Undecidability

Dr. Mousumi Dutt

Decidable Problems

A problem is decidable if we can construct a Turing machine which will halt in finite amount of time for every input and give answer as ‘yes’ or ‘no’. A decidable problem has an algorithm to determine the answer for a given input.

Examples

- **Equivalence of two regular languages:** Given two regular languages, there is an algorithm and Turing machine to decide whether two regular languages are equal or not.
- **Finiteness of regular language:** Given a regular language, there is an algorithm and Turing machine to decide whether regular language is finite or not.
- **Emptiness of context free language:** Given a context free language, there is an algorithm whether CFL is empty or not.

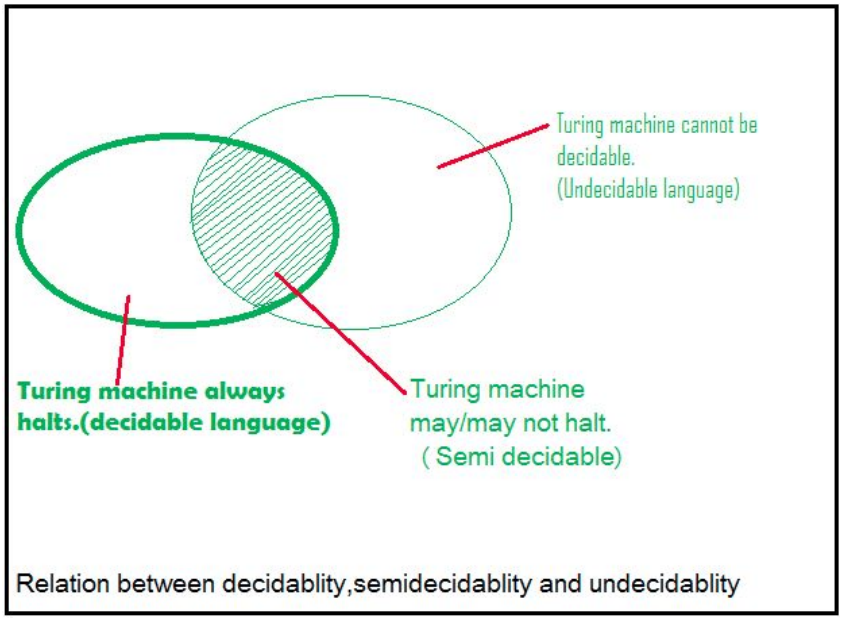
Undecidable Problems

A problem is undecidable if there is no Turing machine which will always halt in finite amount of time to give answer as ‘yes’ or ‘no’. An undecidable problem has no algorithm to determine the answer for a given input.

- **Ambiguity of context-free languages:** Given a context-free language, there is no Turing machine which will always halt in finite amount of time and give answer whether language is ambiguous or not.
- **Equivalence of two context-free languages:** Given two context-free languages, there is no Turing machine which will always halt in finite amount of time and give answer whether two context free languages are equal or not.
- **Everything or completeness of CFG:** Given a CFG and input alphabet, whether CFG will generate all possible strings of input alphabet ($\sum^*$)is undecidable.
- **Regularity of CFL, CSL, REC and REC:** Given a CFL, CSL, REC or REC, determining whether this language is regular is undecidable.

Undecidable Problems

Two popular undecidable problems are halting problem of TM and PCP (Post Correspondence Problem). Semi-decidable Problems. A semi-decidable problem is subset of undecidable problems for which Turing machine will always halt in finite amount of time for answer as ‘yes’ and may or may not halt for answer as ‘no’.



Relation between decidablity,semidecidablity and undecidablity

Finding Unsolvable Problems

We can use the fact that $L_D \notin \mathbf{RE}$ to show that other languages are also not **RE**.

General proof approach: to show that some language L is not **RE**, we will do the following:

- Assume for the sake of contradiction that $L \in \mathbf{RE}$, meaning that there is some TM M for it.
- Show that we can build a TM that uses M as a subroutine in order to recognize L_D.
- Reach a contradiction, since no TM recognizes L_D.
- Conclude, therefore, that $L \notin \mathbf{RE}$.

The Complement of A_{TM}

Recall: the language A_{TM} is the language of the universal Turing machine U_{TM}:

$$\mathbf{A_{TM} = \mathscr{L}(U_{TM}) = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}}$$

The complement of A_{TM} (denoted $\overline{A}_{TM}$) is the language of all strings not contained in A_{TM}.

Questions:

- What language is this?
- Is this language **RE**?

The Complement of A_{TM}

The language A_{TM} is defined as

$$\{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$$

Equivalently:

$$\{x \mid x = \langle M, w\rangle \text{ for some TM } M \text{ and string } w, \text{ and } M \text{ accepts } w\}$$

Thus $\overline{A}_{TM}$ is

$$\{x \mid x \neq \langle M, w\rangle \text{ for any TM } M \text{ and string } w, \text{ or } M \text{ is a TM that does not accept } w\}$$

The Complement of A_{TM}

Although the language $A_{TM} \in \mathbf{RE}$ (since it's the language of U_{TM}), its complement $\overline{A}_{TM} \notin \mathbf{RE}$.

We will prove this as follows:

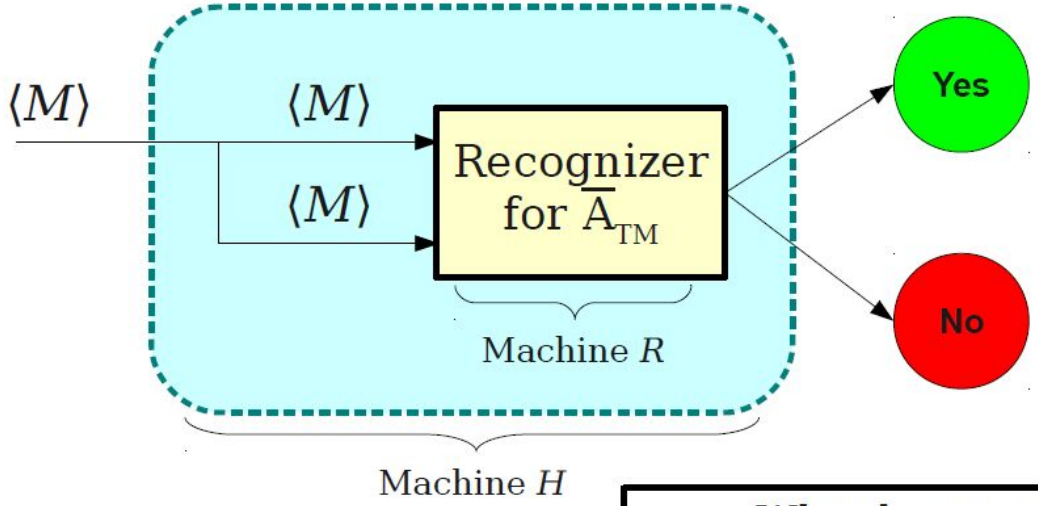
- Assume, for contradiction, that $\overline{A}_{TM} \in \mathbf{RE}$.
- This means there is a TM R for $\overline{A}_{TM}$.
- Using R as a subroutine, we will build a TM H that will recognize L_D.
- This is impossible, since $L_D \notin \mathbf{RE}$.
- Conclude, therefore, that $\overline{A}_{TM} \notin \mathbf{RE}$.

The Complement of A_{TM}

The languages L_D and $\overline{A}_{TM}$ are closely related:

- L_D: Does M not accept $\langle M \rangle$?
- $\overline{A}_{TM}$: Does M not accept string w?

Given this connection, we will show how to turn a hypothetical recognizer for $\overline{A}_{TM}$ into a hypothetical recognizer for L_D.



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

- Construct the string $\langle M, \langle M \rangle \rangle$.
- Run R on $\langle M, \langle M \rangle \rangle$.
- If R accepts $\langle M, \langle M \rangle \rangle$, then H accepts $\langle M \rangle$.
- If R rejects $\langle M, \langle M \rangle \rangle$, then H rejects $\langle M \rangle$."

What happens if...

M does not accept $\langle M \rangle$?
Accept

M accepts $\langle M \rangle$?
Reject or **Loop**

H is a TM for L_D!

Summary

- We *finally* have found concrete examples of unsolvable problems!
- We are starting to see a line of reasoning we can use to find unsolvable problems:
  - Start with a known unsolvable problem.
  - Try to show that the unsolvability of that problem entails the unsolvability of other problems.

Language of a TM

The language of a Turing machine M, denoted $\mathscr{L}(M)$, is the set of all strings that M accepts:

$$\mathscr{L}(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$$

For any $w \in \mathscr{L}(M)$, M accepts w.

For any $w \notin \mathscr{L}(M)$, M does not accept w.

- It might loop forever, or it might explicitly reject.

A language is called **recognizable** if it is the language of some TM.

Notation: **RE** is the set of all recognizable languages.

$$L \in \mathbf{RE} \text{ iff } L \text{ is recognizable}$$

Why Recognizable

Given TM M with language $\mathscr{L}(M)$, running M on a string w will not necessarily tell you whether $w \in \mathscr{L}(M)$.

If the machine is running, you can't tell whether

- It is eventually going to halt, but just needs more time, or
- It is never going to halt.

However, if you know for a fact that $w \in \mathscr{L}(M)$, then the machine can confirm this (it eventually accepts).

The machine can't *decide* whether or not $w \in \mathscr{L}(M)$, but it can *recognize* strings that are in the language.

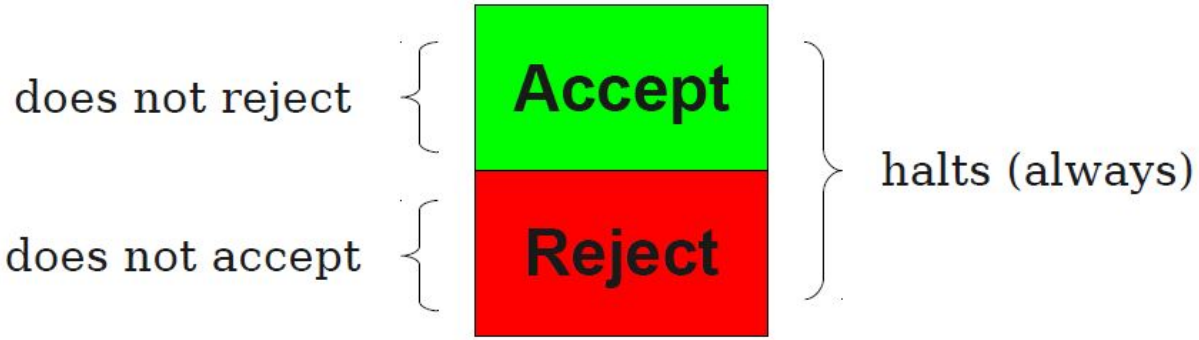
We sometimes call a TM for a language L a **recognizer** for L.

Deciders

Some Turing machines always halt; they never go into an infinite loop.

Turing machines of this sort are called **deciders**.

For deciders, accepting is the same as not rejecting and rejecting is the same as not accepting.



Decidable Language

A language L is called **decidable** iff there is a decider M such that $\mathscr{L}(M) = L$.

Given a decider M, you *can* learn whether or not a string $w \in \mathscr{L}(M)$.

- Run M on w.
- Although it might take a staggeringly long time, M will eventually accept or reject w.

The set **R** is the set of all decidable languages.

$L \in \mathbf{R}$ iff L is decidable

R and RE Languages

Intuitively, a language is in **RE** if there is some way that you could exhaustively search for a proof that $w \in L$.

- If you find it, accept!
- If you don't find one, keep looking!

Intuitively, a language is in **R** if there is a concrete algorithm that can determine whether $w \in L$.

- It tends to be *much* harder to show that a language is in **R** than in **RE**.

Examples of R Languages

All regular languages are in **R**.

- If L is regular, we can run the DFA for L on a string w and then either accept or reject w based on what state it ends in.

$\{ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N} \}$ is in **R**.

- The TM we built last Wednesday is a decider.

Multiplication is in **R**.

- Can check if $m \times n = p$ by repeatedly subtracting out copies of n. If the equation balances, accept; if not, reject.

Why R matters

If a language is in **R**, there is an algorithm that can decide membership in that language.

- Run the decider and see what it says.

If there is an algorithm that can decide membership in a language, that language is in **R**.

- By the Church-Turing thesis, any effective model of computation is equivalent in power to a Turing machine.
- Thus if there is *any* algorithm for deciding membership in the language, there must be a decider for it.
- Thus the language is in **R**.

A language is in R iff there is an algorithm for deciding membership in that language.

Is R = RE?

Every decider is a Turing machine, but not every Turing machine is a decider.

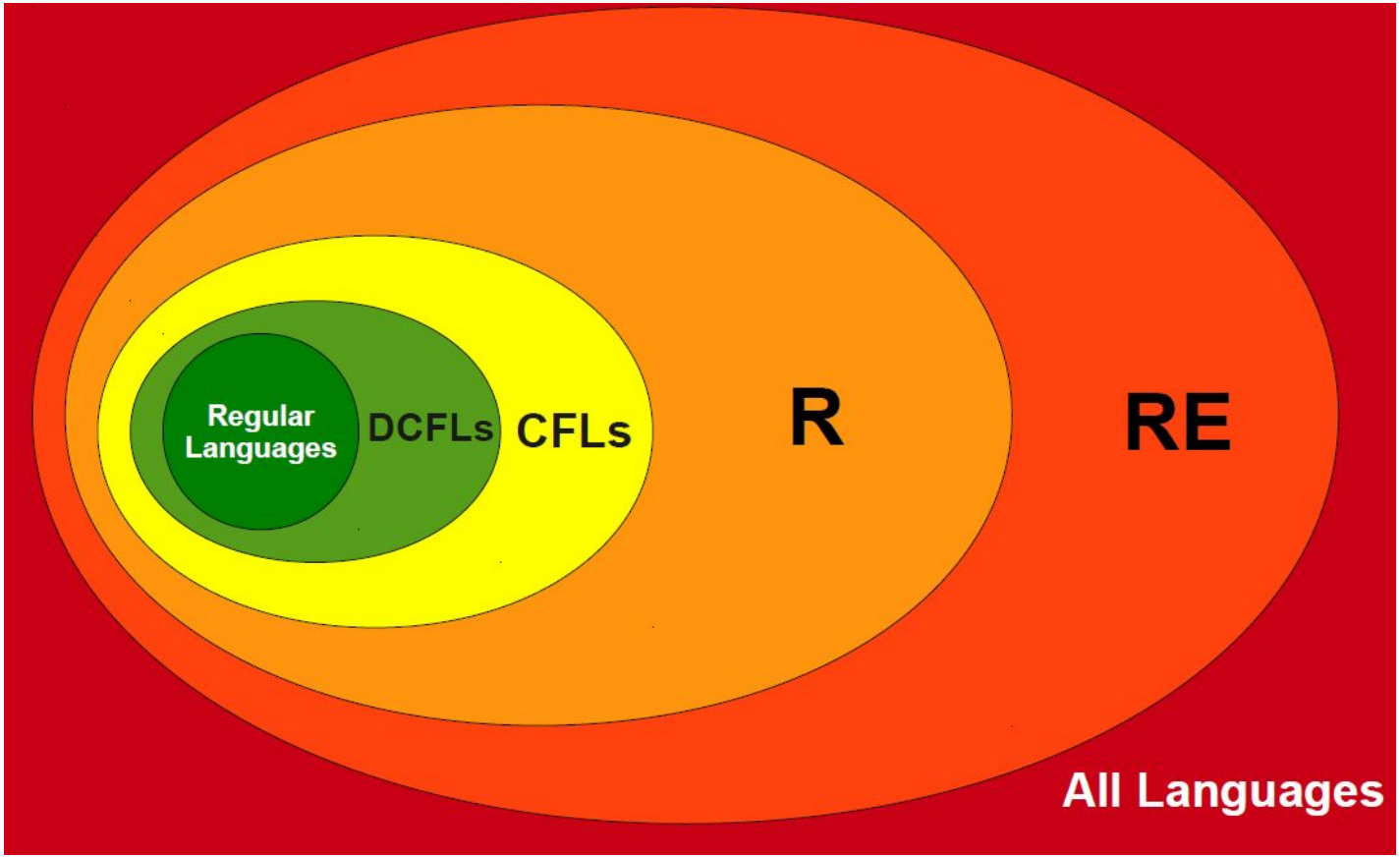
Thus $\mathbf{R} \subseteq \mathbf{RE}$.

Hugely important theoretical question:

Is R = RE?

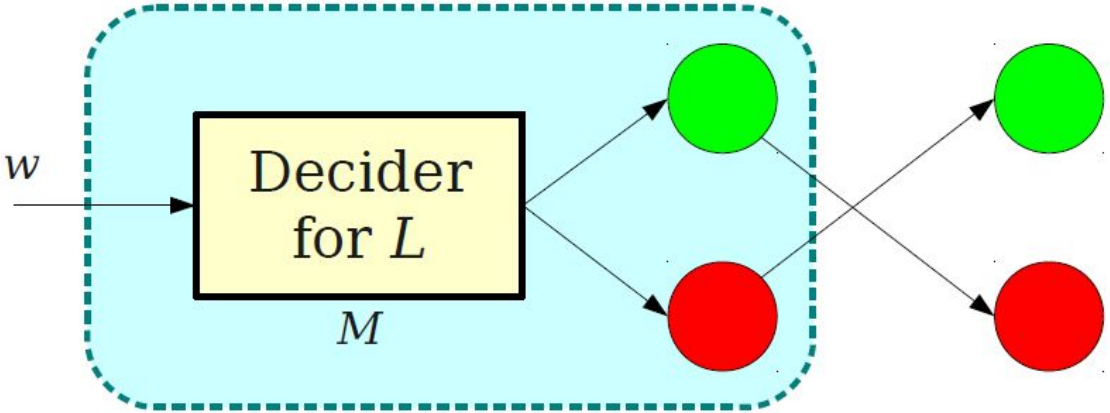
That is, if we can *verify* that a string is in a language, can we *decide* whether that string is in the language?

R and RE Languages



R is closed under complementation

If $L \in \mathbf{R}$, then $\overline{L} \in \mathbf{R}$ as well.



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

Will this work if M is a recognizer, rather than a decider?

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w. If M rejects w, M' accepts w. Thus M' always accepts or rejects, so M' is a decider.

To show that $\mathscr{L}(M') = \overline{L}$, we will prove that M' accepts w iff $w \in \overline{L}$. Note that M' accepts w iff $w \in \Sigma^*$ and M rejects w. Since M is a decider, M rejects w iff M does not accept w. M does not accept w iff $w \notin \mathscr{L}(M)$. Thus M' accepts w iff $w \in \Sigma^*$ and $w \notin \mathscr{L}(M)$, so M' accepts w iff $w \in \overline{L}$. Therefore, $\mathscr{L}(M') = \overline{L}$.

Since M' is a decider with $\mathscr{L}(M') = \overline{L}$, we have $\overline{L} \in \mathbf{R}$, as required. ■

Is R = RE?

We can now resolve the question of $\mathbf{R} \stackrel{?}{=} \mathbf{RE}$.

If $\mathbf{R} = \mathbf{RE}$, we need to show that if there is a recognizer for *any* **RE** language L, there has to be a decider for L.

If $\mathbf{R} \neq \mathbf{RE}$, we just need to find a single language in **RE** that is not in **R**.

A_{TM}

Recall: the language A_{TM} is the language of the universal Turing machine U_{TM}.

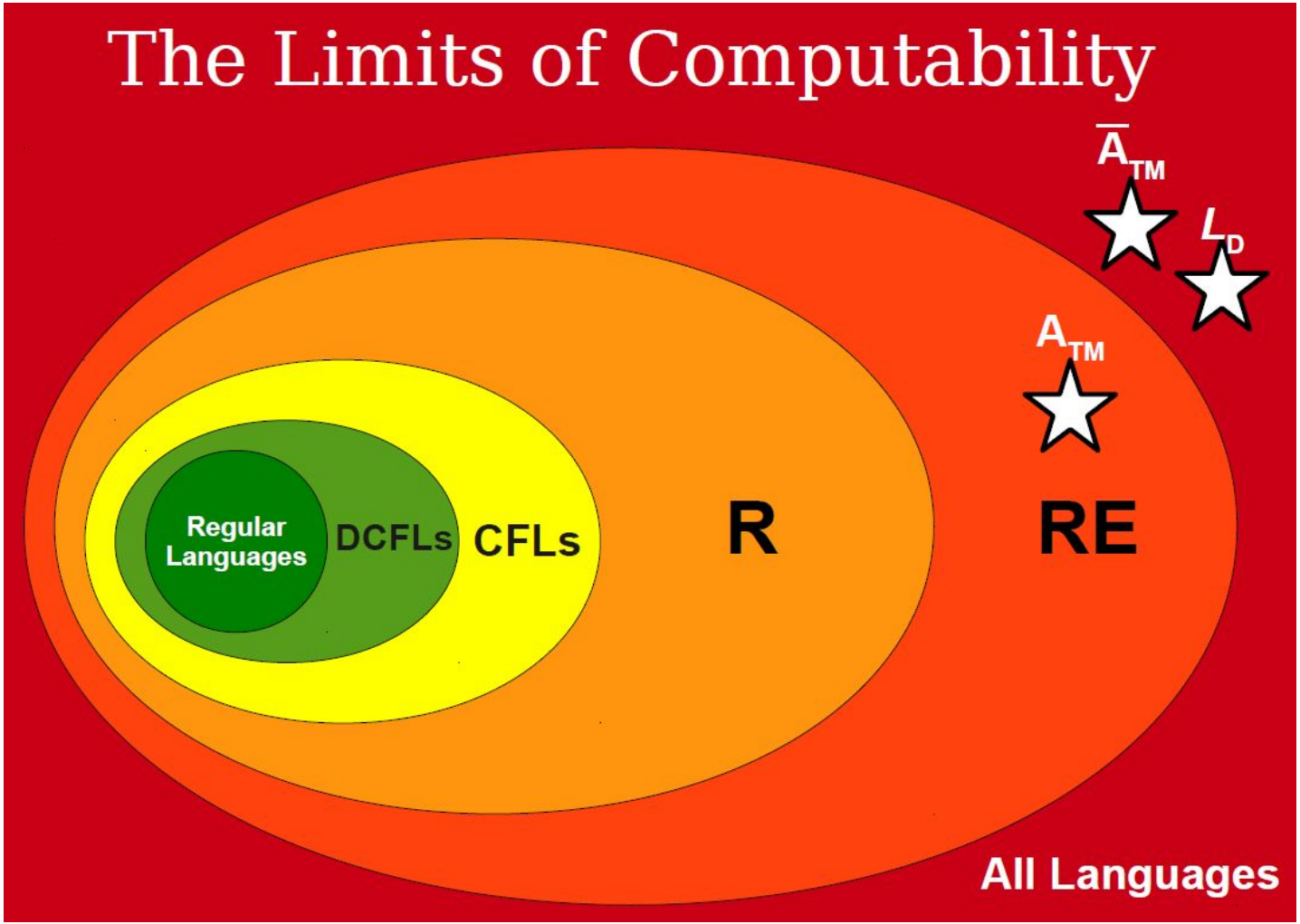
Consequently, $A_{TM} \in \mathbf{RE}$.

Is $A_{TM} \in \mathbf{R}$?

Theorem: $A_{TM} \notin \mathbf{R}$.

Proof: By contradiction; assume $A_{TM} \in \mathbf{R}$. Since **R** is closed under complementation, this means that $\overline{A}_{TM} \in \mathbf{R}$. Since $\mathbf{R} \subseteq \mathbf{RE}$, this means that $\overline{A}_{TM} \in \mathbf{RE}$. But this is impossible, since we know $\overline{A}_{TM} \notin \mathbf{RE}$.

We have reached a contradiction, so our assumption must have been incorrect. Thus $A_{TM} \notin \mathbf{R}$, as required. ■


The Limits of Computability
$\overline{A}_{TM}$
L_D
A_{TM}
Regular Languages
DCFLs
CFLs
R
RE
All Languages

Some More

The undecidability of A_{TM} means that we cannot "cheat" with Turing machines.

We cannot necessarily build a TM to do an exhaustive search over a space (i.e. a recognizer), then decide whether it accepts without running it.

Intuition: In most cases, you cannot *decide* what a TM will do without running it to see what happens.

In some cases, you can *recognize* when a TM has performed some task.

In some cases, you can't do either. For example, you cannot always recognize that a TM will not accept a string.

Some More

Major result: R ≠ RE.

There are some problems where we can only give a "yes" answer when the answer is "yes" and cannot necessarily give a yes-or-no answer.

Solving a problem is *fundamentally harder* than recognizing a correct answer.

The complement of Diagonalization of the Language

The language L_D is the language

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

Therefore, $\overline{L}_D$ is the language

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

Two questions:

- What is this language?
- Is this language **RE**?

$\overline{L}_D \in \mathbf{RE}$

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

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

  Run M on $\langle M \rangle$.

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

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

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

Is $\overline{L}_D$ Decidable?

- We know that $\overline{L}_D \in \mathbf{RE}$. Is $\overline{L}_D \in \mathbf{R}$?
- **No** – by a similar argument from before.
  - If $\overline{L}_D \in \mathbf{R}$, then $\overline{\overline{L}}_D = L_D \in \mathbf{R}$.
  - Since $\mathbf{R} \subset \mathbf{RE}$, this means that $L_D \in \mathbf{RE}$.
  - This contradicts that $L_D \notin \mathbf{RE}$.
  - So our assumption is wrong and $\overline{L}_D \notin \mathbf{R}$.

Rice's Theorem

Every non-trivial (answer is not known) problem on Recursive Enumerable languages is undecidable.e.g.; If a language is Recursive Enumerable, its complement will be recursive enumerable or not is undecidable.

Reducibility and Undecidability

Language A is reducible to language B (represented as A≤B) if there exists a function f which will convert strings in A to strings in B as:

$$w \in A \iff f(w) \in B$$

Theorem 1: If A≤B and B is decidable then A is also decidable.

Theorem 2: If A≤B and A is undecidable then B is also undecidable.

Question: Which of the following is/are undecidable?

1. G is a CFG. Is L(G)=ϕ?
2. G is a CFG. Is L(G)=∑*?
3. M is a Turing machine. Is L(M) regular?
4. A is a DFA and N is an NFA. Is L(A)=L(N)?

A. 3 only

B. 3 and 4 only

C. 1, 2 and 3 only

D. 2 and 3 only

- Option 1 is whether a CFG is empty or not, this problem is decidable.
- Option 2 is whether a CFG will generate all possible strings (everything or completeness of CFG), this problem is undecidable.
- Option 3 is whether language generated by TM is regular is undecidable.
- Option 4 is whether language generated by DFA and NFA are same is decidable.

So option D is correct.

Question: Which of the following problems are decidable?

1. Does a given program ever produce an output?
2. If L is context free language then L' is also context free?
3. If L is regular language then L' is also regular?
4. If L is recursive language then L' is also recursive?

A. 1,2,3,4

B. 1,2

C. 2,3,4

D. 3,4

- As regular and recursive languages are closed under complementation, option 3 and 4 are decidable problems.
- Context free languages are not closed under complementation, option 2 is undecidable.
- Option 1 is also undecidable as there is no TM to determine whether a given program will produce an output. **So, option D is correct.**

Question: Consider three decision problems P1, P2 and P3. It is known that P1 is decidable and P2 is undecidable. Which one of the following is TRUE?

A. P3 is undecidable if P2 is reducible to P3

B. P3 is decidable if P3 is reducible to P2's complement

C. P3 is undecidable if P3 is reducible to P2

D. P3 is decidable if P1 is reducible to P3

- Option A says P2≤P3. According to theorem 2 discussed, if P2 is undecidable then P3 is undecidable. It is given that P2 is undecidable, so P3 will also be undecidable. So option **(A) is correct**.
- Option C says P3≤P2. According to theorem 2 discussed, if P3 is undecidable then P2 is undecidable. But it is not given in question about undecidability of P3. So option **(C) is not correct.**
- Option D says P1≤P3. According to theorem 1 discussed, if P3 is decidable then P1 is also decidable. But it is not given in question about decidability of P3. So option **(D) is not correct**.
- Option (B) says P3≤P2'. According to theorem 2 discussed, if P3 is undecidable then P2' is undecidable. But it is not given in question about undecidability of P3. So option **(B) is not correct**.

Church Turing Thesis

Turing machine is defined as an abstract representation of a computing device such as hardware in computers. Alan Turing proposed Logical Computing Machines (LCMs), i.e. Turing's expressions for Turing Machines. This was done to define algorithms properly. So, Church made a mechanical method named as 'M' for manipulation of strings by using logic and mathematics.

This method M must pass the following statements:

- Number of instructions in M must be finite.
- Output should be produced after performing finite number of steps.
- It should not be imaginary, i.e. can be made in real life.
- It should not require any complex understanding.

Using these statements Church proposed a hypothesis called **Church's Turing thesis** that can be stated as: "The assumption that the intuitive notion of computable functions can be identified with partial recursive functions."