using M and $\bar{M}$.

We need to run M and $\bar{M}$ in parallel using dovetailing.

$co\text{-}RE$

Co-Recursively Enumerable Language (Co-RE/$\overline{RE}$/nRE): C ∈ **Co-RE** if

$$\forall \omega \in C, M(\omega) \textbf{ doesn't reject}$$
$$\forall \omega \notin C, M(\omega) \textbf{ rejects}$$



- $\boldsymbol{RE}$: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- $\boldsymbol{co\text{-}RE}$: May loop when $w \in L$. Halts and rejects if $w \notin L$.

Note: Every Recursive Language $\boldsymbol{R}$, is both in $\boldsymbol{RE}$ and $\boldsymbol{co\text{-}RE}$, i.e. $\boldsymbol{R \subseteq RE \cap co\text{-}RE}$

Is $\boldsymbol{R = RE \cap co\text{-}RE}$?

We have to prove the following: **If $\boldsymbol{L \in RE}$ and $\boldsymbol{L \in co\text{-}RE}$, then $\boldsymbol{L}$ is Recursive**

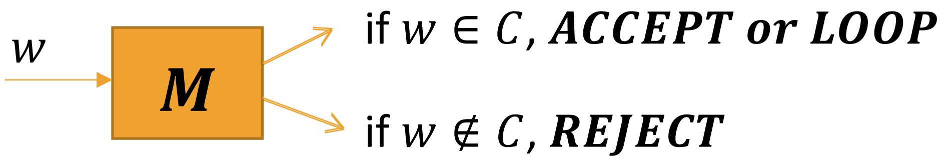
Proof: Let M be a TM such that $L(M) = L$ which halts and accepts whenever $w \in L$. As $L \in co\text{-}RE$, there also exists a $\overline{M}$ that halts and outputs reject whenever $w \notin L$. We shall construct a Total Turing Machine D by using M and $\overline{M}$.

D = On input w
Run $M(w)$ and $\overline{M}(w)$ in parallel
If $M(w)$ accepts, output $ACCEPT$
If $\overline{M}(w)$ rejects, output $REJECT$

$co\text{-}RE$

Co-Recursively Enumerable Language (Co-RE/$\overline{RE}$/nRE): $C \in$ **Co-RE** if

$\forall \omega \in C, M(\omega)$ **doesn't reject**
$\forall \omega \notin C, M(\omega)$ **rejects**



- $\boldsymbol{RE}$: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- $\boldsymbol{co\text{-}RE}$: May loop when $w \in L$. Halts and rejects if $w \notin L$.

Note: Every Recursive Language $\boldsymbol{R}$, is both in $\boldsymbol{RE}$ and $\boldsymbol{co\text{-}RE}$, i.e. $\boldsymbol{R \subseteq RE \cap co\text{-}RE}$

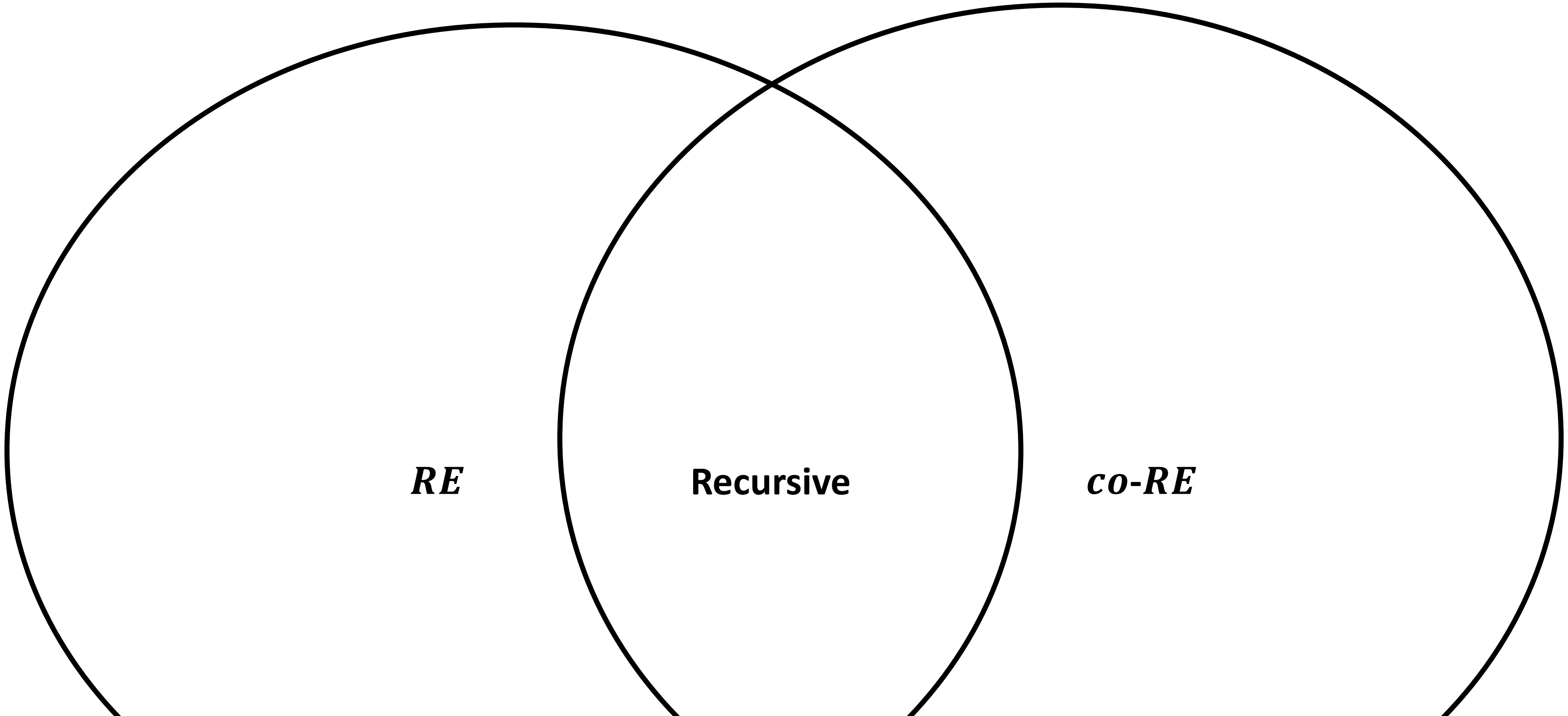
Is $\boldsymbol{R = RE \cap co\text{-}RE}$?

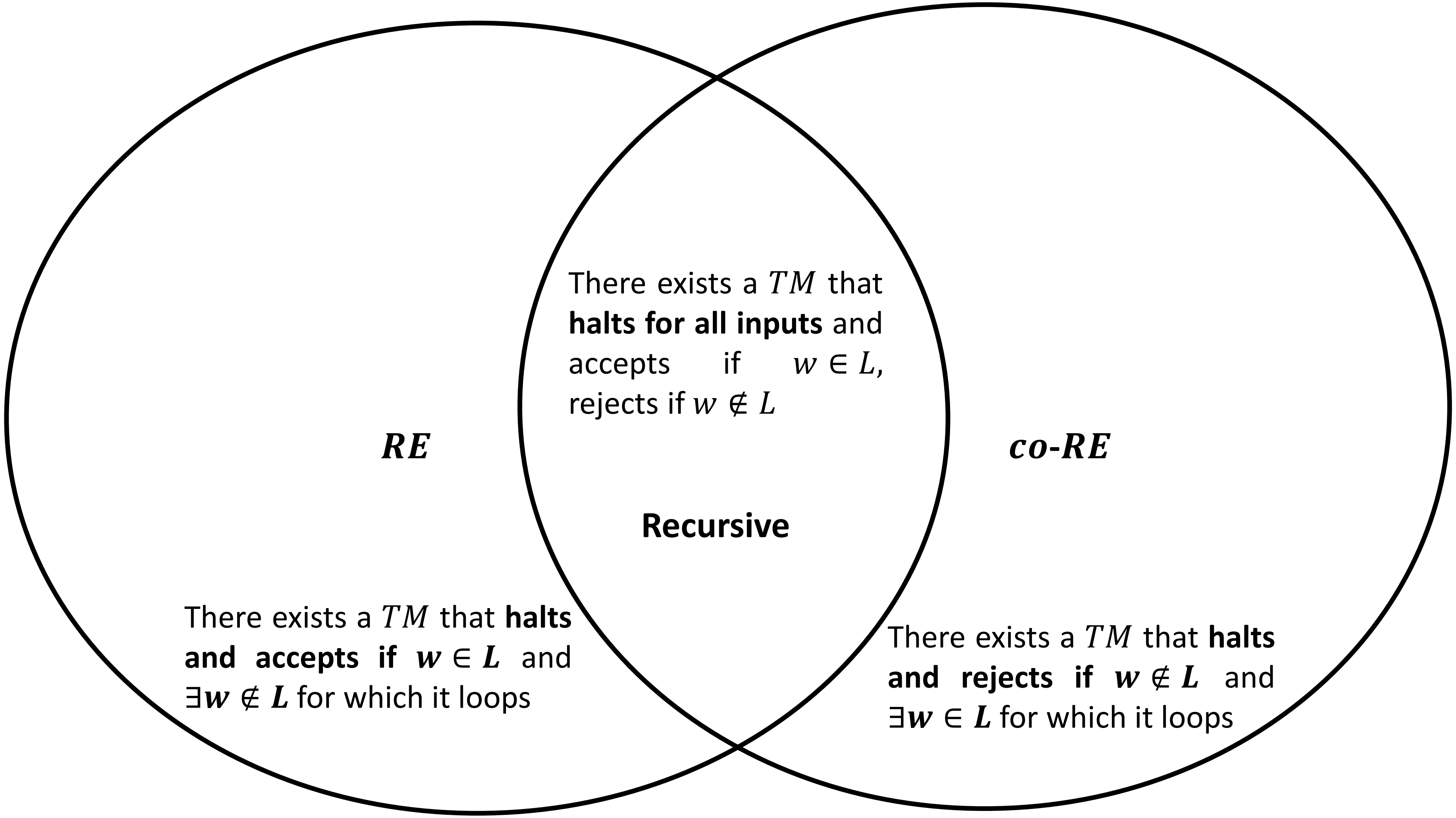
We have to prove the following: **If $\boldsymbol{L \in RE}$ and $\boldsymbol{L \in co\text{-}RE}$, then $\boldsymbol{L}$ is Recursive**

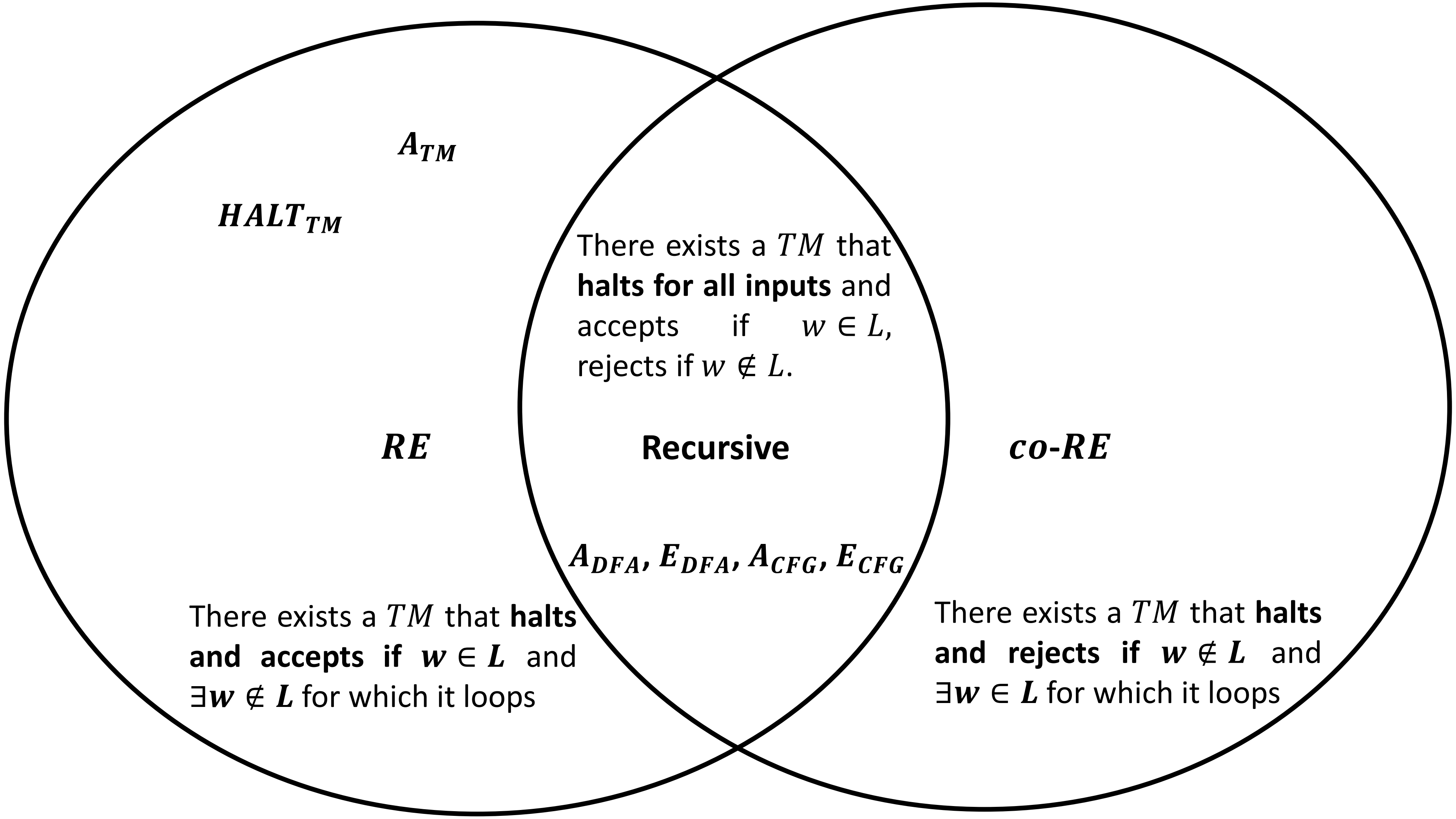
Proof: Let M be a TM such that $L(M) = L$ which halts and accepts whenever $w \in L$. As $L \in co\text{-}RE$, there also exists a $\overline{M}$ that halts and outputs reject whenever $w \notin L$. We shall construct a Total Turing Machine D by using M and $\overline{M}$.

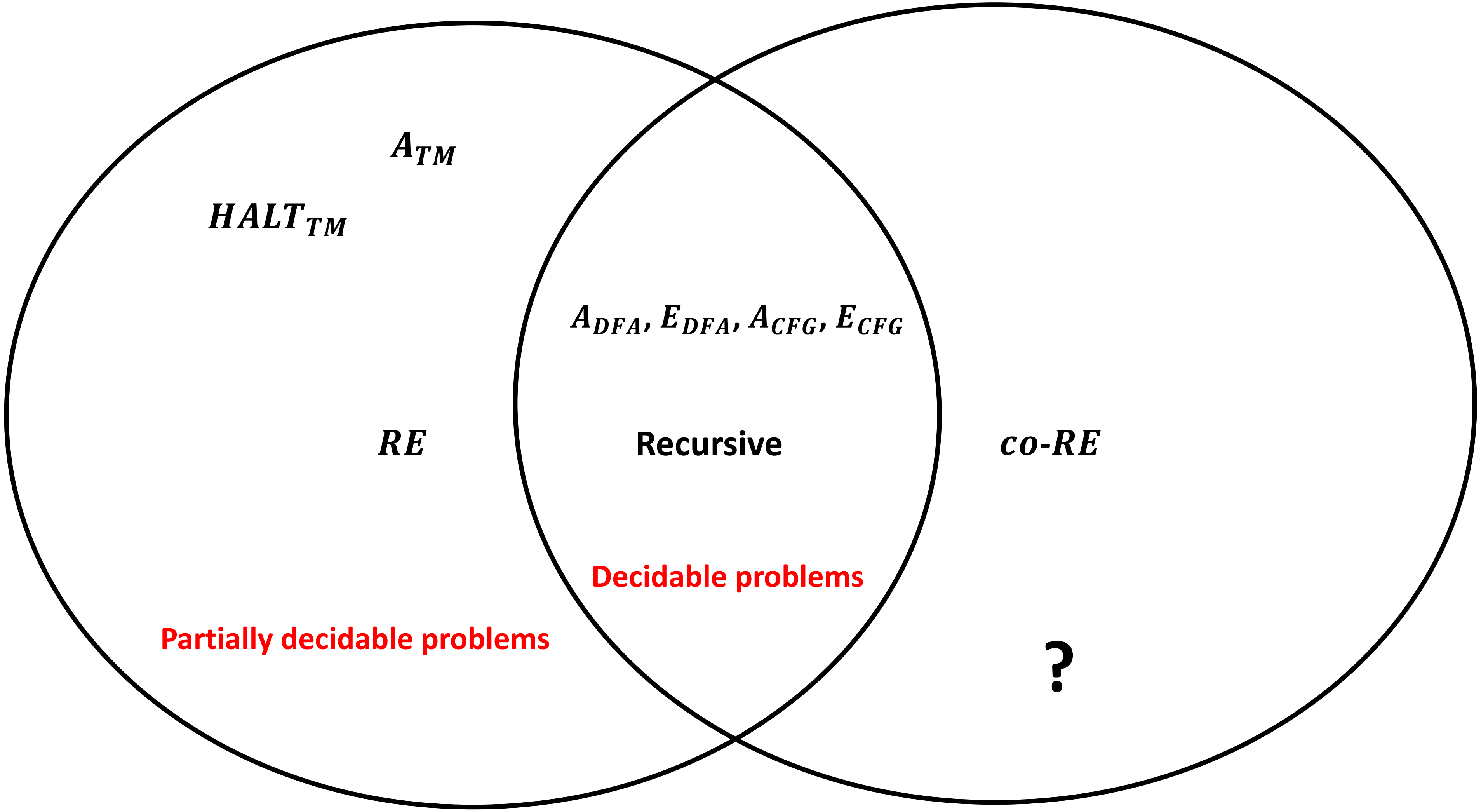
$D =$ On input w
Run $M(w)$ and $\overline{M}(w)$ in parallel
If $M(w)$ accepts, output $ACCEPT$
If $\overline{M}(w)$ rejects, output $REJECT$

So $\boldsymbol{R = RE \cap co\text{-}RE}$


RE
Recursive
$co\text{-}RE$


There exists a TM that **halts for all inputs** and accepts if $w \in L$, rejects if $w \notin L$
$\boldsymbol{RE}$
$\boldsymbol{co\text{-}RE}$
Recursive
There exists a TM that **halts and accepts if** $\boldsymbol{w \in L}$ and $\exists \boldsymbol{w} \notin \boldsymbol{L}$ for which it loops
There exists a TM that **halts and rejects if** $\boldsymbol{w \notin L}$ and $\exists \boldsymbol{w} \in \boldsymbol{L}$ for which it loops

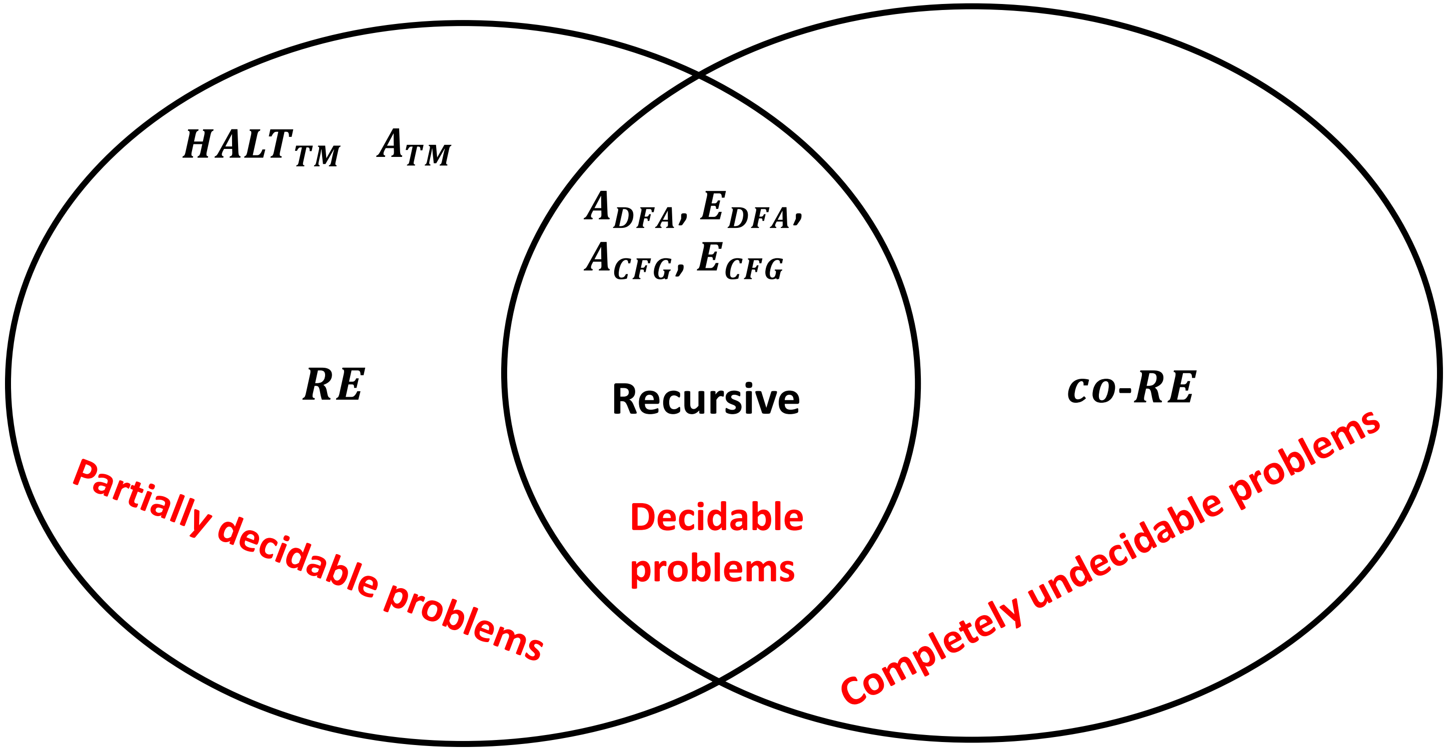

A_{TM}
$HALT_{TM}$
There exists a TM that **halts for all inputs** and accepts if $w \in L$, rejects if $w \notin L$.
RE
Recursive
$co\text{-}RE$
$A_{DFA}, E_{DFA}, A_{CFG}, E_{CFG}$
There exists a TM that **halts and accepts if $w \in L$** and $\exists w \notin L$ for which it loops
There exists a TM that **halts and rejects if $w \notin L$** and $\exists w \in L$ for which it loops


A_{TM}
$HALT_{TM}$
$A_{DFA}, E_{DFA}, A_{CFG}, E_{CFG}$
RE
Recursive
$co\text{-}RE$
Decidable problems
Partially decidable problems
?

Closure properties

Completely undecidable languages: Languages L for which there exists at least one instance $w \in L$, for which the TM enters into an infinite loop.

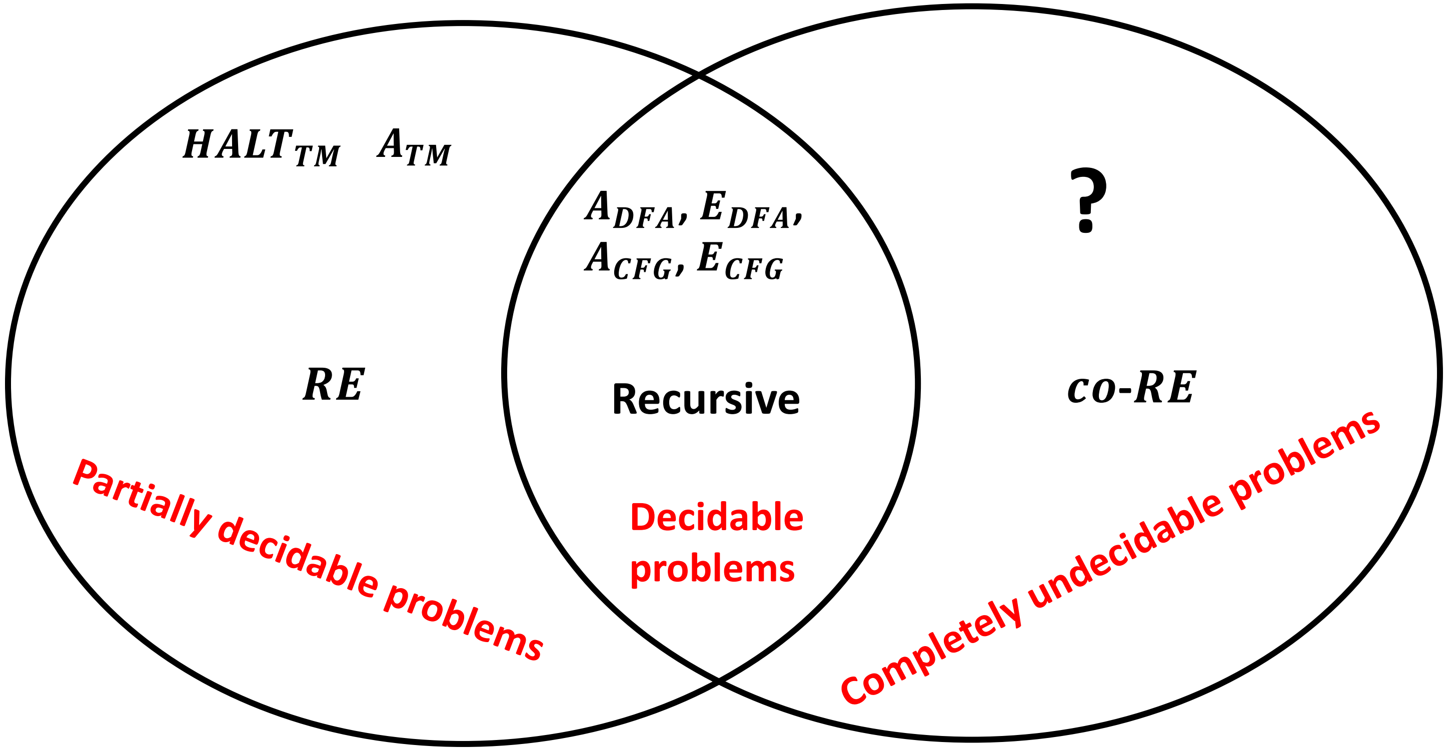
So, languages **that are in $\boldsymbol{co\text{-}RE}$ but are not recursive** are **completely undecidable.**



Closure properties

Completely undecidable languages: Languages L for which there exists at least one instance $w \in L$, for which the TM enters into an infinite loop.

So, languages **that are in $co\text{-}RE$ but are not recursive** are **completely undecidable.**



Closure properties

Completely undecidable languages: Languages L for which there exists at least one instance $w \in L$, for which the TM enters into an infinite loop.

If $L \in RE$ but is not Recursive (partially decidable), then $\bar{L} \in co\text{-}RE$ but is not recursive. So, Complement of any partially decidable language is completely undecidable

- E.g.: $A_{TM} \in RE$ and so $\overline{A_{TM}} \in co\text{-}RE$ and is **completely undecidable**

$\boldsymbol{A_{TM} = \{\langle M, w\rangle | M \text{ accepts input } w\}}$

$\boldsymbol{\overline{A_{TM}} = \{\langle M, w\rangle | M \text{ doesn't accept input } w\}}$

Closure properties

Completely undecidable languages: Languages L for which there exists at least one instance $w \in L$, for which the TM enters into an infinite loop.

If $\boldsymbol{L \in RE}$ but is not Recursive (partially decidable), then $\boldsymbol{\bar{L} \in co\text{-}RE}$ but is not recursive. So Complement of any partially decidable language is completely undecidable

- E.g.: $A_{TM} \in RE$ and so $\overline{A_{TM}} \in co\text{-}RE$ and is **completely undecidable**

$\boldsymbol{A_{TM} = \{\langle M, w\rangle | M}$ **accepts input** $\boldsymbol{w\}}$

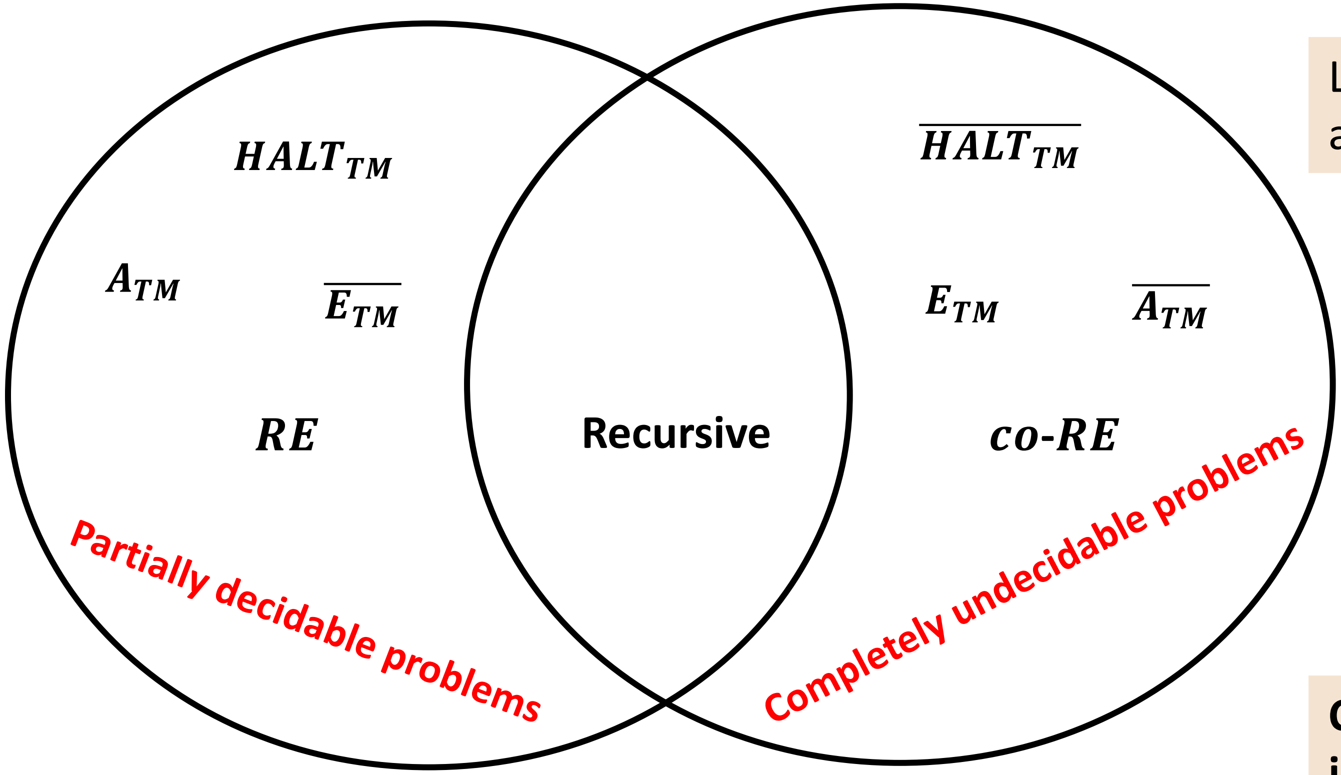
$\boldsymbol{\overline{A_{TM}} = \{\langle M, w\rangle | M}$ **doesn't accept input** $\boldsymbol{w\}}$

- Similarly, $\boldsymbol{\overline{HALT_{TM}}}$ **is also completely undecidable**

$\boldsymbol{\overline{HALT_{TM}} = \{\langle M, w\rangle | M}$ **doesn't halt on input** $\boldsymbol{w\}}$

Closure properties

Completely undecidable languages: Languages L for which there exists at least one instance $w \in L$, for which the TM enters into an infinite loop.



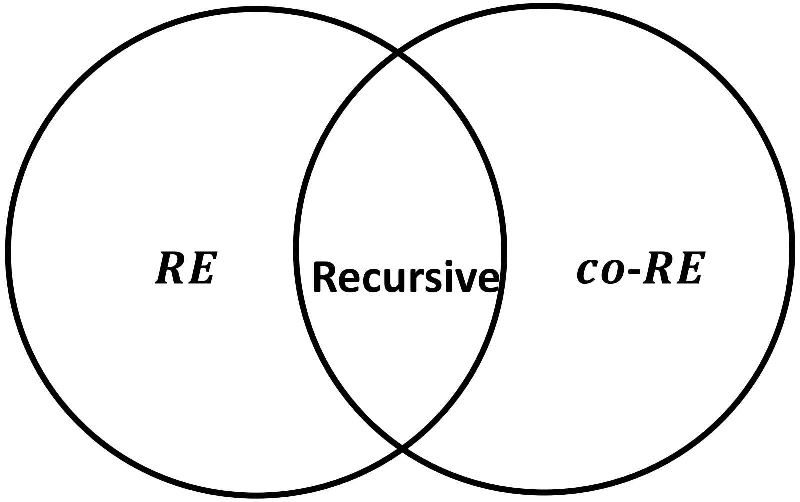
Languages **that are in $co\text{-}RE$ but are not recursive** are **completely undecidable.**

Complement of any partially decidable language is completely undecidable.

Summing up

We have the following:

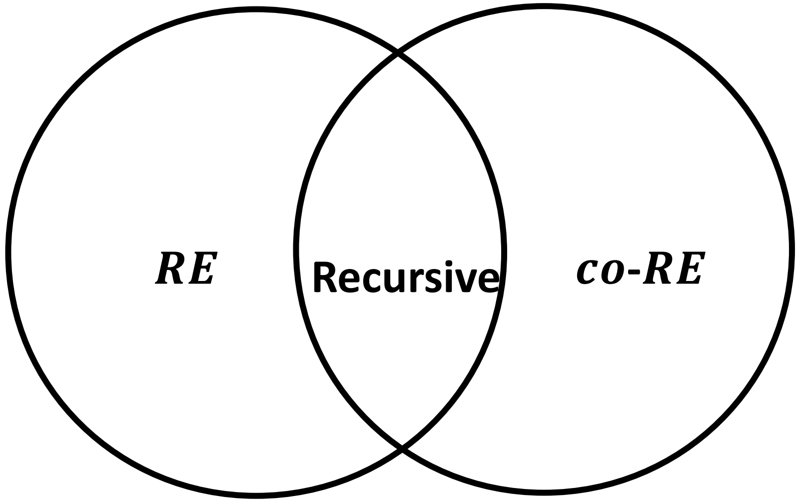
- **Recursive Languages are closed under complement, union & intersection**
- **RE is closed under union & intersection but not complement**



Summing up

We have the following:

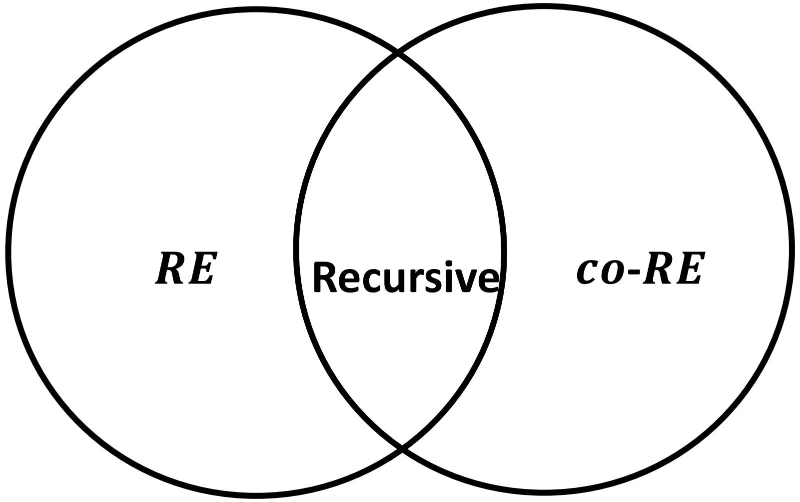
- **Recursive Languages are closed under complement, union & intersection**
- $\boldsymbol{RE}$ **is closed under union & intersection but not complement**
- $\boldsymbol{L \in RE}$ **and** $\boldsymbol{\bar{L} \in RE}$**, iff** $\boldsymbol{L}$ **is Recursive**.
- **If** $\boldsymbol{L \in RE}$ **then** $\boldsymbol{\bar{L} \in co\text{-}RE}$**.**
- **If** $\boldsymbol{L \in co}$**-RE then** $\boldsymbol{\bar{L} \in RE}$**.**



Summing up

We have the following:

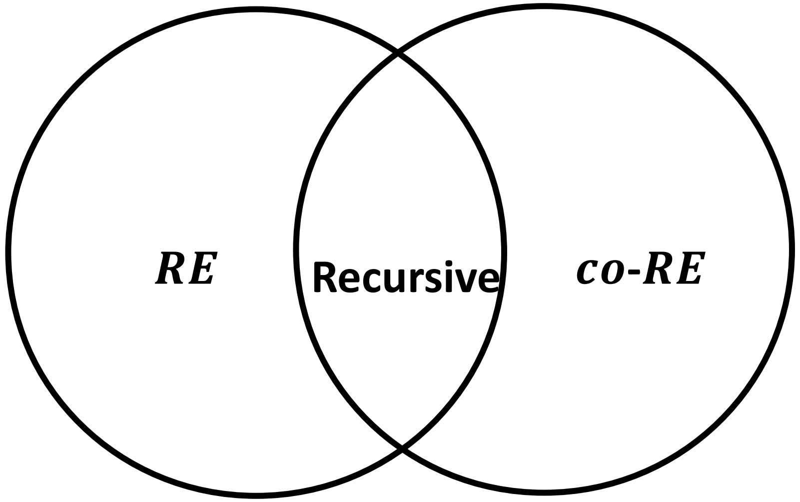
- **Recursive Languages are closed under complement, union & intersection**
- $\boldsymbol{RE}$ **is closed under union & intersection but not complement**
- $\boldsymbol{L \in RE}$ **and** $\boldsymbol{\bar{L} \in RE}$**, iff** $\boldsymbol{L}$ **is Recursive**.
- **If** $\boldsymbol{L \in RE}$ **then** $\boldsymbol{\bar{L} \in co\text{-}RE}$**.**
- **If** $\boldsymbol{L \in co}$**-RE then** $\boldsymbol{\bar{L} \in RE}$**.**
- $\boldsymbol{R = RE \cap co\text{-}RE}$



Summing up

We have the following:

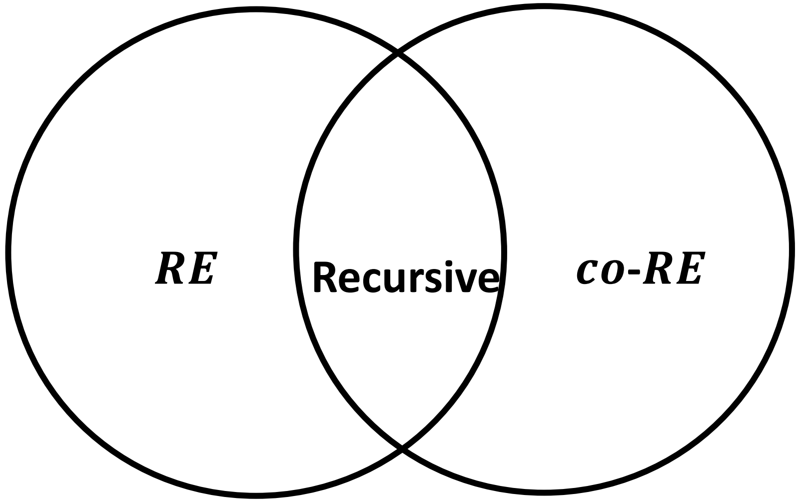
- **Recursive Languages are closed under complement, union & intersection**
- $\boldsymbol{RE}$ **is closed under union & intersection but not complement**
- $\boldsymbol{L \in RE}$ **and** $\boldsymbol{\bar{L} \in RE}$**, iff** $\boldsymbol{L}$ **is Recursive**.
- **If** $\boldsymbol{L \in RE}$ **then** $\boldsymbol{\bar{L} \in co\text{-}RE}$**.**
- **If** $\boldsymbol{L \in co}$**-RE then** $\boldsymbol{\bar{L} \in RE}$**.**
- $\boldsymbol{R = RE \cap co\text{-}RE}$
- **Co-RE languages are closed under union and intersection but not complement**



Summing up

We have the following:

- **Recursive Languages are closed under complement, union & intersection**
- $\boldsymbol{RE}$ **is closed under union & intersection but not complement**
- $\boldsymbol{L \in RE}$ **and** $\boldsymbol{\overline{L} \in RE}$**, iff** $\boldsymbol{L}$ **is Recursive**.
- **If** $\boldsymbol{L \in RE}$ **then** $\boldsymbol{\overline{L} \in co\text{-}RE}$**.**
- **If** $\boldsymbol{L \in co}$**-RE then** $\boldsymbol{\overline{L} \in RE}$**.**
- $\boldsymbol{R = RE \cap co\text{-}RE}$
- **Co-RE languages are closed under union and intersection but not complement**



Note that there are languages outside of $\boldsymbol{RE \cup co\text{-}RE}$**.**

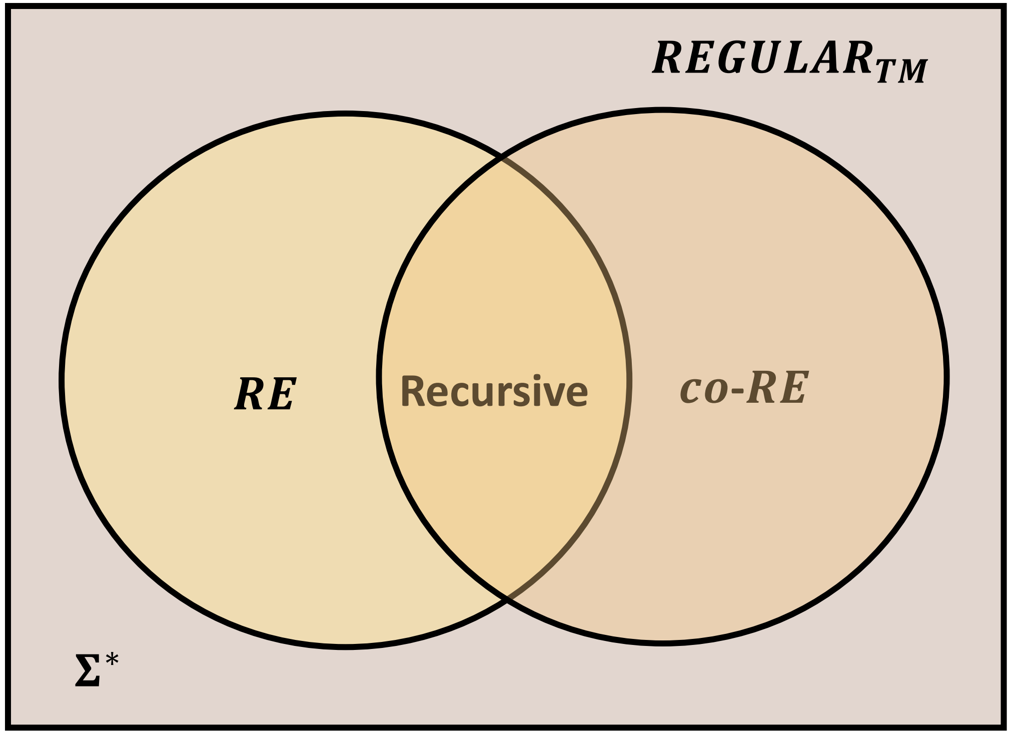
Summing up

We have the following:

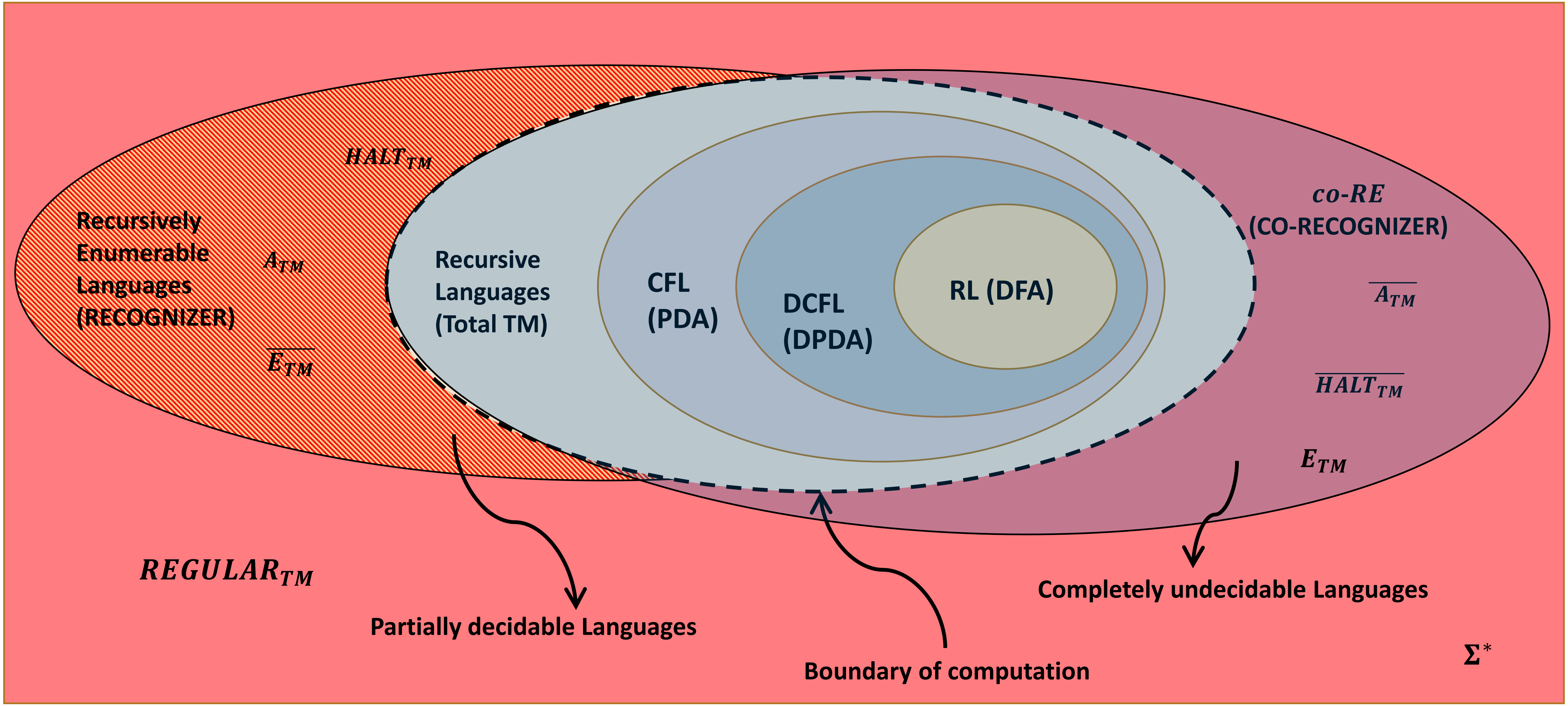
- **Recursive Languages are closed under complement, union & intersection**
- $\boldsymbol{RE}$ **is closed under union & intersection but not complement**
- $\boldsymbol{L \in RE}$ **and** $\boldsymbol{\bar{L} \in RE}$**, iff** $\boldsymbol{L}$ **is Recursive**.
- **If** $\boldsymbol{L \in RE}$ **then** $\boldsymbol{\bar{L} \in co\text{-}RE.}$
- **If** $\boldsymbol{L \in co}$**-RE then** $\boldsymbol{\bar{L} \in RE.}$
- $\boldsymbol{R = RE \cap co\text{-}RE}$
- **If** $\boldsymbol{L \in RE}$ **but is not Recursive, then** $\boldsymbol{L}$ **is partially decidable**
- **If** $\boldsymbol{L \in co\text{-}RE}$ **but is not Recursive, then** $\boldsymbol{L}$ **is completely undecidable.**

Note that there are languages outside of $\boldsymbol{RE \cup co\text{-}RE.}$

E.g.: $\boldsymbol{REGULAR_{TM}} = \{\langle \boldsymbol{M} \rangle | \boldsymbol{L}(\boldsymbol{M})$ **is regular**$\}$



Everything in one slide



The Road ahead to Complexity Theory…

- We finished up by looking at problems that are decidable/undecidable.

- There are many things that I couldn't cover:

  - Several cool problems that can be proven to be decidable/undecidable and classified to be in $R, RE, co\text{-}RE$ etc
  - Recursion Theorem, Rice's Theorem

- Problems that are not computable are highly likely to never be solved on feasible computational devices.

- In how much time/space can **computable problems** be solved in? Complexity Theory: classify problems according to their hardness.

- Long-standing open problems waiting to be solved!

- **New computing models:** Quantum computers model how nature computes at the fundamental level: provably faster than classical machines for several problems and most likely violates the Extended Church Turing Thesis.

Thank You!