

CS 302.1 - Automata Theory

Lecture 11

Shantanav Chakraborty

Center for Quantum Science and Technology (CQST)

Center for Security, Theory and Algorithms (CSTAR)

IIIT Hyderabad



INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY

H Y D E R A B A D

Quick Recap

The standard TM model is quite robust. It can simulate other seemingly “powerful” variants such as

- Lazy TM
- Multi-tape TM
- Two-way infinite tape TM
- Enumerator
- Non-deterministic TM

The set of problems that are decided by a standard TM = the set of problems decided by any of these variants

Turing Machines: some definitions

Total Turing Machines: A TM M is total if for all input strings $w \in \Sigma^*$, $M(w)$ accepts or rejects but never runs infinitely.

On every input, M halts

An **Algorithm** is nothing but a Total Turing Machine.

Recursive Language/Turing Decidable/Decidable: A language L is called Recursive or Turing decidable or Decidable if there exists a Total Turing Machine M for L , i.e.

$$\left. \begin{array}{l} \forall \omega \in L, M(\omega) \text{ accepts} \\ \forall \omega \notin L, M(\omega) \text{ rejects} \end{array} \right\} \text{Halts on all inputs}$$

Total TM M = On input w ,
If $M(w)$ reaches an accept state, ACCEPT
If $M(w)$ reaches a reject state, REJECT

Recursively Enumerable Language/Turing Recognizable (RE): A language L is called Recursively Enumerable (RE) or Turing Recognizable if

$$\left. \begin{array}{l} \forall \omega \in L, M(\omega) \text{ accepts} \\ \forall \omega \notin L, M(\omega) \text{ doesn't accept} \end{array} \right\} \text{(rejects or runs infinitely)}$$

Turing Machines: some definitions

Total Turing Machines: A TM M is total if for all input strings $w \in \Sigma^*$, $M(w)$ accepts or rejects but never runs infinitely.

On every input, M halts

An **Algorithm** is nothing but a Total Turing Machine.

Recursive Language/Turing Decidable/Decidable: A language L is called Recursive or Turing decidable or Decidable if there exists a Total Turing Machine M for L , i.e.

$$\left. \begin{array}{l} \forall \omega \in L, M(\omega) \text{ accepts} \\ \forall \omega \notin L, M(\omega) \text{ rejects} \end{array} \right\} \text{Halts on all inputs}$$

Recursively Enumerable Language/Turing Recognizable (RE): A language L is called Recursively Enumerable (RE) or Turing Recognizable if

$$\begin{array}{ll} \forall \omega \in L, M(\omega) \text{ accepts} & \\ \forall \omega \notin L, M(\omega) \text{ doesn't accept} & \text{(rejects or runs infinitely)} \end{array}$$

M = On input w ,
If $M(w)$ reaches an accept state, ACCEPT
If $M(w)$ reaches a reject state, REJECT
If $M(w)$ loops,

L is in RE if **L is recognized by some Turing Machine M** , i.e. $L(M) = L$. It halts for ALL the YES instances.

All Recursive Languages are Recursively Enumerable but not vice versa.

Turing Machines: some definitions

Total Turing Machines: A TM M is total if for all input strings $w \in \Sigma^*$, $M(w)$ accepts or rejects but never runs infinitely.

On every input, M halts

An **Algorithm** is nothing but a Total Turing Machine.

Recursive Language/Turing Decidable/Decidable: A language L is called Recursive or Turing decidable or Decidable if there exists a Total Turing Machine M for L , i.e.

$$\left. \begin{array}{l} \forall \omega \in L, M(\omega) \text{ accepts} \\ \forall \omega \notin L, M(\omega) \text{ rejects} \end{array} \right\} \text{Halts on all inputs}$$

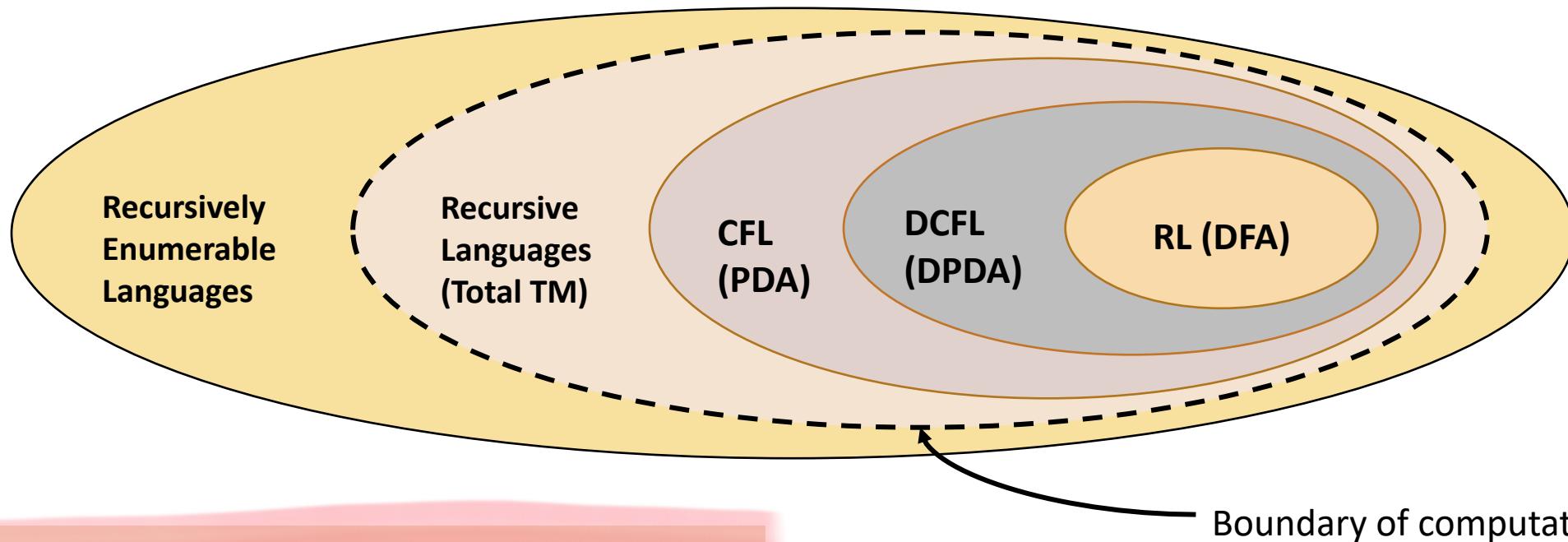
Recursively Enumerable Language/Turing Recognizable: A language L is called Recursively Enumerable (RE) or Turing Recognizable if

$$\begin{array}{ll} \forall \omega \in L, M(\omega) \text{ accepts} & \\ \forall \omega \notin L, M(\omega) \text{ doesn't accept} & \text{(rejects or runs infinitely)} \end{array}$$

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

$$\begin{array}{ll} \forall \omega \in L, M(\omega) \text{ doesn't reject} & \text{(accepts or loops)} \\ \forall \omega \notin L, M(\omega) \text{ rejects} & \end{array}$$

Hierarchy of Languages



Any problem that is not Recursive (not decidable) is called Undecidable. There exists some input w for which the Turing Machine loops forever and hence, cannot decide whether or not w belongs to the Language.

We cannot write Algorithms to decide the membership of undecidable problems

There are problems in RE which are not Recursive. For such problems there exists some $\omega \notin L$, the TM never halts but rather loops forever. So, such problems are undecidable.

However, they can recognize any $\omega \in L$, so these undecidable problems are also called partially decidable.

Turing Machines: some definitions

Undecidable language: A language L is undecidable if it is not decidable/recursiv. Any TM for L will loop infinitely for some input ω . You cannot write an algorithm to decide the membership of L .

Undecidable languages can be of two kinds:

- **Partially decidable Language:** A language L is partially decidable if L is Recursively Enumerable as well as Undecidable (not recursive) (TM accepts all the YES instances and loops infinitely for at least one NO instance), i.e.

$\forall \omega \in L, M(\omega)$ accepts

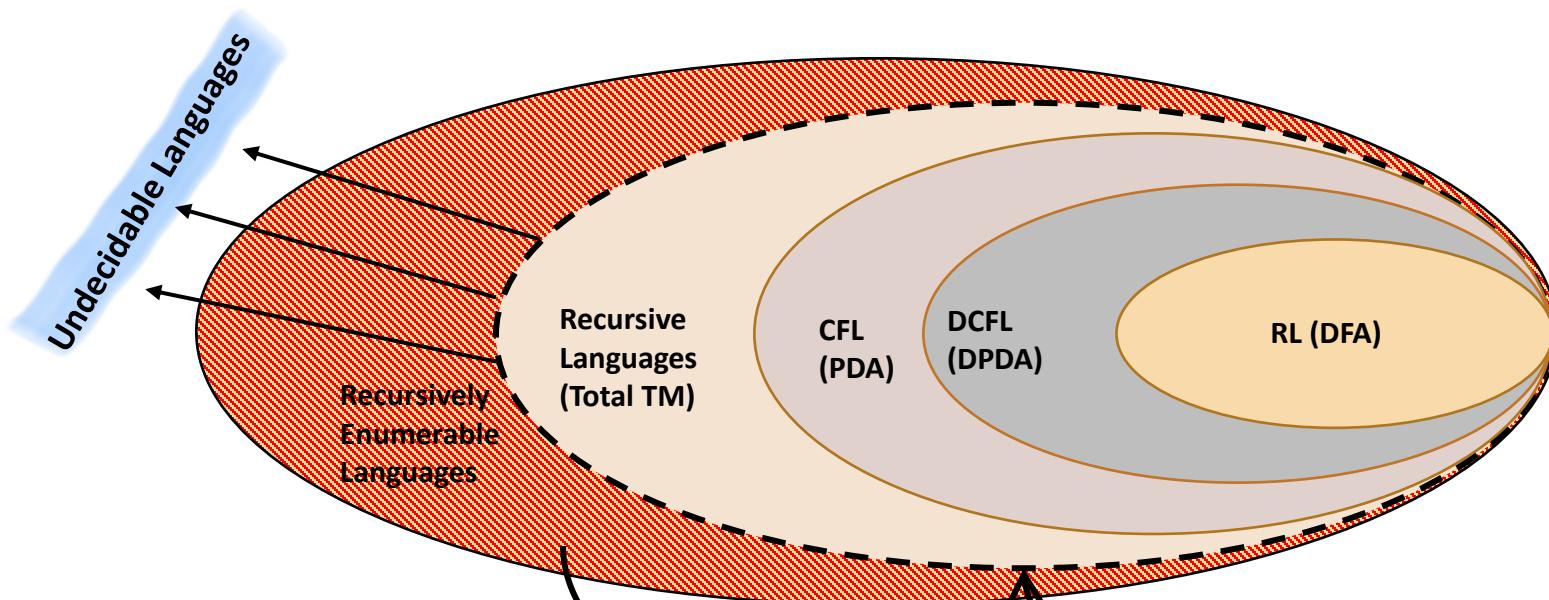
$\forall \omega \notin L, M(\omega)$ doesn't accept but \exists at least one instance where the program will loop forever.

- **Completely undecidable language:** A language L is completely undecidable if L is undecidable but not partially decidable (TM loops infinitely for at least one YES instance), i.e.

$\forall \omega \in L, M(\omega)$ doesn't accept and \exists at least one instance where the program will loop forever

$\forall \omega \notin L, M(\omega)$ rejects/loops forever

Hierarchy of Languages



Partially decidable
Languages

Boundary of computation

Any Language here is computable/decidable (you can write algorithms)

$(RL \equiv DFA) \subseteq (DCFL \equiv DPDA) \subseteq (CFL \equiv PDA) \subseteq (Recursive\ Lang \equiv Total\ TM)$

$\subseteq RE$

Undecidable Languages – Problems
here are NOT computable

Boundary of computation

Encoding

The input to a TM are often strings/sequences of strings.

$M(w_1, w_2) =$ If w_1 is a substring of w_2 , ACCEPT
Otherwise, REJECT.

Consider this example:

$M(\langle M_1, w \rangle) =$ Run M_1 on input w .
If $M_1(w)$ accepts, ACCEPT
If $M_1(w)$ rejects, REJECT

Not just numbers, seemingly complicated objects such as a **graph**, a **DFA**, a **CFG** and even a **Turing Machine** itself can be encoded as a string – and hence can be an input to a TM.

- $\langle M_1 \rangle$ is the encoding of TM M_1 as a string.
- M simulates the run of M_1 on input w .
- Observe that M can accept a description of itself as input.

- Encoding objects such as TMs as strings will help define a Universal Turing Machine U_{TM} which is a DTM that accepts as input the encoding of a DTM M and an input string w , and simulates $M(w)$.
- To prove that problems related to regular languages, CFLs are decidable/undecidable, we need to provide encodings DFAs/CFGs as inputs to a TM.
- How can we encode objects as strings? We will show a simple encoding of a DTM into a binary string.

Encoding a Turing Machine

- We will provide a simple mapping from a DTM to a **binary string**.
- Of course, this is not the only encoding.
- You can come up with your own encoding.

Encoding a Turing Machine

Recall that a DTM M is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.

- Let $Q = \{q_0, \dots, q_{m-1}\}$, $\Sigma = \{0, 1, \dots, k-1\}$, $\Gamma = \{0, 1, \dots, n-1\}$. As $\Sigma \subseteq \Gamma$, $k < n$ and without loss of generality B corresponds to the last symbol $n-1$ in Γ .
- Any state $q_i \in Q$ can be encoded as a binary string, where

$$\langle q_0 \rangle = 0, \langle q_1 \rangle = 1, \langle q_2 \rangle = 10, \dots$$

- Any symbol in Γ (or Σ) can be encoded as

$$\langle 0 \rangle = 0, \langle 1 \rangle = 1, \langle 2 \rangle = 10, \dots$$

- The directions $\langle L \rangle = 0$ and $\langle R \rangle = 1$. So the transition function $\delta(q_i, a) = (q_j, b, L/R)$ is just the sequence

$$\langle \langle q_i \rangle, \langle a \rangle, \langle q_j \rangle, \langle b \rangle, \langle L/R \rangle \rangle$$

- All such transitions are listed in lexicographic order into

$$\langle \delta \rangle = \langle \langle \delta_0 \rangle, \langle \delta_1 \rangle, \langle \delta_2 \rangle, \dots \rangle$$

Encoding a Turing Machine

Recall that a DTM M is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.

Let $Q = \{q_0, \dots, q_{m-1}\}$, $\Sigma = \{0, 1, \dots, k-1\}$, $\Gamma = \{0, 1, \dots, n-1\}$.

Following these encodings, we can simply encode the DTM M as

$$\langle M \rangle = (\langle m \rangle, \langle k \rangle, \langle n \rangle, \langle \delta \rangle, \langle q_{accept} \rangle, \langle q_{reject} \rangle)$$

We are almost there but not quite. We have to find a way to combine this tuple of binary strings into one bigger binary string. Note that $\langle \delta \rangle$ itself is a tuple of binary strings.

We can combine multiple sequences of binary strings into one as follows. Consider the sequence

$$\langle \langle a_1 \rangle, \langle a_2 \rangle, \dots, \langle a_n \rangle \rangle = \langle \langle a_1 \rangle \# \langle a_2 \rangle \# \dots \# \langle a_n \rangle \rangle,$$

where a_i are binary strings of finite length.

We claim that using the following map suffices

$$0 \mapsto 00$$

$$1 \mapsto 01$$

$$\# \mapsto 1$$

Encoding a Turing Machine

$$\langle M \rangle = (\langle m \rangle, \langle k \rangle, \langle n \rangle, \langle \delta \rangle, 0/1, \langle q_{accept} \rangle, \langle q_{reject} \rangle)$$

We can combine multiple sequences of binary strings into one as follows. Consider the sequence

$$\langle \langle a_1 \rangle, \langle a_2 \rangle, \dots, \langle a_n \rangle \rangle = \langle \langle a_1 \rangle \# \langle a_2 \rangle \# \dots \# \langle a_n \rangle \rangle,$$

where a_i are binary strings of finite length.

We claim that using the following map suffices

$$\begin{aligned} 0 &\mapsto \textcolor{green}{00} \\ 1 &\mapsto \textcolor{red}{01} \\ \# &\mapsto \textcolor{blue}{1} \end{aligned}$$

Why does this work?

- For a 0 in an odd position, the symbol immediately following it corresponds to the symbol that was encoded
- We can identify the delimiter as the 1 that appears in an odd position.

Encoding a Turing Machine

$$\langle M \rangle = (\langle m \rangle, \langle k \rangle, \langle n \rangle, \langle \delta \rangle, 0/1, \langle q_{accept} \rangle, \langle q_{reject} \rangle)$$

We can combine multiple sequences of binary strings into one as follows. Consider the sequence

$$\langle \langle a_1 \rangle, \langle a_2 \rangle, \dots, \langle a_n \rangle \rangle = \langle \langle a_1 \rangle \# \langle a_2 \rangle \# \dots \# \langle a_n \rangle \rangle,$$

where a_i are binary strings of finite length.

We claim that using the following map suffices

$$\begin{aligned} 0 &\mapsto \textcolor{green}{00} \\ 1 &\mapsto \textcolor{red}{01} \\ \# &\mapsto \textcolor{blue}{1} \end{aligned}$$

E.g. Let $a_1 = 1101$ and $a_2 = 010$. Then $\langle 1101, 010 \rangle \mapsto \textcolor{red}{0101} \textcolor{green}{0001} \textcolor{red}{1} \textcolor{green}{0001} \textcolor{red}{00}$

To recover a_1 and a_2 from the encoding:

- For any 0 in odd positions, the symbol that follow in the even positions, belong to a_1 .
- If a 1 is obtained in an odd position, it corresponds to the delimiter/partition $\Rightarrow a_1$ has been recovered, now a_2 will be obtained similarly.
- This can be generalized to multiple tuples of binary strings which is what we need to encode M .

Encoding a Turing Machine

So for any DTM M , we obtain an encoding

$$\langle M \rangle = (\langle m \rangle, \langle k \rangle, \langle n \rangle, \langle \delta \rangle, 0/1, \langle q_{accept} \rangle, \langle q_{reject} \rangle)$$

such that $\langle M \rangle \in \{0,1\}^*$.

Every DTM corresponds to a binary string but the reverse is not necessarily true. Some binary strings are not valid descriptions of DTMs.

Can we make this a bijection?

- Lexicographically generate finite-length binary strings.
- For any length k , there are 2^k binary strings of length k
- So any TM that can be described by a k -length binary string will be within this finite set.
- Some of these will not correspond to a valid DTM. Ignore them.
- Relabel the first binary string that corresponds to a valid TM as 0.
- Relabel the second binary string that corresponds to a valid TM as 1.

0
1
01
...
00000011
...
0001100001

Say this is the first binary string that corresponds to a valid TM. Relabel this as 0

If this is the next binary string, corresponding to a valid TM, relabel this as 1 and so on.

Now we have a one-one mapping (bijective relationship) between the set of finite-length binary strings and DTMs.

Universal Turing Machines

Now that we have shown how to encode objects including Turing Machines as binary strings, we can now define **Universal Turing Machines** – or Turing Machines that simulate other Turing Machines.

Universal Turing Machine: A Universal Turing Machine, denoted as U_{TM} accepts as input (i) the encoding of a Turing Machine M , (ii) an input string w and **simulates M running on w** , i.e.

$$U_{TM}(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects} \\ \text{LOOPS INFINITELY, if } M(w) \text{ loops infinitely} \end{cases}$$

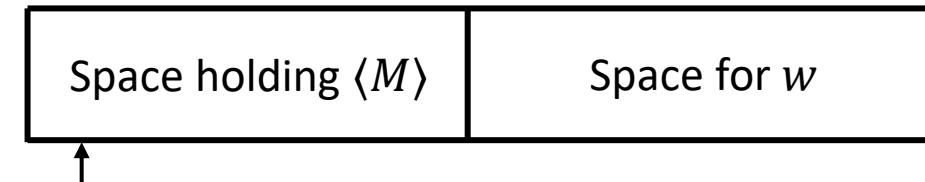
By the **Church-Turing thesis**, a U_{TM} can perform any computation on any feasible computational device.

So, in principle using U_{TM} , Turing Machines can answer questions about Turing Machines!

So for any DTM M , we obtain an encoding

$$\langle M \rangle = (\langle m \rangle, \langle k \rangle, \langle n \rangle, \langle \delta \rangle, 0/1, \langle q_{accept} \rangle, \langle q_{reject} \rangle)$$

such that $\langle M \rangle \in \{0,1\}^*$.



U_{TM} checks

- the space for w to determine the symbol currently being read
- And the space containing $\langle M \rangle$ for determining the transition function to be implemented

Some Decidable Languages

Much like Turing Machines, DFAs, NFAs, CFGs can also be encoded as binary strings. In fact, a bijection can be established between binary strings and these objects.

This is useful as it helps answer the decidability of Languages related to them.

Examples: The following languages are decidable

- $A_{DFA} = \{\langle DFA, w \rangle | w \in L(DFA)\}$

So, for any DTM M , we obtain an encoding

$$\langle M \rangle = (\langle m \rangle, \langle k \rangle, \langle n \rangle, \langle \delta \rangle, 0/1, \langle q_{accept} \rangle, \langle q_{reject} \rangle)$$

such that $\langle M \rangle \in \{0,1\}^*$.

M = On input $\langle DFA, w \rangle$:

- Simulate the run of $\langle DFA \rangle$ on w .
- If w is accepted, output *ACCEPT*
- If w is rejected, output *REJECT*

Some Decidable Languages

Much like Turing Machines, DFAs, NFAs, CFGs can also be encoded as binary strings. In fact, a bijection can be established between binary strings and these objects.

This is useful as it helps answer the decidability of Languages related to them.

Examples: The following languages are decidable

- $A_{DFA} = \{\langle DFA, w \rangle | w \in L(DFA)\}$
- $E_{DFA} = \{\langle DFA \rangle | L(DFA) = \Phi\}$

So, for any DTM M , we obtain an encoding

$$\langle M \rangle = (\langle m \rangle, \langle k \rangle, \langle n \rangle, \langle \delta \rangle, 0/1, \langle q_{accept} \rangle, \langle q_{reject} \rangle)$$

such that $\langle M \rangle \in \{0,1\}^*$.

M = On input $\langle DFA \rangle$:

- Mark the start state of $\langle DFA \rangle$
- Repeat until no new states are marked
 - Mark any state that has an incoming transition from a marked state
- If the final state is unmarked, *ACCEPT*, else *REJECT*

Some Decidable Languages

Much like Turing Machines, DFAs, NFAs, CFGs can also be encoded as binary strings. In fact, a bijection can be established between binary strings and these objects.

This is useful as it helps answer the decidability of Languages related to them.

So, for any DTM M , we obtain an encoding

$$\langle M \rangle = (\langle m \rangle, \langle k \rangle, \langle n \rangle, \langle \delta \rangle, 0/1, \langle q_{accept} \rangle, \langle q_{reject} \rangle)$$

such that $\langle M \rangle \in \{0,1\}^*$.

Examples: The following languages are decidable

- $A_{DFA} = \{\langle DFA, w \rangle | w \in L(DFA)\}$
- $E_{DFA} = \{\langle DFA \rangle | L(DFA) = \Phi\}$
- $A_{CFG} = \{\langle CFG, w \rangle | w \in L(CFG)\}$

$M =$ On input $\langle CFG, w \rangle$:

- Convert $\langle CFG \rangle$ into CNF
- List all derivations of $2|w| - 1$ steps
- If any of these derivations yield w , ACCEPT, else REJECT

Or, run the CYK algorithm

Some Decidable Languages

Much like Turing Machines, DFAs, NFAs, CFGs can also be encoded as binary strings. In fact, a bijection can be established between binary strings and these objects.

This is useful as it helps answer the decidability of Languages related to them.

So, for any DTM M , we obtain an encoding

$$\langle M \rangle = (\langle m \rangle, \langle k \rangle, \langle n \rangle, \langle \delta \rangle, 0/1, \langle q_{accept} \rangle, \langle q_{reject} \rangle)$$

such that $\langle M \rangle \in \{0,1\}^*$.

Examples: The following languages are decidable

- $A_{DFA} = \{\langle DFA, w \rangle | w \in L(DFA)\}$
- $E_{DFA} = \{\langle DFA \rangle | L(DFA) = \Phi\}$
- $A_{CFG} = \{\langle CFG, w \rangle | w \in L(CFG)\}$
- $E_{CFG} = \{\langle CFG \rangle | L(CFG) = \Phi\}$

Idea similar to DFAs: Check if the Start Variable leads to any terminal

Some Decidable Languages

Much like Turing Machines, DFAs, NFAs, CFGs can also be encoded as binary strings. In fact, a bijection can be established between binary strings and these objects.

This is useful as it helps answer the decidability of Languages related to them.

Examples: The following languages are decidable

- $A_{DFA} = \{\langle DFA, w \rangle | w \in L(DFA)\}$
- $E_{DFA} = \{\langle DFA \rangle | L(DFA) = \Phi\}$
- $A_{CFG} = \{\langle CFG, w \rangle | w \in L(CFG)\}$
- $E_{CFG} = \{\langle CFG \rangle | L(CFG) = \Phi\}$

So, for any DTM M , we obtain an encoding

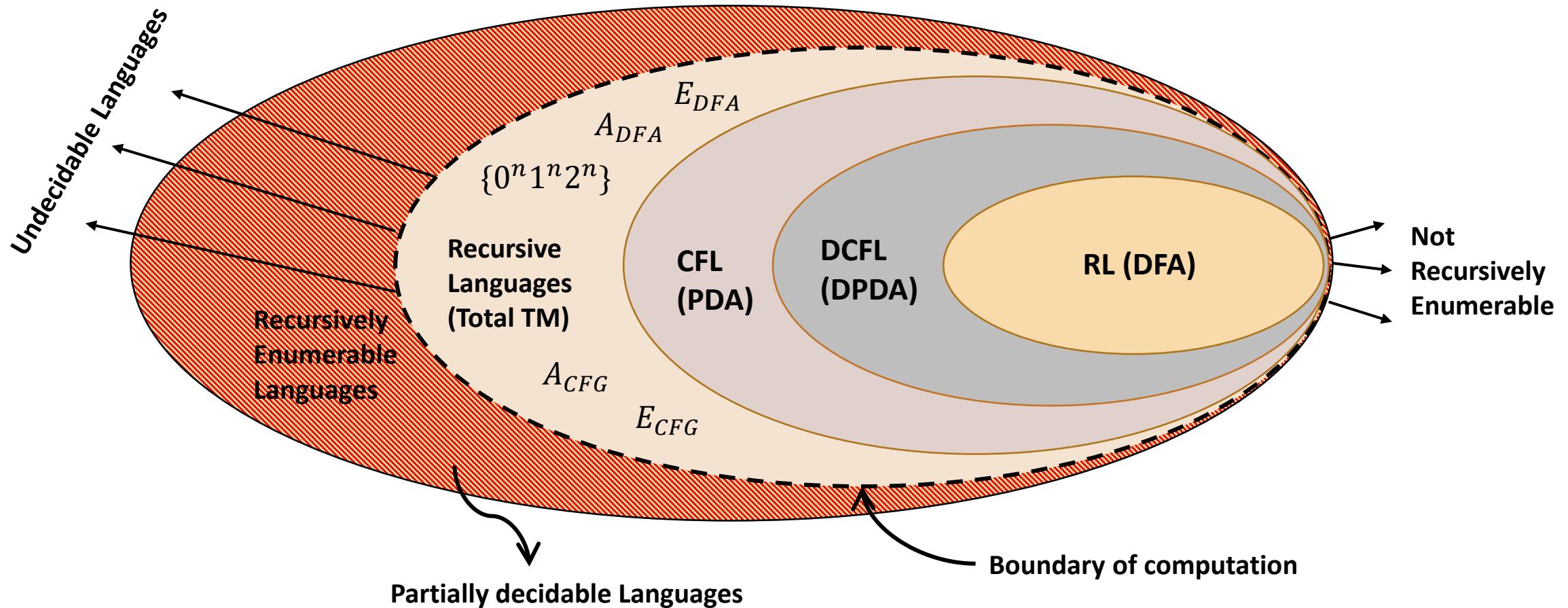
$$\langle M \rangle = (\langle m \rangle, \langle k \rangle, \langle n \rangle, \langle \delta \rangle, 0/1, \langle q_{accept} \rangle, \langle q_{reject} \rangle)$$

such that $\langle M \rangle \in \{0,1\}^*$.

M = On input $\langle CFG \rangle$:

- Mark all terminal symbols
- Repeat until no new variables are marked
 - Mark any V , s.t. $V \rightarrow X_1X_2 \cdots X_l$.
- If S is unmarked, ACCEPT. Else REJECT

Some Decidable Languages



- $A_{DFA} = \{\langle DFA \rangle, w | w \in L(DFA)\}$
- $E_{DFA} = \{\langle DFA \rangle | L(DFA) = \Phi\}$
- $A_{CFG} = \{\langle CFG, w \rangle | w \in L(CFG)\}$
- $E_{CFG} = \{\langle CFG \rangle | L(CFG) = \Phi\}$

An undecidable problem

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

A_{TM} : Does there exist a Total Turing Machine A that accepts as input a Turing Machine M and an input string w and outputs ACCEPT, if $M(w)$ accepts w and REJECT, if $M(w)$ does not accept w (rejects or loops forever)?

$$A(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects or loops infinitely} \end{cases}$$

Every (finite length) binary string is a TM and vice versa. So, the **input may have two copies of the same string (say w)**:

- The first copy corresponds to the encoding of some TM M_w .
- The second copy is the input string $w = \langle M_w \rangle$.

$$A(w, w) = A(M_w, \langle M_w \rangle)$$

In this case, A simulates the run of TM M_w on the input string w , which is the binary encoding of M_w itself

An undecidable problem

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

A_{TM} : Does there exist a Total Turing Machine A that accepts as input a Turing Machine M and an input string w and outputs ACCEPT, if $M(w)$ accepts w and REJECT, if $M(w)$ does not accept w (rejects or loops forever)?

$$A(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects or loops infinitely} \end{cases}$$

There can be inputs such as $A(\langle w, w \rangle)$

Let $w = \langle M_w \rangle$

$$A(\langle w, w \rangle) = \begin{cases} \text{ACCEPTS, if } M_w(\langle M_w \rangle) \text{ accepts} \\ \text{REJECTS, if } M_w(\langle M_w \rangle) \text{ rejects or loops infinitely} \end{cases}$$

An undecidable problem

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

A_{TM} : Does there exist a Total Turing Machine A that accepts as input a Turing Machine M and an input string w and outputs ACCEPT, if $M(w)$ accepts w and REJECT, if $M(w)$ does not accept w (rejects or loops forever)?

$$A(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects or loops infinitely} \end{cases}$$

- We will show that if such a Total TM A exists, we run into the following contradiction

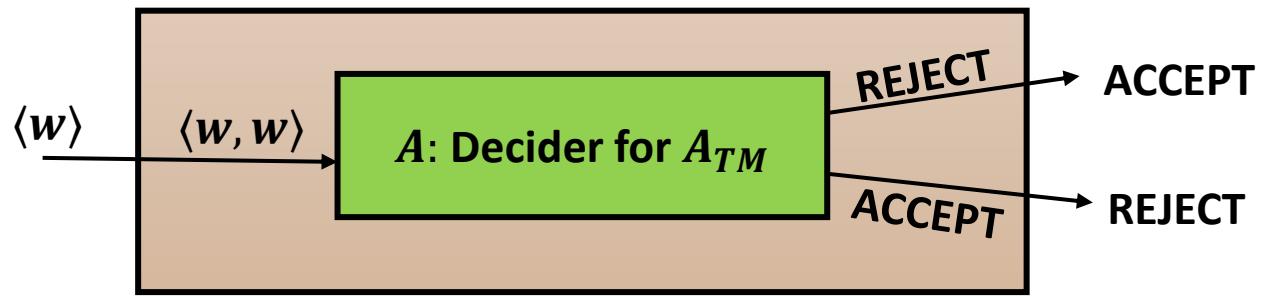
Using A , we can build a new Total TM for which there exists an instance for which the machine **both accepts and rejects!**

An undecidable problem

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

$$A(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects or loops infinitely} \end{cases}$$

Proof: Let us assume that a Total Turing machine A exists. Then we can construct a special Total Turing Machine D that accepts an input w and uses A as a subroutine to simulate $A(\langle w, w \rangle)$ in the following way:



$$D(w) = \{ \text{ Run } A(\langle w, w \rangle)$$

If $A(\langle w, w \rangle)$ accepts, D outputs REJECT
If $A(\langle w, w \rangle)$ rejects, D outputs ACCEPT

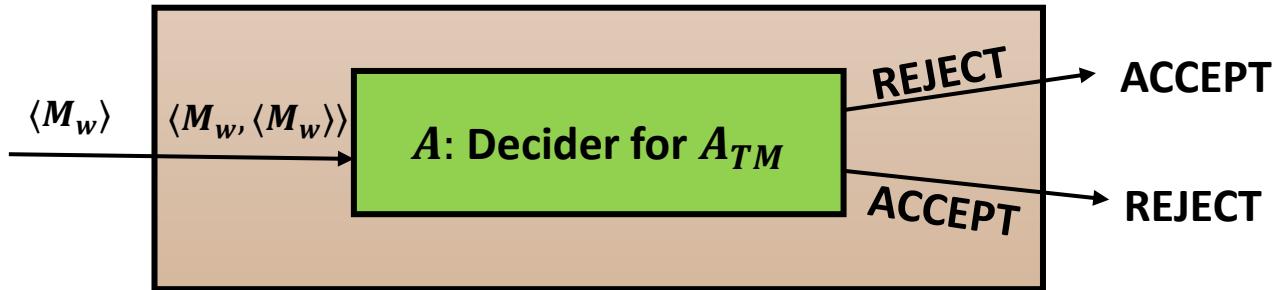
}

An undecidable problem

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

$$A(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects or loops infinitely} \end{cases}$$

Proof: Let us assume that a Total Turing machine A exists. Then we can construct a special Total Turing Machine D that accepts an input w and uses A as a subroutine to simulate $A(\langle w, w \rangle)$ in the following way:



Let $w = \langle M_w \rangle$, then

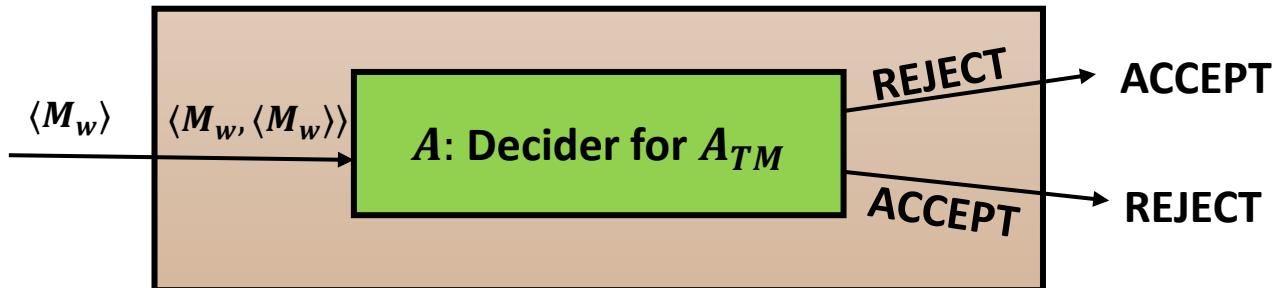
$$D(\langle M_w \rangle) = \begin{cases} \text{ACCEPTS, if } M_w(\langle M_w \rangle) \text{ doesn't accept} \\ \text{REJECTS, if } M_w(\langle M_w \rangle) \text{ accepts} \end{cases}$$

An undecidable problem

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

$$A(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects or loops infinitely} \end{cases}$$

Proof: Let us assume that a Total Turing machine A exists. Then we can construct a special Total Turing Machine D that accepts an input w and uses A as a subroutine to simulate $A(\langle w, w \rangle)$ in the following way:



Let $w = \langle M_w \rangle$, then

$$D(\langle M_w \rangle) = \begin{cases} \text{ACCEPTS, if } M_w(\langle M_w \rangle) \text{ doesn't accept} \\ \text{REJECTS, if } M_w(\langle M_w \rangle) \text{ accepts} \end{cases}$$

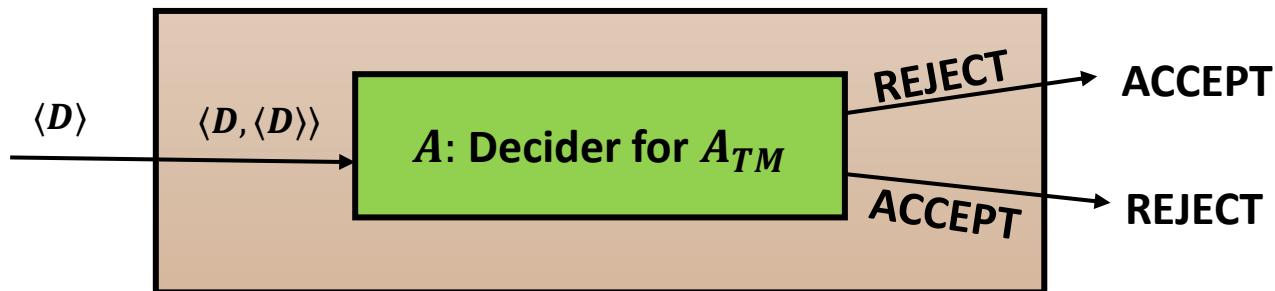
What happens when $w = \langle D \rangle$ i.e., $M_w = D$?

An undecidable problem

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

$$A(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects or loops infinitely} \end{cases}$$

Proof: Let us assume that a Total Turing machine A exists. Then we can construct a special Total Turing Machine D that accepts an input w and uses A as a subroutine to simulate $A(\langle w, w \rangle)$ in the following way:



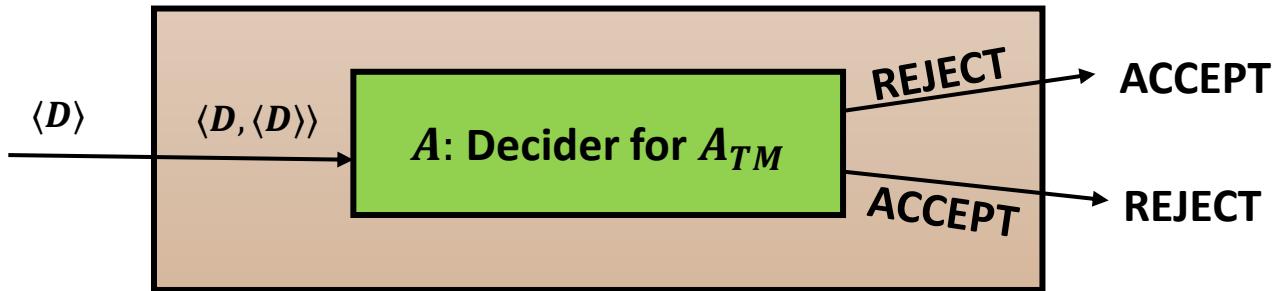
$$D(\langle M_w \rangle) = \begin{cases} \text{ACCEPTS, if } M_w(\langle M_w \rangle) \text{ doesn't accept} \\ \text{REJECTS, if } M_w(\langle M_w \rangle) \text{ accepts} \end{cases}$$

An undecidable problem

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

$$A(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects or loops infinitely} \end{cases}$$

Proof: Let us assume that a Total Turing machine A exists. Then we can construct a special Total Turing Machine D that accepts an input w and uses A as a subroutine to simulate $A(\langle w, w \rangle)$ in the following way:



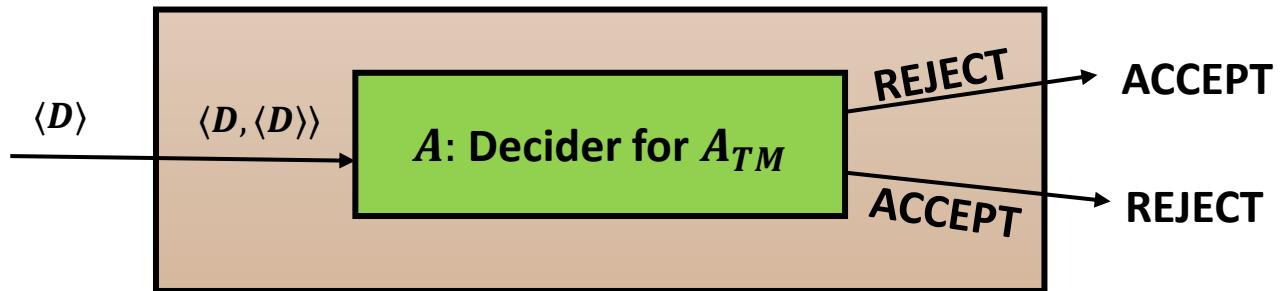
$$D(\langle D \rangle) = \begin{cases} \text{ACCEPTS, if } D(\langle D \rangle) \text{ doesn't accept} \\ \text{REJECTS, if } D(\langle D \rangle) \text{ accepts} \end{cases}$$

An undecidable problem

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

$$A(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects or loops infinitely} \end{cases}$$

Proof: Let us assume that a Total Turing machine A exists. Then we can construct a special Total Turing Machine D that accepts an input w and uses A as a subroutine to simulate $A(\langle w, w \rangle)$ in the following way:



$$D(\langle D \rangle) = \begin{cases} \text{ACCEPTS, if } D(\langle D \rangle) \text{ doesn't accept} \\ \text{REJECTS, if } D(\langle D \rangle) \text{ accepts} \end{cases}$$

CONTRADICTION!

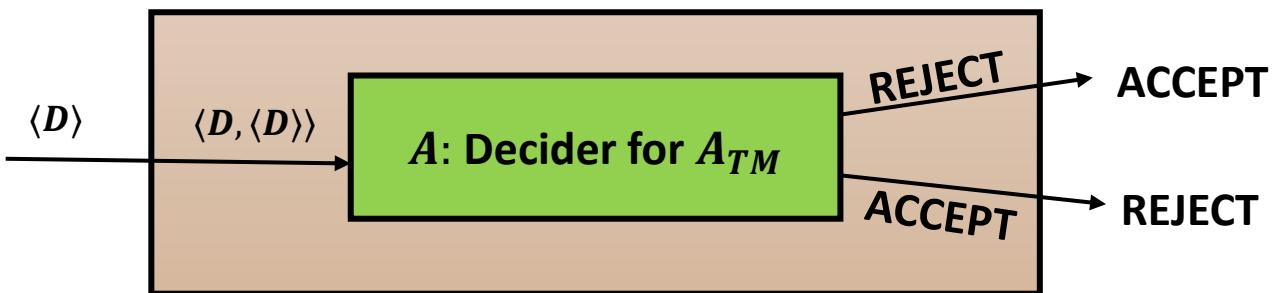
An undecidable problem

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

$$A(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects or loops infinitely} \end{cases}$$

Proof: Let us assume that a Total Turing machine A exists. Then we can construct a special Total Turing Machine D that accepts an input w and uses A as a subroutine to simulate $A(\langle w, w \rangle)$ in the following way:

$$D(\langle D \rangle) = \begin{cases} \text{ACCEPTS, if } D(\langle D \rangle) \text{ doesn't accept} \\ \text{REJECTS, if } D(\langle D \rangle) \text{ accepts} \end{cases}$$



CONTRADICTION!

- D cannot be a Total TM as it cannot decide input $\langle D \rangle$.
- If a total TM A existed, we could have constructed a total TM D .
- So, a total TM A cannot exist and hence A_{TM} is not decidable.

An undecidable problem

$A_{TM} = \{\langle M, w \rangle \mid M \text{ accepts input } w\}$. Is A_{TM} decidable? NO!

$$A(\langle M, w \rangle) = \begin{cases} \text{ACCEPTS, if } M(w) \text{ accepts} \\ \text{REJECTS, if } M(w) \text{ rejects or loops infinitely} \end{cases}$$

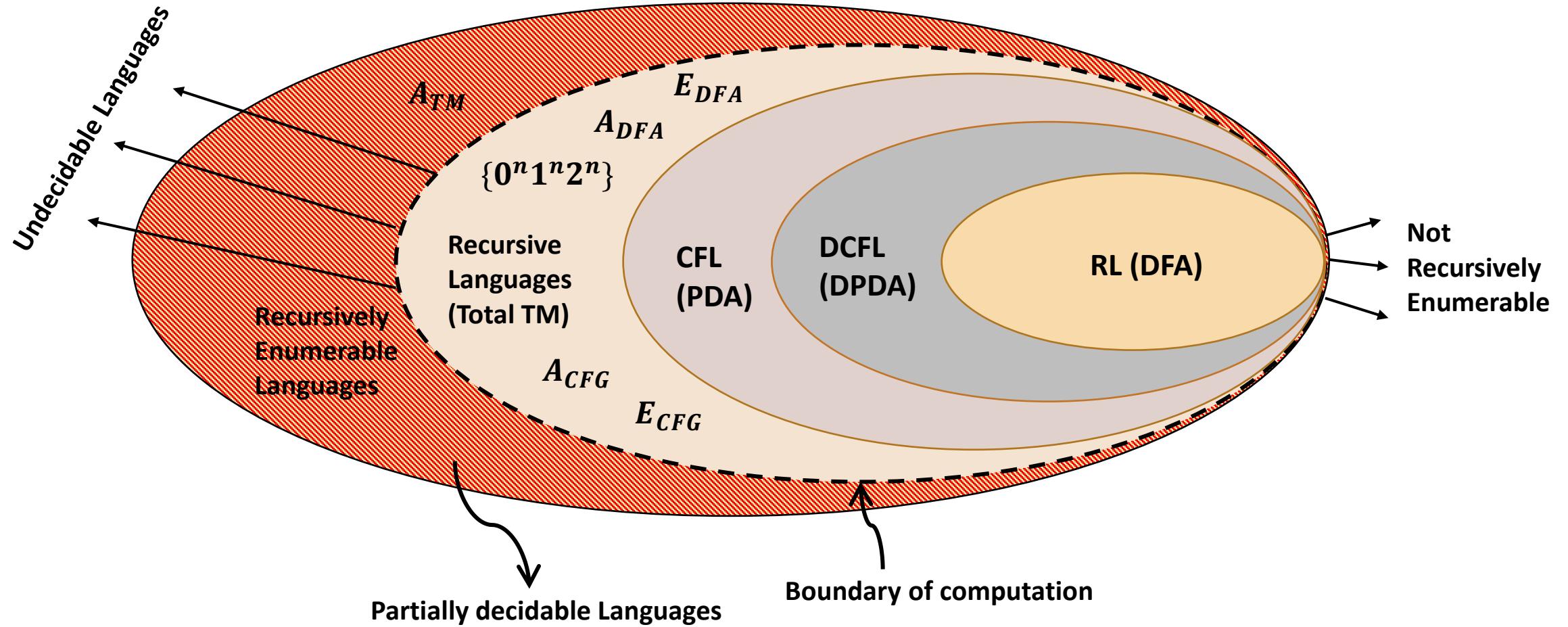
Of course, $A_{TM} \in RE$ as A halts whenever M accepts w and so

U = On input $\langle M, w \rangle$:

- Simulate M on input w
- If M accepts w , ACCEPT; if M rejects w , REJECT

U recognizes A_{TM}

- A_{TM} is undecidable
- $A_{TM} \in RE$ but not recursive
- A_{TM} is partially decidable



Next Lecture:

- Halting Problem
- More on Recursive & RE languages
- Co-RE language

Thank You!