

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



Quick Recap

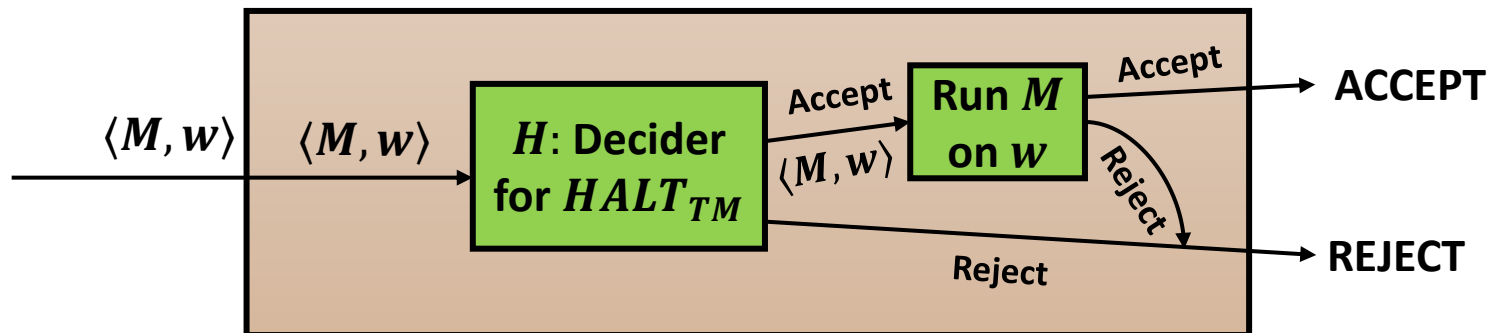
$HALT_{TM} = \{\langle M, w \rangle \mid M \text{ halts on input } w\}$. Is $HALT_{TM}$ decidable?

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.

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

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

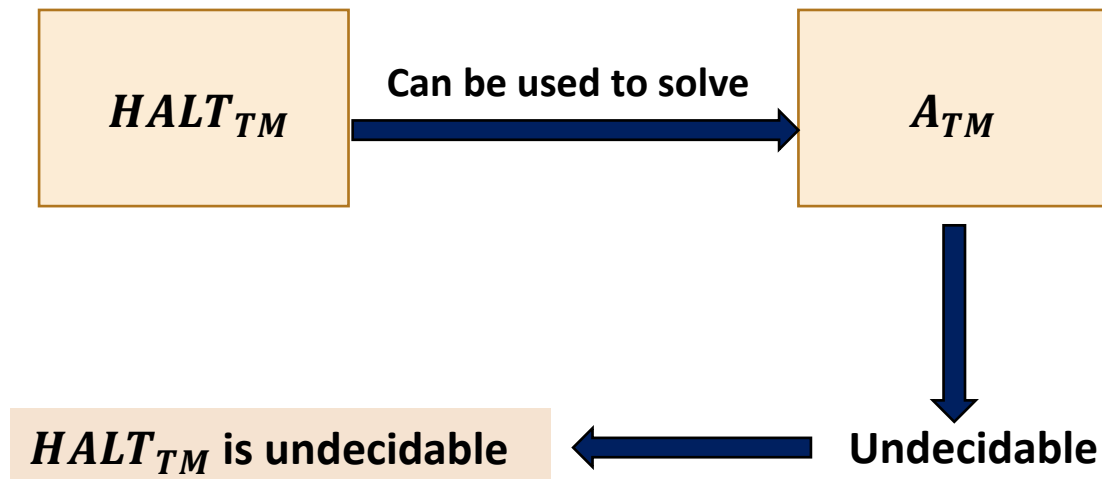
$A =$ On input $\langle M, w \rangle$
Run $H(\langle M, w \rangle)$
If H rejects, output *REJECT*
If H accepts,
Run $M(w)$
If $M(w)$ accepts, output *ACCEPT*
If $M(w)$ rejects, output *REJECT*



Quick Recap

Generally,

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



Suppose, $A \leq B$

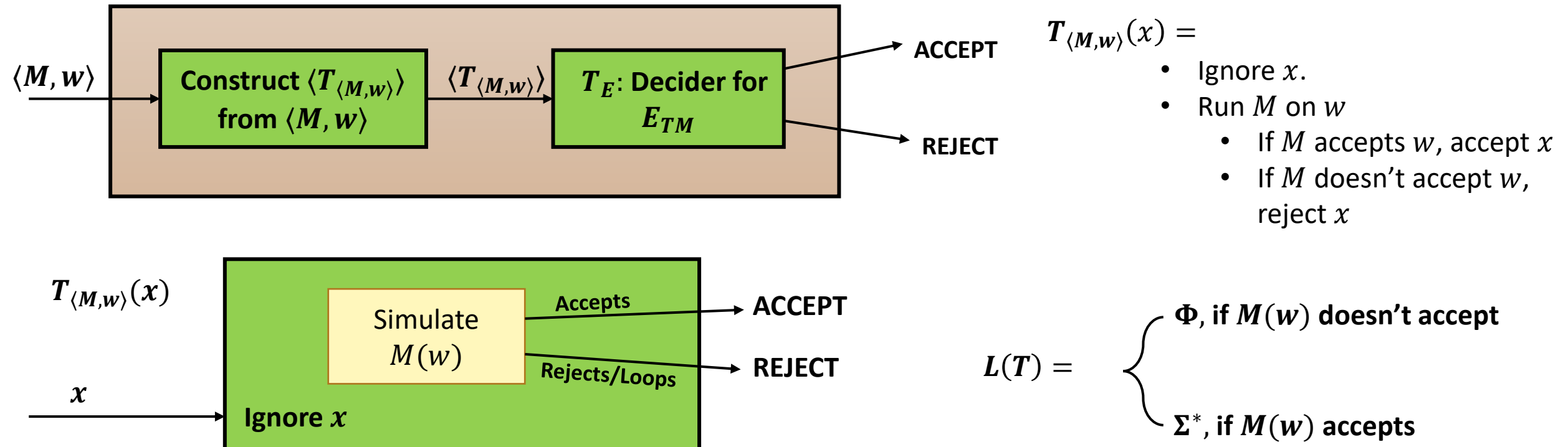
- If A is undecidable then B is undecidable.
- **If B is decidable then A is decidable.**

Quick Recap

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

NO! $\overline{A_{TM}} \leq E_{TM}$

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



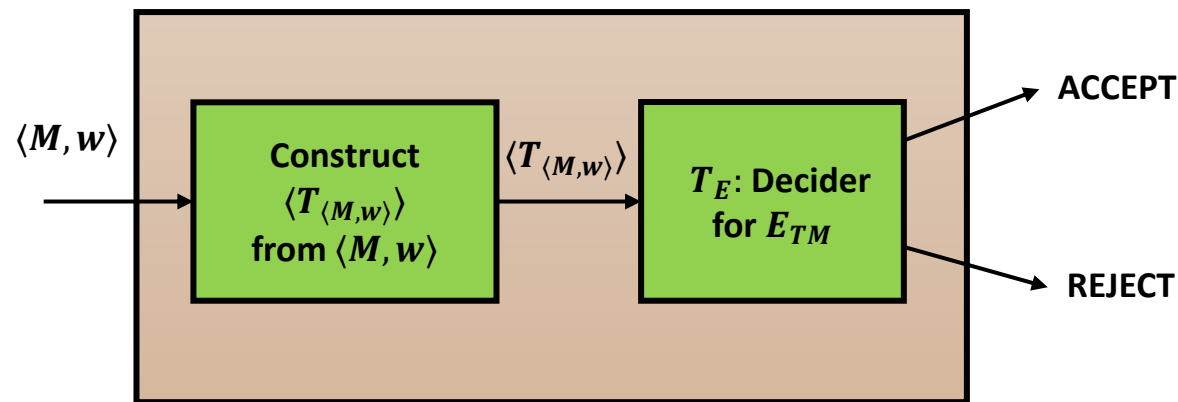
Quick Recap

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

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

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

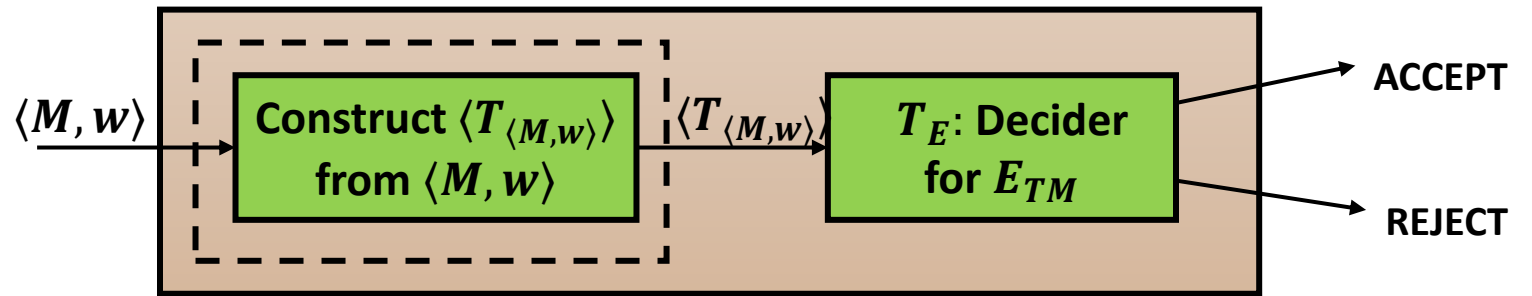
- Construct $\langle T_{\langle M, w \rangle} \rangle$, the encoding of $T_{\langle M, w \rangle}$ such that for any input x it works as follows:
 - Ignore x .
 - Run M on w
 - If M accepts w , accept x
 - If M rejects w , reject x
- Send $\langle T_{\langle M, w \rangle} \rangle$ to T_E and Output
ACCEPT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



- $\overline{A_{TM}} \leq E_{TM}$
- $\overline{A_{TM}}$ is undecidable
- E_{TM} is undecidable!

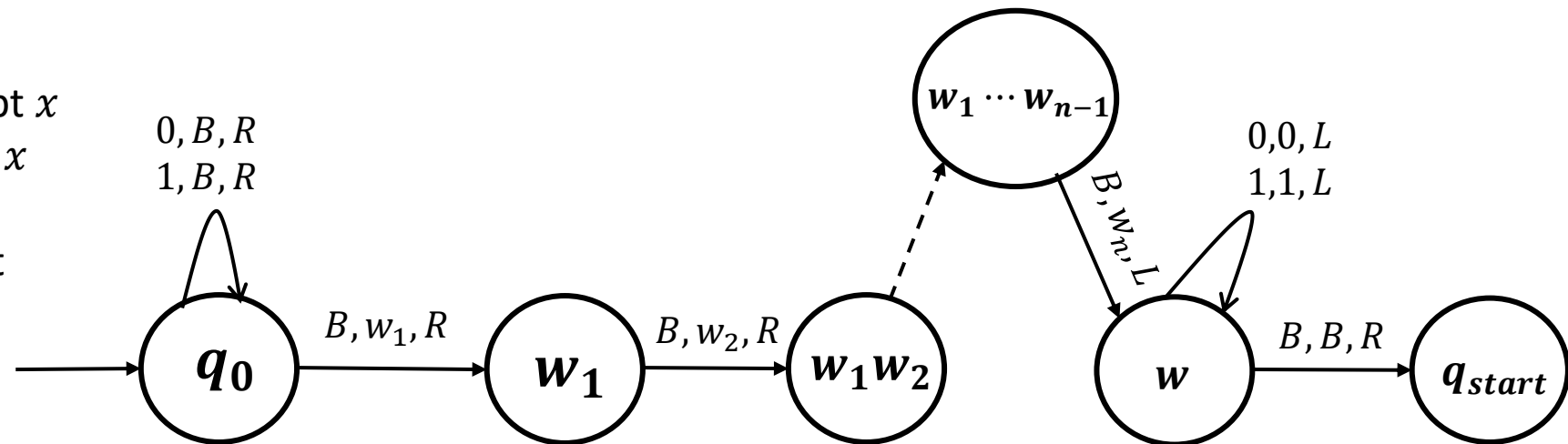
Quick Recap

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

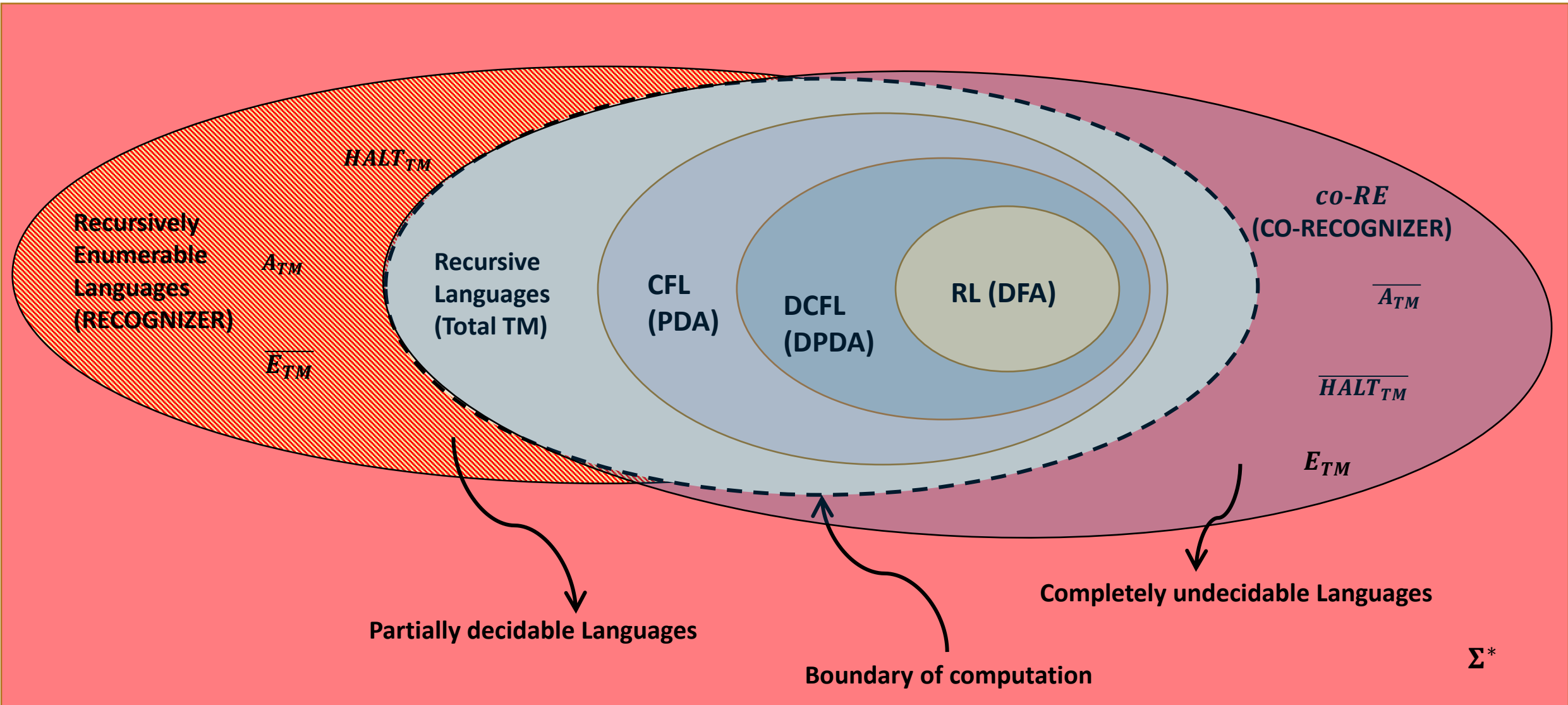


$N(\langle M, w \rangle) =$

- 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



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 M , 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 M , 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

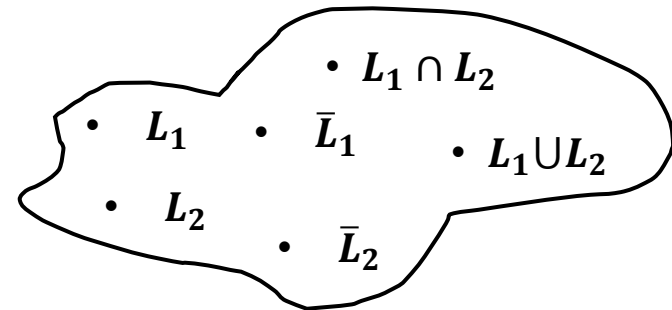
Are Recursive languages **closed under complementation**? If some language R is recursive, is \bar{R} also recursive?

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

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 .

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.

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 for some input M_1 gets stuck in an infinite loop and M_2 never gets control.

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 for 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 for 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 $i = 1, 2, \dots$

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 for 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 $i = 1, 2, \dots$

Run $M_1(w)$ for i steps.

Run $M_2(w)$ for i steps.

If M_1 accepts, M' outputs *ACCEPT*.

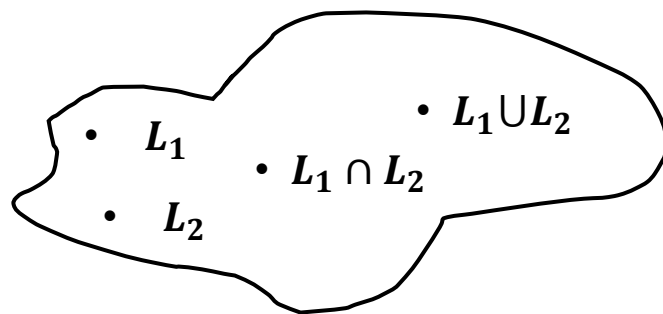
If M_2 accepts, M' outputs *REJECT*.

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*



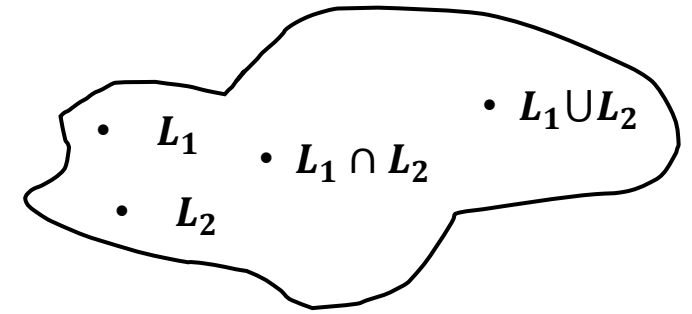
Set of all recursively enumerable Languages

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*

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



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*

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

- Well, we just proved that L and \bar{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 \bar{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 $TM \bar{M}$ that outputs the opposite of M ?

Closure properties

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

- Well, we just proved that L and \bar{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 $TM \bar{M}$ that outputs the opposite of M ?

\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?

- Well, we just proved that L and \bar{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 $TM \bar{M}$ that outputs the opposite of M ?

\bar{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., $w \in \bar{L}$, \bar{M} outputs *ACCEPT* or loops forever

$w \in L$, i.e., $w \notin \bar{L}$, \bar{M} outputs *REJECT*

Closure properties

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

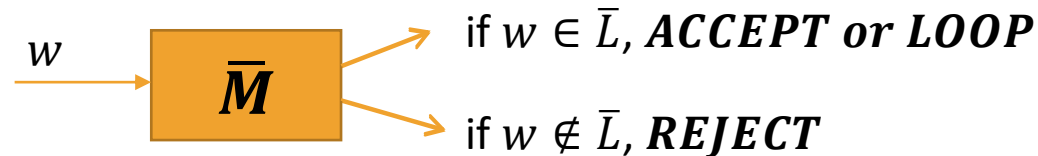
\bar{M} = On input w

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?

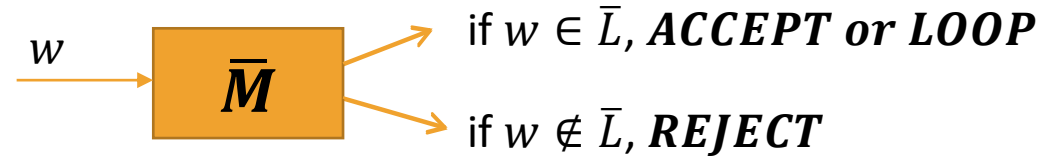
\bar{M} = On input w

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**

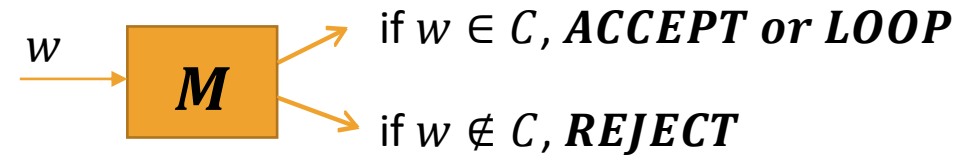
If $L \in RE$, $\bar{L} \in co-RE$ and vice versa

co-RE

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

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

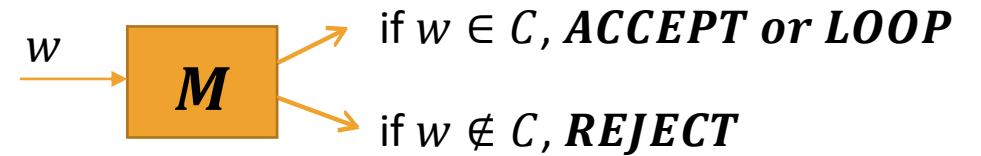
- **RE**: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- **co-RE**: May loop when $w \in L$. Halts and rejects if $w \notin L$.



$co-RE$

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

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



- **RE** : Halts and accepts if $w \in L$. May loop when $w \notin L$.
- **$co-RE$** : May loop when $w \in L$. Halts and rejects if $w \notin L$.

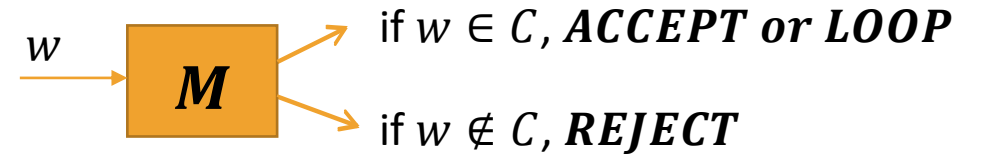
Note: Every Recursive Language R , is both in RE and $co-RE$, i.e. $R \subseteq RE \cap co-RE$

co-RE

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

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

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



- ***RE***: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- ***co-RE***: May loop when $w \in L$. Halts and rejects if $w \notin L$.

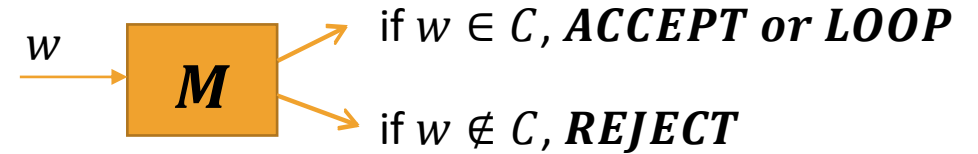
Note: Every Recursive Language R , is both in RE and $co-RE$, i.e. $R \subseteq RE \cap co-RE$

Is $R = RE \cap co-RE$?

$co-RE$

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

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



- **RE** : Halts and accepts if $w \in L$. May loop when $w \notin L$.
- **$co-RE$** : May loop when $w \in L$. Halts and rejects if $w \notin L$.

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

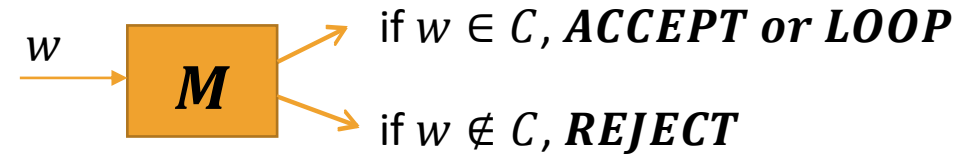
Is $R = RE \cap co-RE$?

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

$co-RE$

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

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



- **RE** : Halts and accepts if $w \in L$. May loop when $w \notin L$.
- **$co-RE$** : May loop when $w \in L$. Halts and rejects if $w \notin L$.

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

Is $R = RE \cap co-RE$?

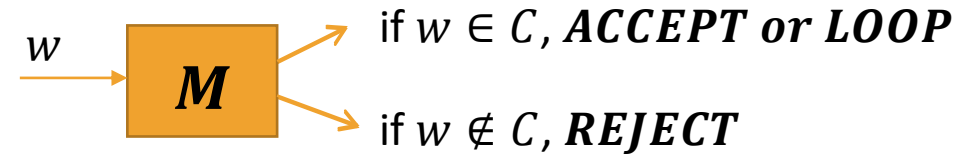
We have to prove the following: **If $L \in RE$ and $L \in co-RE$, then L is Recursive**

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

$co-RE$

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

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



- **RE** : Halts and accepts if $w \in L$. May loop when $w \notin L$.
- **$co-RE$** : May loop when $w \in L$. Halts and rejects if $w \notin L$.

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

Is $R = RE \cap co-RE$?

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

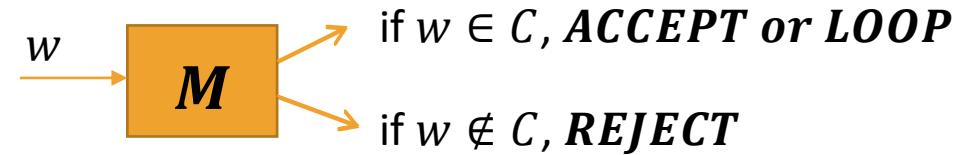
Proof: Let M be a TM such that $L(M) = L$. As $L \in co-RE$, there also exists a \bar{M} (that outputs the opposite of M) that halts and outputs reject whenever $w \notin L$. We shall construct a Total Turing Machine D by using M and \bar{M} .

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

co-RE

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

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



- ***RE***: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- ***co-RE***: May loop when $w \in L$. Halts and rejects if $w \notin L$.

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

Is $R = RE \cap co-RE$?

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

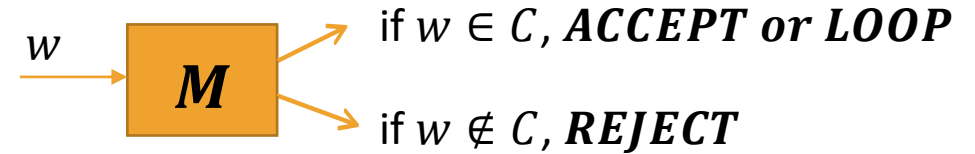
Proof: Let M be a TM such that $L(M) = L$. As $L \in co-RE$, there also exists a \bar{M} (that outputs the opposite of M) that halts and outputs reject whenever $w \notin L$. We shall construct a Total Turing Machine D by using M and \bar{M} .

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

co-RE

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

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



- **RE**: Halts and accepts if $w \in L$. May loop when $w \notin L$.
- **co-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-RE**, i.e. $R \subseteq RE \cap co-RE$

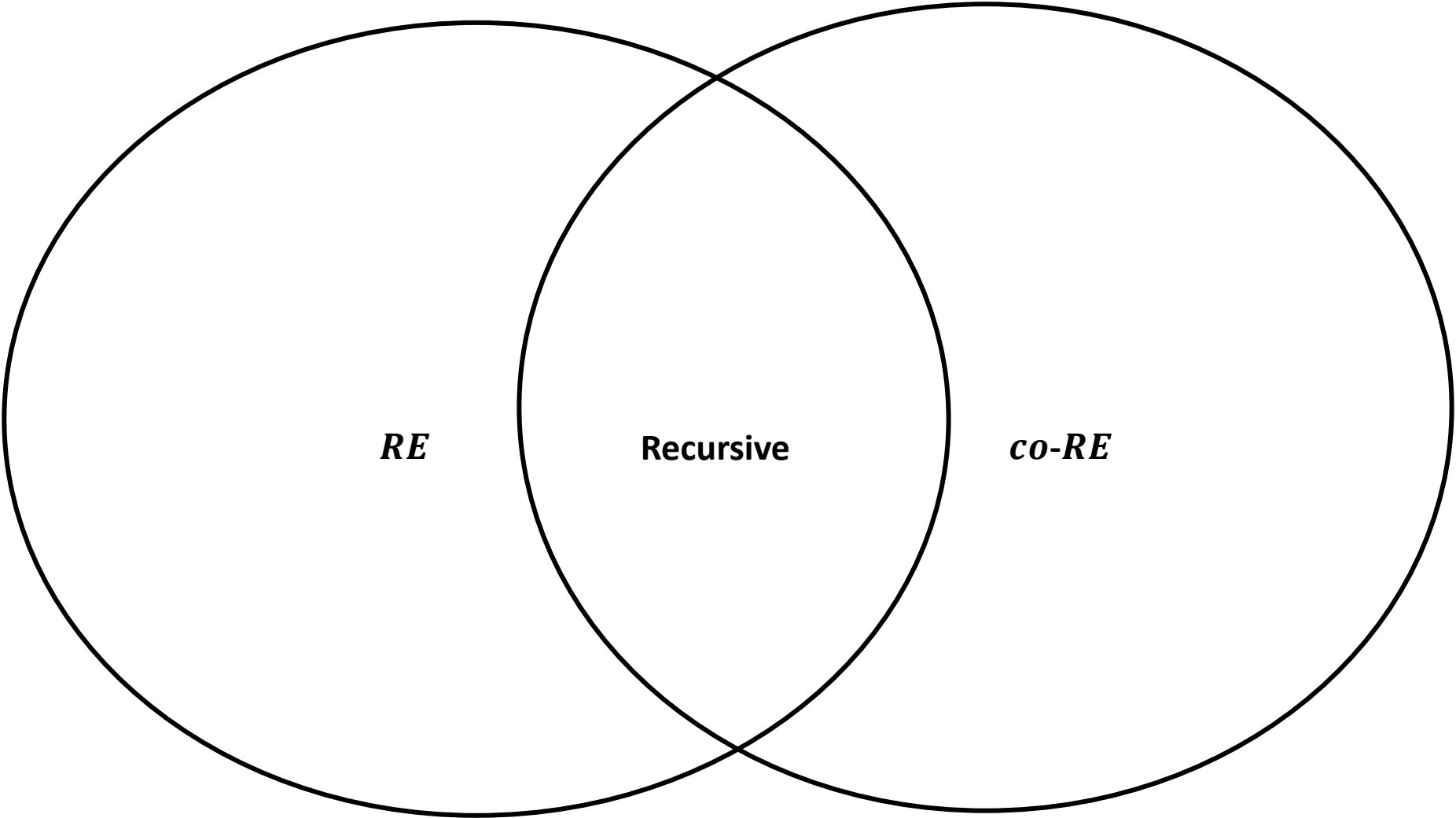
Is $R = RE \cap co-RE$?

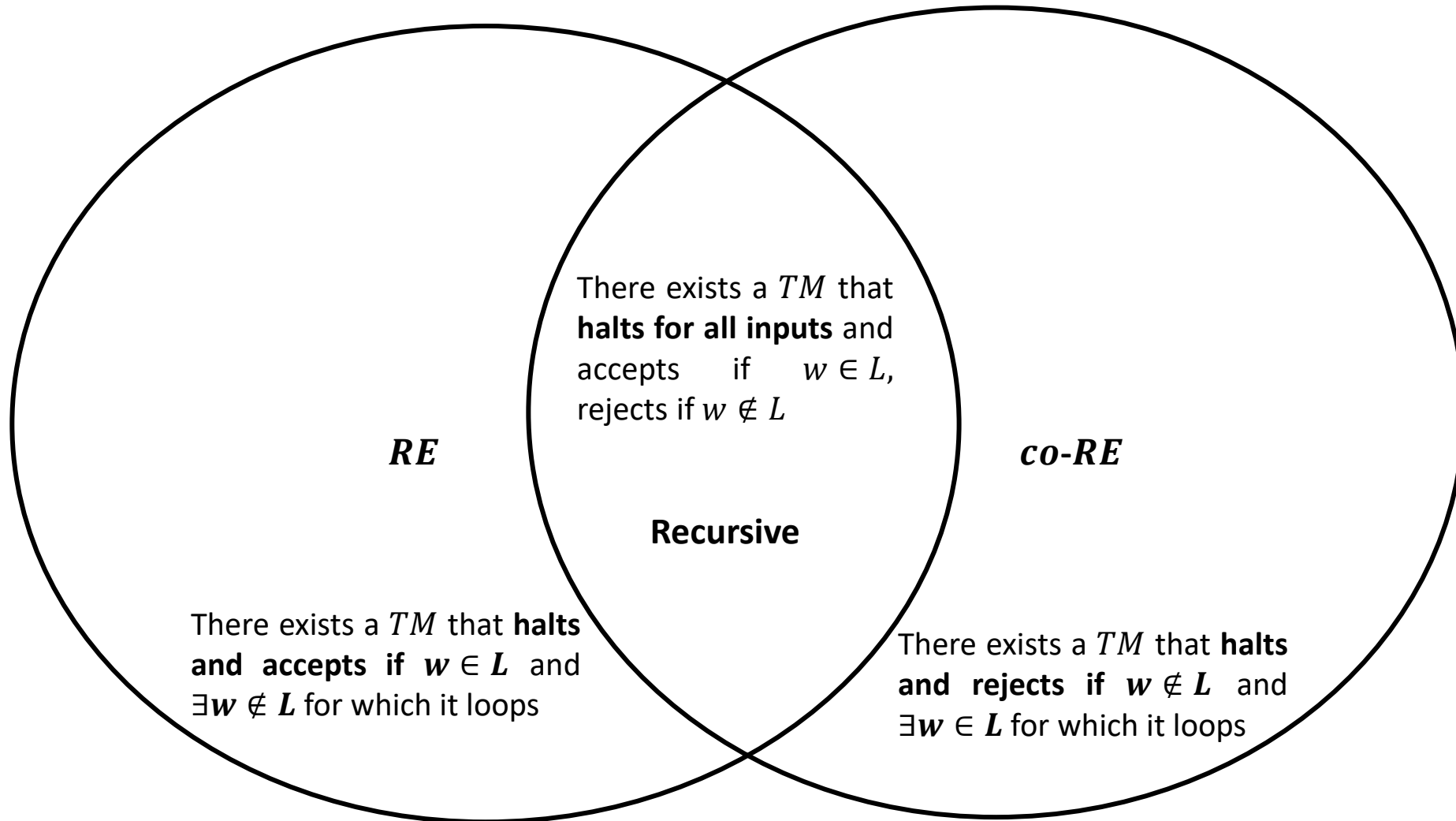
We have to prove the following: **If $L \in RE$ and $L \in co-RE$, then L is Recursive**

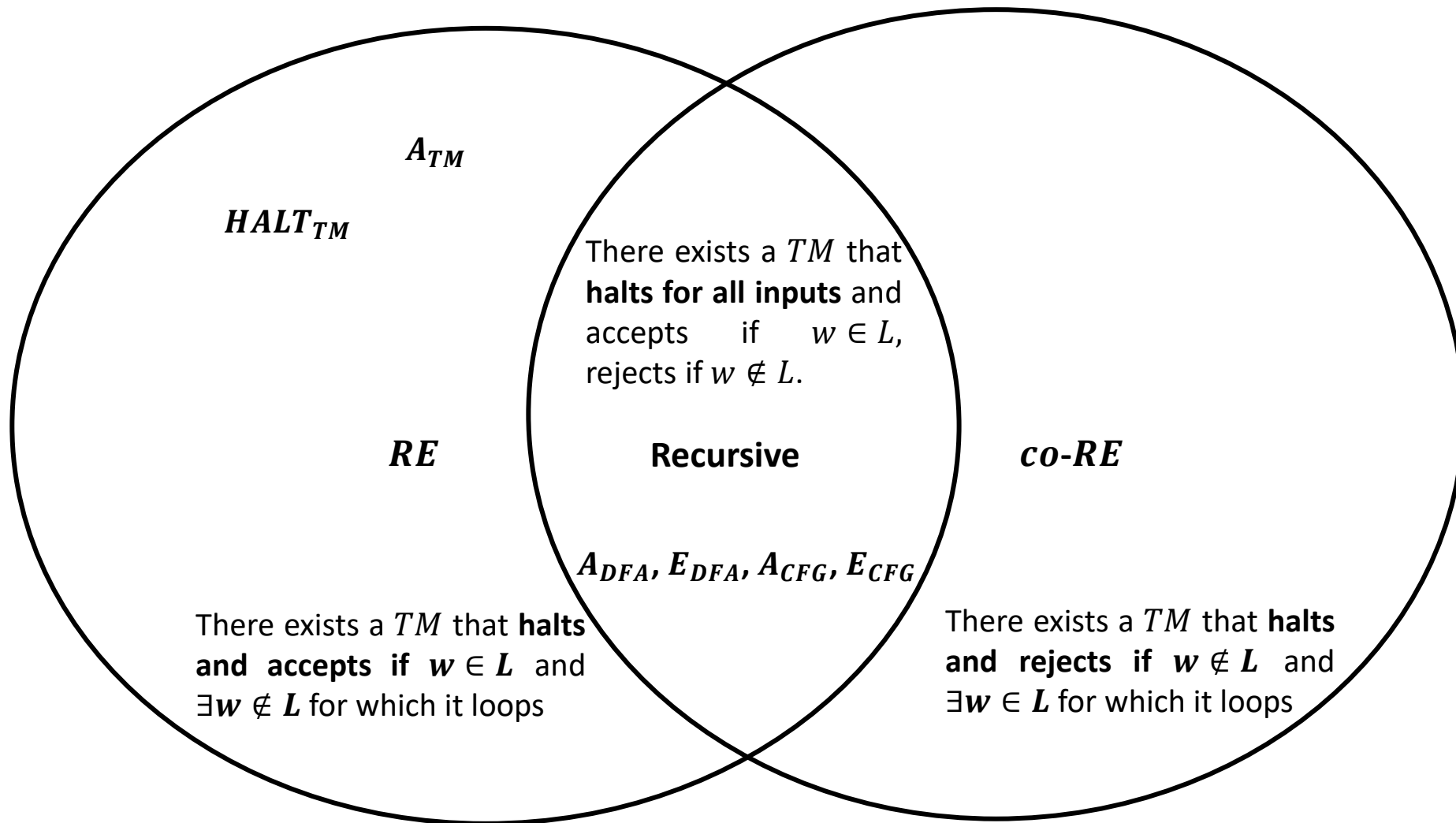
Proof: Let M be a TM such that $L(M) = L$. As $L \in co-RE$, there also exists a \bar{M} (that outputs the opposite of M) that halts and outputs reject whenever $w \notin L$. We shall construct a Total Turing Machine D by using M and \bar{M} .

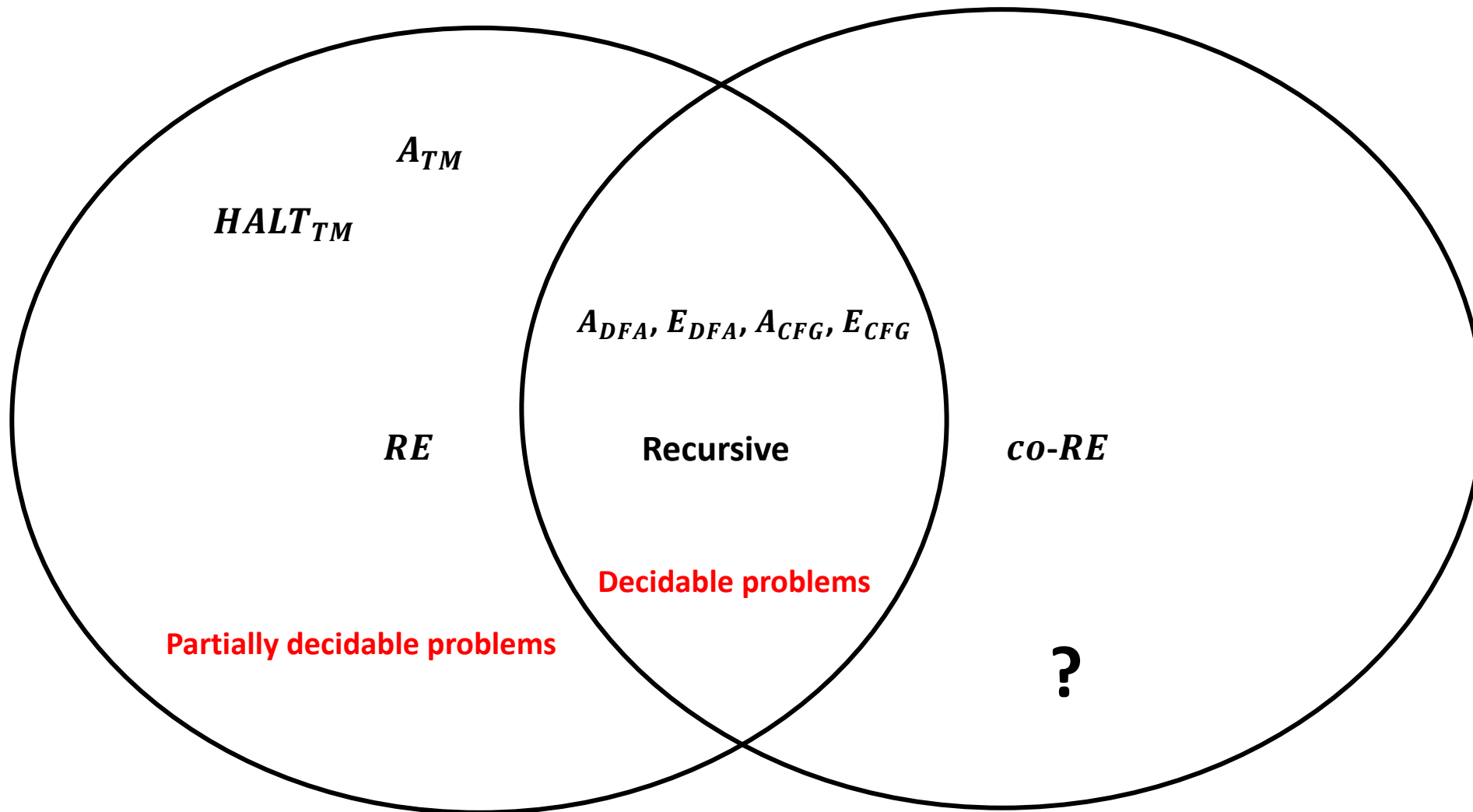
D = On input w
Run $M(w)$ and $\bar{M}(w)$ in parallel
If $M(w)$ accepts, output **ACCEPT**
If $\bar{M}(w)$ rejects, output **REJECT**

So $R = RE \cap co-RE$





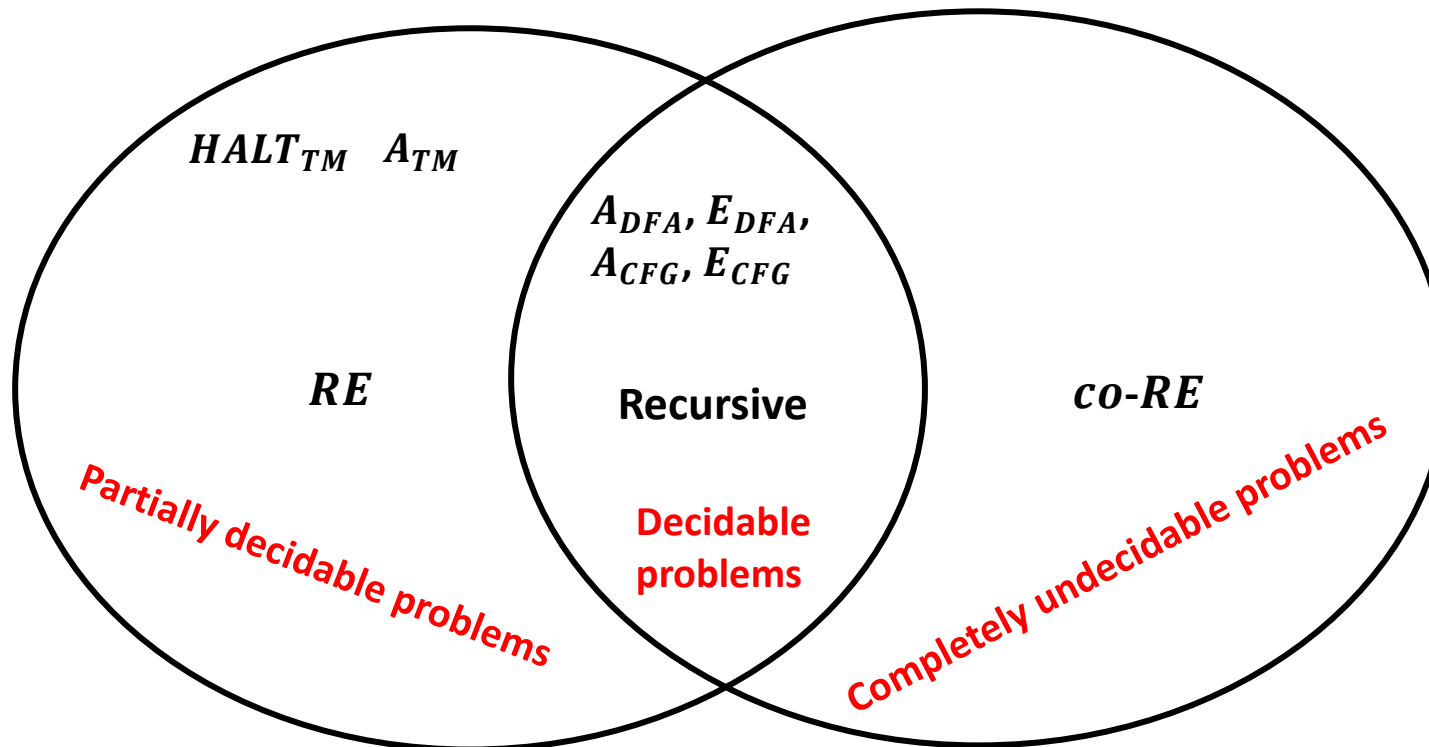




Closure properties

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.

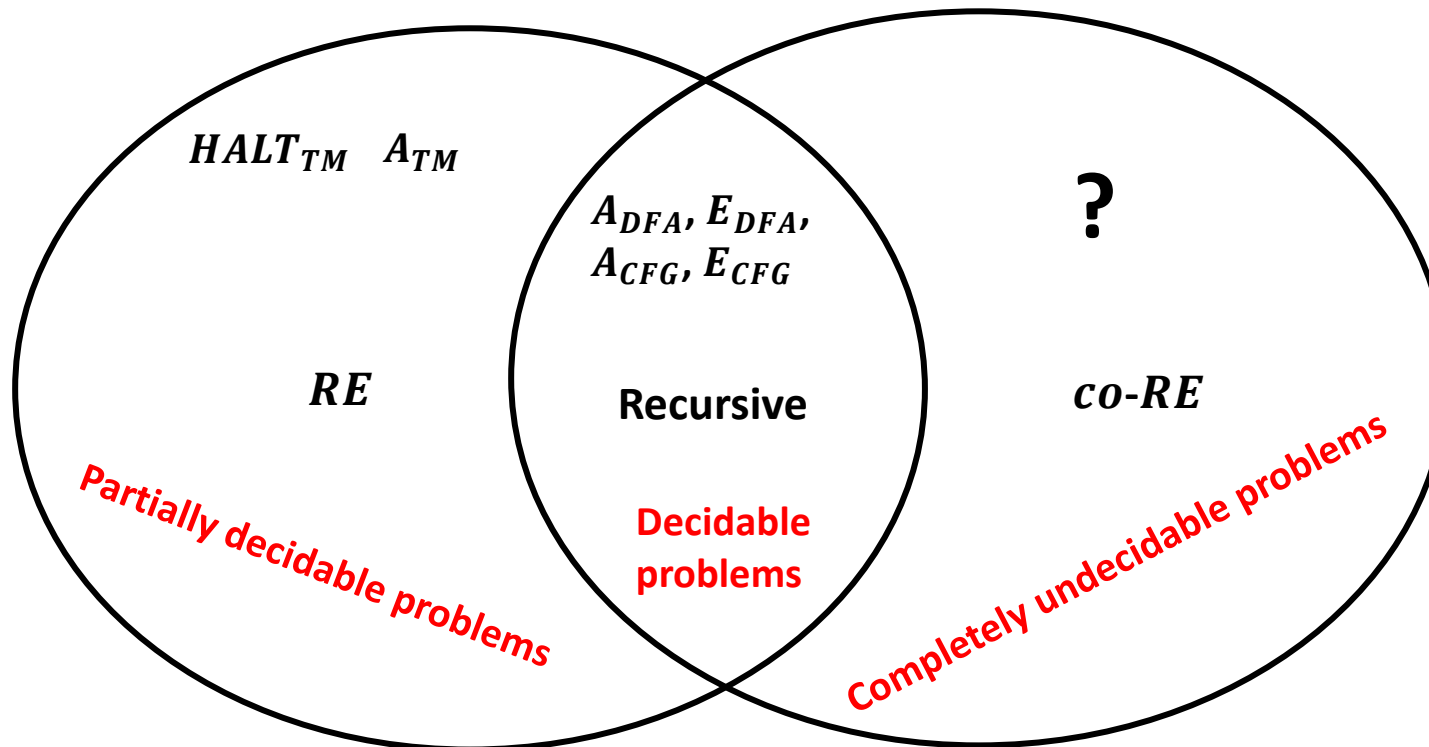
So, languages that are in *co-RE* but are not recursive are **completely undecidable**.



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-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 $\bar{L} \in co-RE$ but is not recursive. So Complement of any partially decidable language is completely undecidable

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

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

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

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

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

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

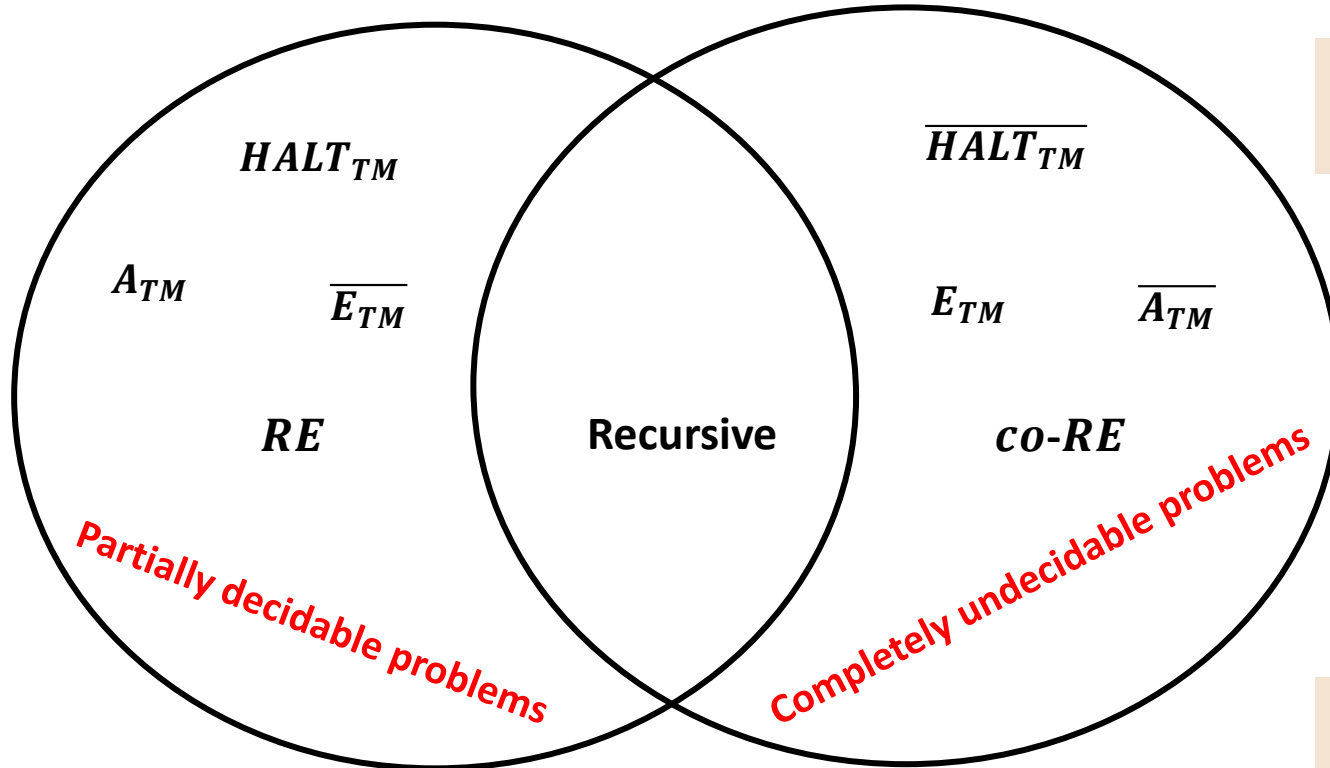
$$\overline{A_{TM}} = \{\langle M, w \rangle \mid M \text{ doesn't accept input } w\}$$

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

$$\overline{HALT_{TM}} = \{\langle M, w \rangle \mid M \text{ doesn't halt on input } w\}$$

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.



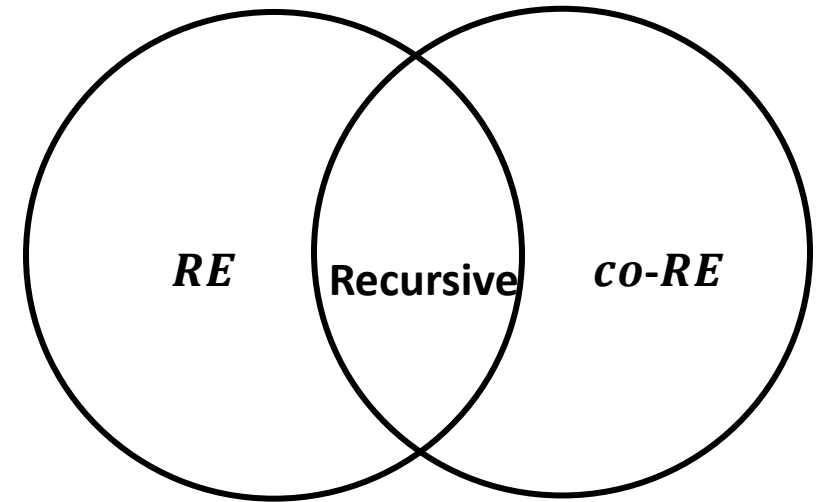
Languages that are in *co-RE* but are not recursive are **completely undecidable**.

Complement of any partially decidable language is completely undecidable.

Summing up

We have the following:

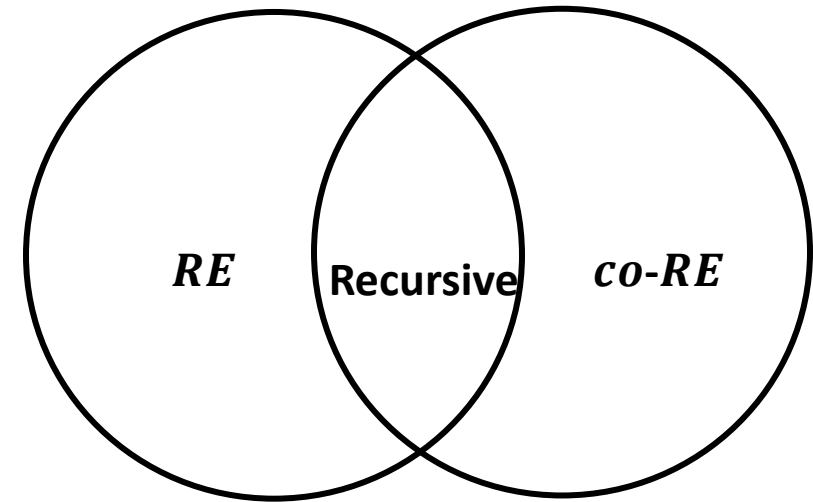
- **Recursive Languages** are closed under complement, union & intersection
- ***RE*** is closed under union & intersection but not complement



Summing up

We have the following:

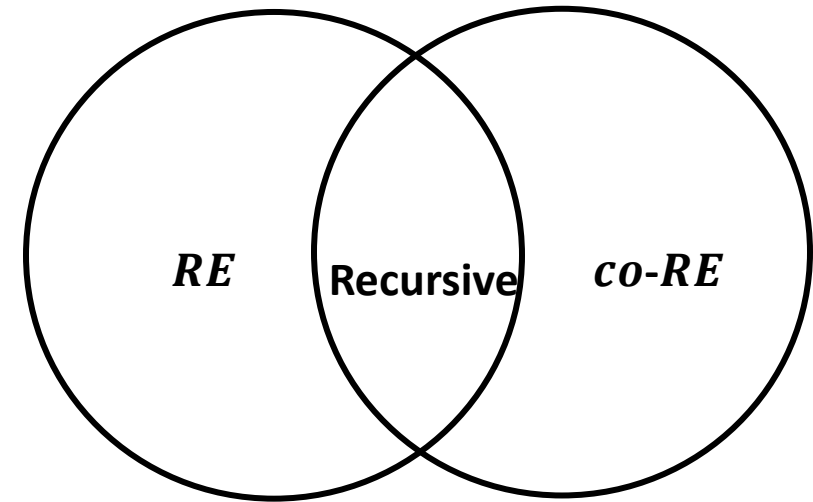
- **Recursive Languages are closed under complement, union & intersection**
- **RE is closed under union & intersection but not complement**
- **$L \in RE$ and $\bar{L} \in RE$, iff L is Recursive.**
- **If $L \in RE$ then $\bar{L} \in co-RE$.**
- **If $L \in co-RE$ then $\bar{L} \in RE$.**



Summing up

We have the following:

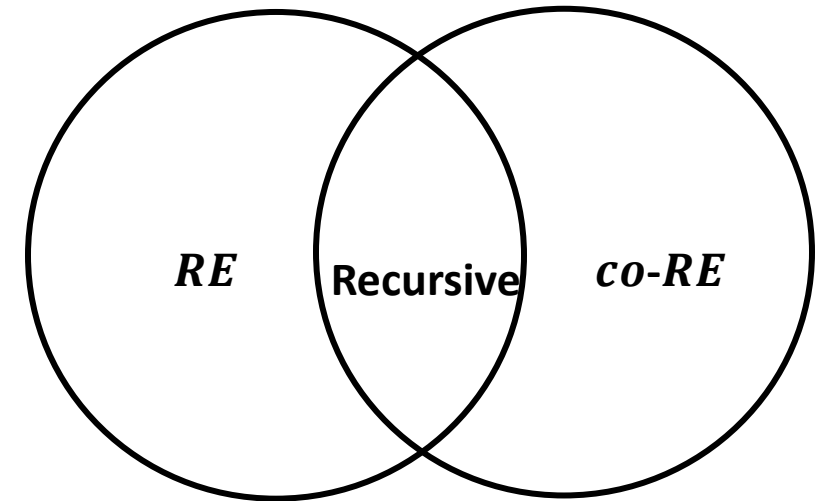
- **Recursive Languages are closed under complement, union & intersection**
- **RE is closed under union & intersection but not complement**
- **$L \in RE$ and $\bar{L} \in RE$, iff L is Recursive.**
- **If $L \in RE$ then $\bar{L} \in co-RE$.**
- **If $L \in co-RE$ then $\bar{L} \in RE$.**
- **$R =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



Summing up

We have the following:

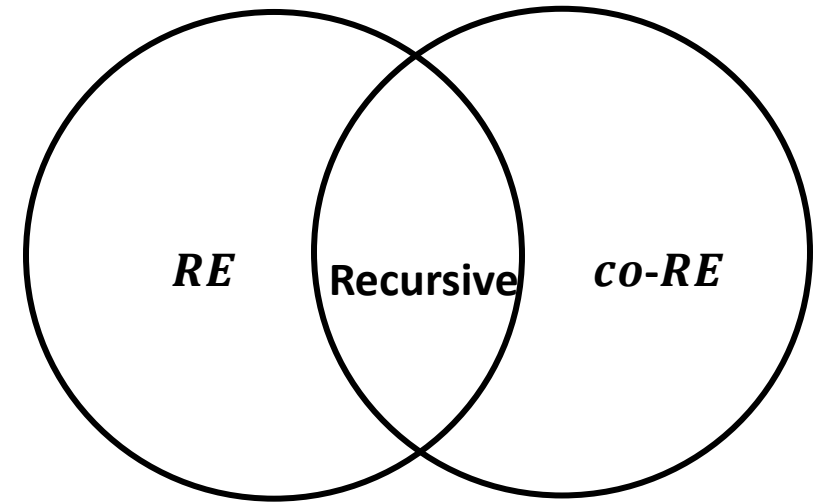
- Recursive Languages are closed under complement, union & intersection
- RE is closed under union & intersection but not complement
- $L \in RE$ and $\bar{L} \in RE$, iff L is Recursive.
- If $L \in RE$ then $\bar{L} \in co-RE$.
- If $L \in co-RE$ then $\bar{L} \in RE$.
- $R = RE \cap co-RE$
- If $L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



Note that there are languages outside of $RE \cup co-RE$.

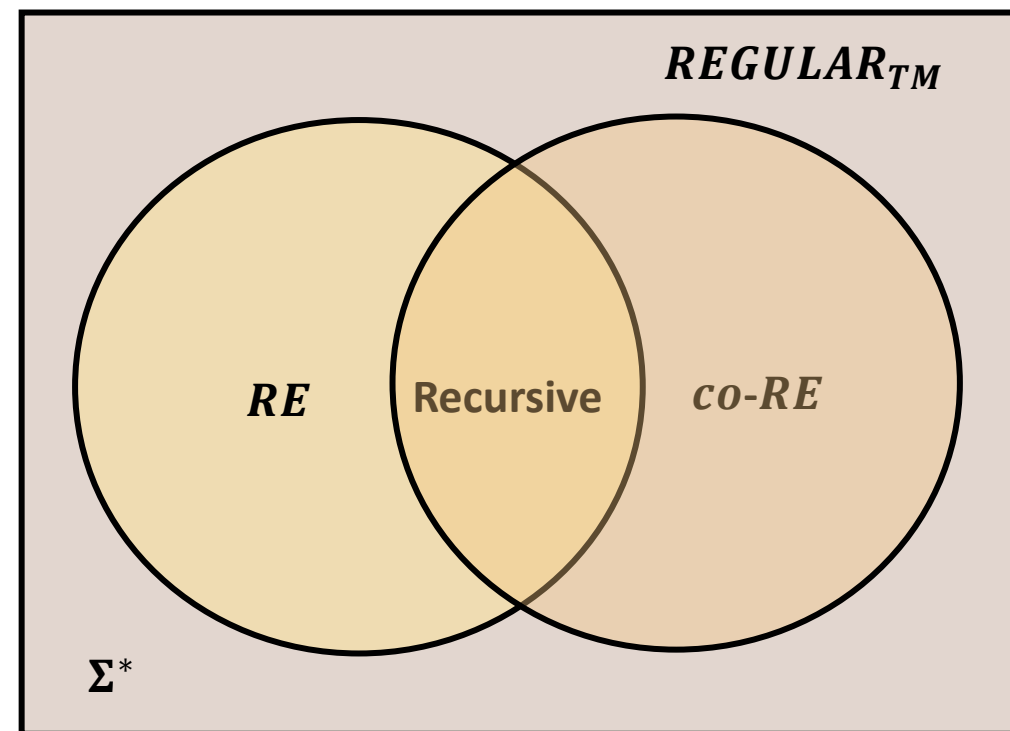
Summing up

We have the following:

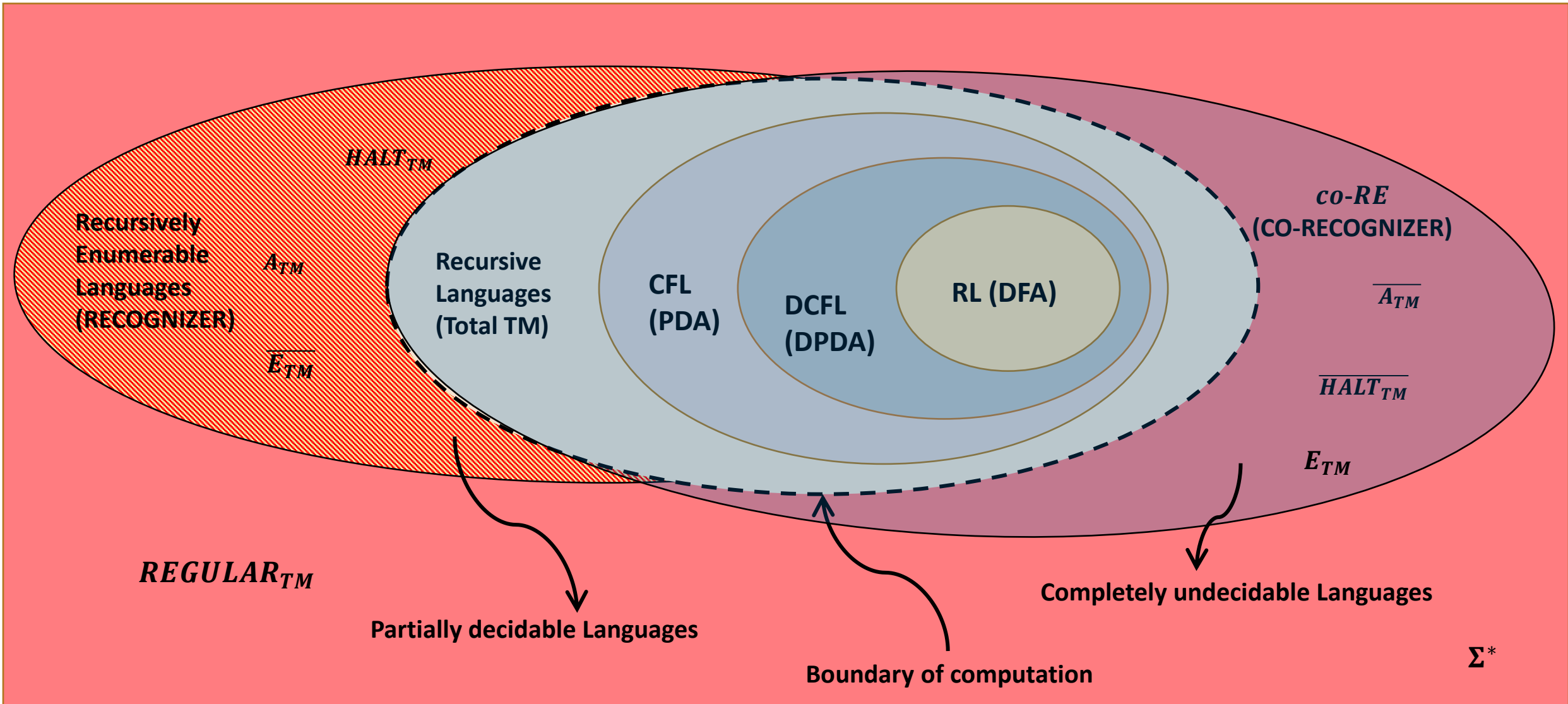
- Recursive Languages are closed under complement, union & intersection
- RE is closed under union & intersection but not complement
- $L \in RE$ and $\bar{L} \in RE$, iff L is Recursive.
- If $L \in RE$ then $\bar{L} \in co-RE$.
- If $L \in co-RE$ then $\bar{L} \in RE$.
- $R = RE \cap co-RE$
- If $L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

Note that there are languages outside of $RE \cup co-RE$.

E.g.: $REGULAR_{TM} = \{\langle M \rangle \mid L(M) \text{ is regular}\}$



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 $R, RE, co-RE$ etc
 - Mapping reduction, 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.
- Million dollar problems waiting to be solved!
- E.g.: Quantum computers model how nature computes at the fundamental level: provably faster than classical machines on several problems and most likely violates the Extended Church Turing Thesis.

Thank You!