San José State University
Department of Computer Science

Ahmad Yazdankhah
ahmad.yazdankhah@sjsu.edu
www.cs.sjsu.edu/~yazdankhah

Computability

Lecture 26

Day 30/31

CS 154

Formal Languages and Computability

Spring 2019

Agenda of Day 30

- Feedback and Solution of Quiz 10
- About Final Exam
- Summary of lecture 25
- Lecture 26: Teaching ...
  - Computability

Solution and Feedback of Quiz 1 (Out of 15)

Section	Average	High Score	Low Score
01 (TR 3:00 PM)	10.35	14	6
02 (TR 4:30 PM)	10	14	3
03 (TR 6:00 PM)	10.87	15	1

About Final Exam

Reminder 1

- **Value**: 20%
- **Topics**: Almost everything covered from the beginning of the semester
- **Type**: Closed all means

Section	Date	Time	Venue
01 (TR 3:00)	Tuesday, May 21	2:45 – 5:00 pm	DH 450
02 (TR 4:30)	Monday, May 20	2:45 – 5:00 pm	DH 450
03 (TR 6:00)	Thursday, May 16	5:15 – 7:30 pm	DH 450

- We won't need whole 2:15 hours.
- As usual, I'll announce officially the type and number of questions via Canvas. (study guide)

Summary of Lecture 26: We learned ...

Non-Regular Languages

- We introduced an important property of infinite regular languages.
- We stated it as a theorem called pumping lemma.

Conclusions

1. Pumping lemma is NOT applicable to finite languages.
2. If an infinite language does not satisfy pumping lemma, it is non-regular.

Pumping Lemma Statement

If L is an infinite regular language,

Then

(1) There exists an $m \geq 1$ such that

If (2) $w \in L$ and (3) $|w| \geq m$

Then //P. L. guarantees that ...

(4) We must be able to divide w into xyz in such a way that all of the following conditions are satisfied:

(5) $|xy| \leq m$, and

(6) $|y| \geq 1$, and

(7) $w_i = x\, y^i\, z \in L$

(8) w_i for $i = 0, 1, 2, \ldots$

Summary of Lecture 26: We learned ...

How to Prove a Language is Non-Regular?

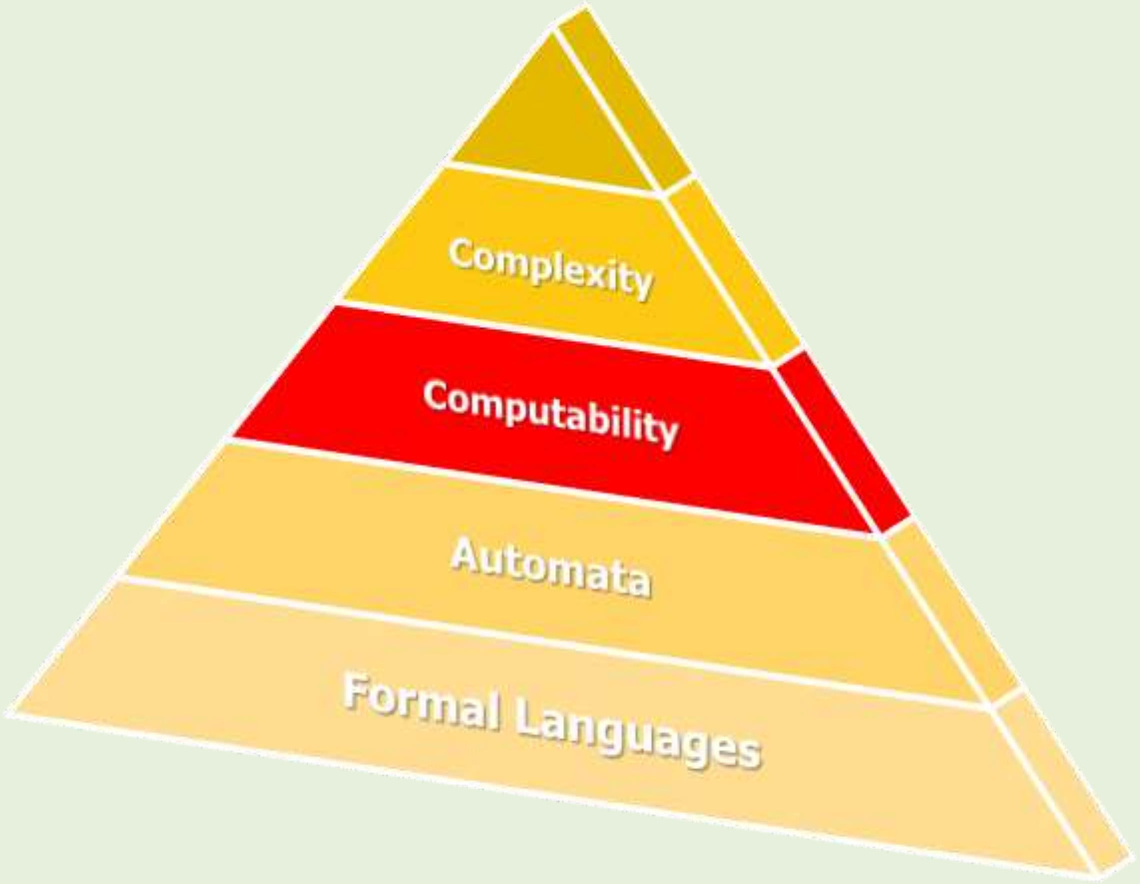
- We use proof by contradiction:
  - Assume it is regular
  - Apply pumping lemma
  - Find a contradiction
  - Blame your assumption

- Pumping lemma CANNOT PROVE that a languages is regular.

Any Question

The Big Picture of the Course

Recap

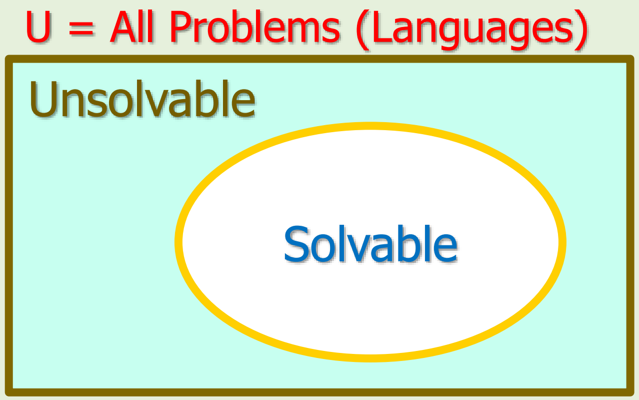


Objective of This Lecture

- Is there an algorithmic solution for any kind of problems?
- Is there any unsolvable problems?

- Like any other tools, computers have capabilities and limitations.
- In this lecture, we want to show that:

  Some problems are beyond the theoretical limits of computation.



Warning!

- To understand these material, you'd need 200% attention!

- Don't panic if you see some black smoke ...
- That's your brain that's burning!

- Please call your parents and give them heads up!
- Because when you reach at home, you look like this ...!

Turing's Thesis

- Alan Turing in 1936 gave a conjecture, now known as "Turing's thesis".
- He gave us a high-level hint about what type of problems are computable.

Turing's Thesis

- Any computation carried out by a "mechanical procedure" can be performed by a TM.

- A procedure P is mechanical if:
  - P has a finite number of instructions.
  - P produces the desired result in a finite number of steps (no infinite loop).
  - P can be carried out by a human being (no specific ingenuity required) unaided by any machinery

Turing's Thesis

- We cannot prove Turing thesis and also we could not refute it yet.
  - Exactly the same way that we cannot prove Newton's laws of motion in physics.
  - Newton's laws is a model that explains much of the physical world.

- So, we claim:

  Turing thesis is true as long as we cannot find a counterexample.

Turing's Thesis

Repeated

- Any computation carried out by a "mechanical procedure" can be performed by a TM.

Turing's Thesis: Notes

1. In Spring 2017, we simulated a microprocessor as a term project to prove Turing thesis experimentally.

2. In some documents, the thesis might be called Church-Turing thesis.
   - But we prefer to call it "Turing's Thesis"!

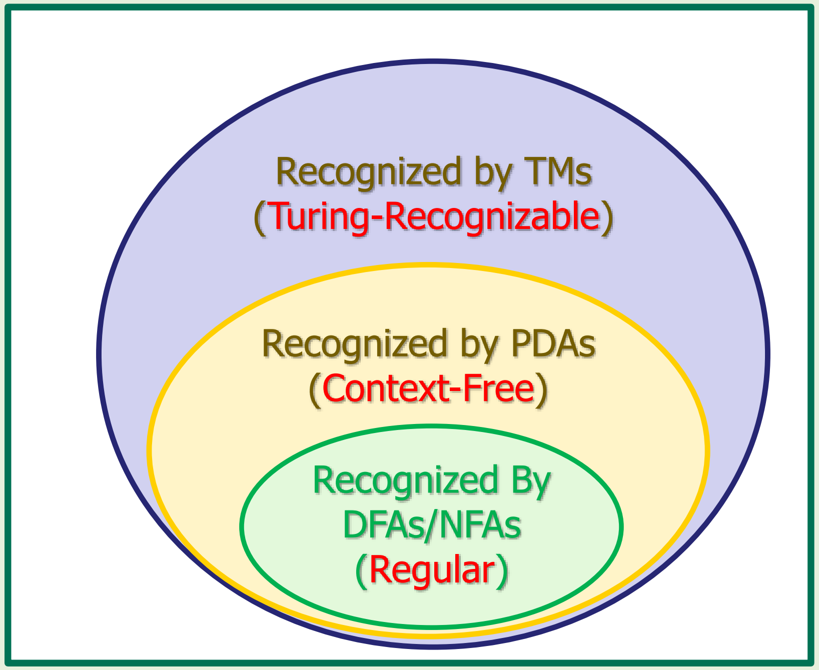
- Before talking about computability,
  we need some backgrounds that should be covered first.

Decidability

Turing Recognizable Languages

- We've learned before that ...
- The languages recognized by TMs are called "Turing recognizable" (aka recursively enumerable).

U = All Formal Languages



Turing Recognizable Languages

- Precisely speaking:
- A language L is called "Turing recognizable" if ...
- ... A TM halts in accepting-states to accept the strings of L.

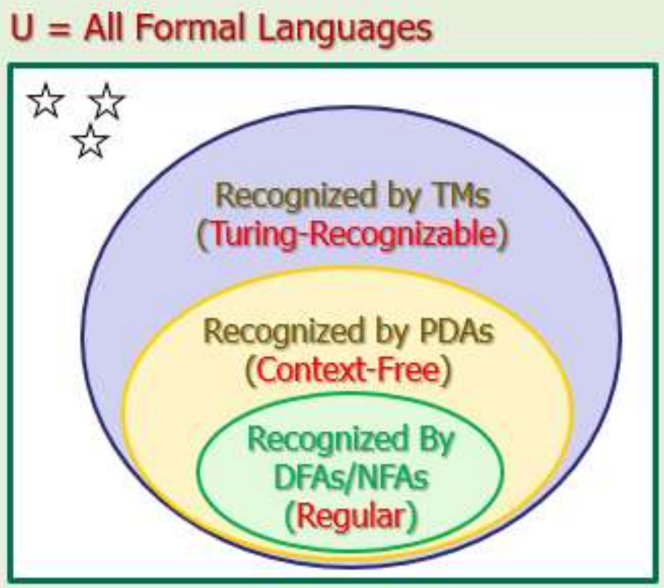
- But to reject the strings of $\overline{L}$, it ...

  ... halts in non-accepting-states,

  OR

  ... gets stuck in infinite-loops!

- And this is a problem! Why?


U = All Formal Languages
Recognized by TMs (Turing-Recognizable)
Recognized by PDAs (Context-Free)
Recognized By DFAs/NFAs (Regular)

The Problem of Turing Recognizable Languages

- Because we mentioned before, and will mention today that ...
- It is impossible to distinguish whether ....

  ... a machine is in an infinite-loop,

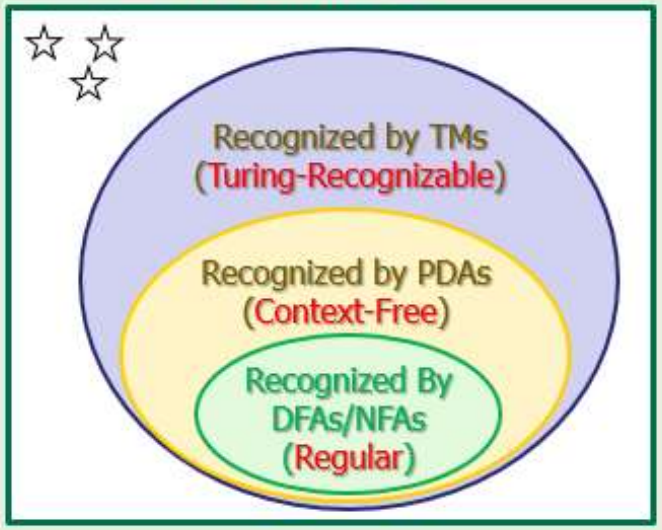
  OR

  ... it is in the middle of a very long computation?

  We'll prove this today!

- So, because of this ambiguity,
  we prefer TMs that ...
  always halt on all inputs.

U = All Formal Languages


☆ ☆ ☆
Recognized by TMs (Turing-Recognizable)
Recognized by PDAs (Context-Free)
Recognized By DFAs/NFAs (Regular)

Turing Decidable Languages

- Those TMs that always halt (for accepting and rejecting) are called "deciders".
  - Because they can always make a decision.

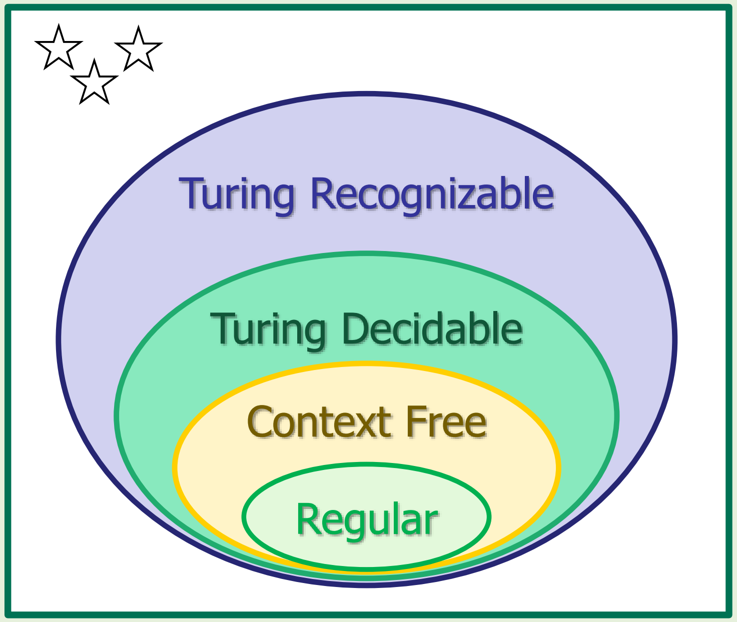
- A language is called "Turing-decidable" or simply "decidable" (aka recursive) if there is a decider for it.

- Let's take another look at the hierarchy of languages!

Formal Languages Hierarchy

- Our new hierarchy of languages!

U = All Formal Languages



- Let's see the difference via some examples.
- Before that, we'd need some background ...

Encoding

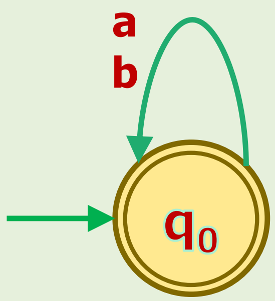
- The input to a TM should be a string.
- So, if we want to input "objects" (e.g. graphs) other than strings, then we have to convert them to strings.
- Converting an object to a string is called "encoding".

- In your term project, you have used this technique.

Example 1

- The following string shows the encoded string of this DFA.

  D10101D101101F1



Encoding Notation

- Encoded object A is denoted by ⟨A⟩.
- If we want to encode several objects, e.g. $A_1, A_2, \ldots, A_k$, then we show them as:

$$\langle A_1, A_2, \ldots, A_k \rangle$$

Example 2

- The combination of a DFA M and a string w is denoted by ⟨M, w⟩.
  - w is the input string of M.

Encoding: Notes

1. By using this technique, we can encode all objects in CS such as:

   Numbers, polynomials, graphs, grammars, automata, formal languages, matrices, etc., and any combination of these objects.

2. The encoding process can be done in many ways.

   For example, we could encode digital circuits by binary numbers.

3. It does not matter what encoding system we use.

   The result of the computation must be the same regardless of how we encode them.

Decidability Examples

Example 3

- Given language L_{DFA} as:

  $L_{DFA} = \{\langle M, w\rangle : M \text{ is a DFA that accepts input string } w\}$

- Is this a decidable language? (Is there a decider for this language?)

- Since DFAs always halt and never falls into infinite-loops,
  so, the TM that simulates DFAs' operation should always halt too.

- In other words, the TM always can decide whether the DFA accepts w or not.

- In Fall 2017, we created a TM that could simulate DFAs operation.

Decidability Examples

Example 4

- Is the following language decidable?

  $L_{REX} = \{\langle R, w\rangle : R \text{ is a REGEX that generates string } w\}$

- There is a theorem (we did not mention it) by which we can convert a REGEX to an NFA.
- Also, there is a theorem (we mentioned it before) by which we can convert an NFA to a DFA.
- By using these two theorems, we can convert the above problem to L_{DFA} (the previous example).

So, L_{REX} is decidable.

Decidability Examples

Example 5

- Is the following language decidable?

  $L_{TM} = \{\langle M, w\rangle : M \text{ is a TM that accepts string } w\}$

- No, because TMs can fall into infinite-loops and consequently, the TM, called UTM, that simulates those TMs will fall into infinite-loops too.

- Let's describe what could happen ...

Decidability Examples

Example 5 (cont'd)

- Precisely speaking, when UTM is simulating M against w, there would be three possibilities:

  1. M accepts w and halts, so UTM halts in an accepting state.
  2. M rejects w and halts, so UTM halts in a non-accepting state.
  3. M rejects w by falling into an infinite-loop and so as UTM.

- So, UTM has the possibility of falling into infinite-loops.
- That's why L_{TM} is not decidable but is recognizable.

Universal Turing Machine

- UTM in the previous example is called "universal Turing machine".
- It is called universal because it is capable of simulating other TMs.
- It was proposed by Alan Turing in 1936.
- It played a significant role in the development of stored-program computers (our current computers).

Halting Problem

Introduction

- Halting problem is one of the most famous problems in computer science.
- This is an example of the limitations of computation.
- This problem was introduced and proved by Alan Turing in 1936.
- To understand halting problem,
  we need to review some paradoxes for warmup.

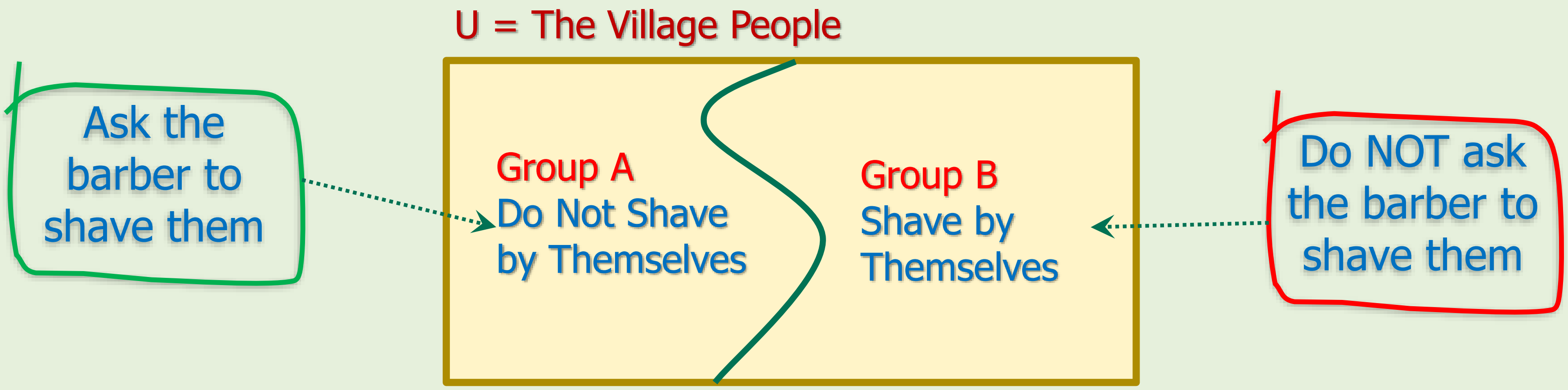
A Weird Game

- This game has two simple rules:
  1. Move if you see a still person.
  2. Stop if you see a moving person.

- Now, look at yourself through a mirror and decide what to do!
- Will you move or stop?!

The Barber Paradox

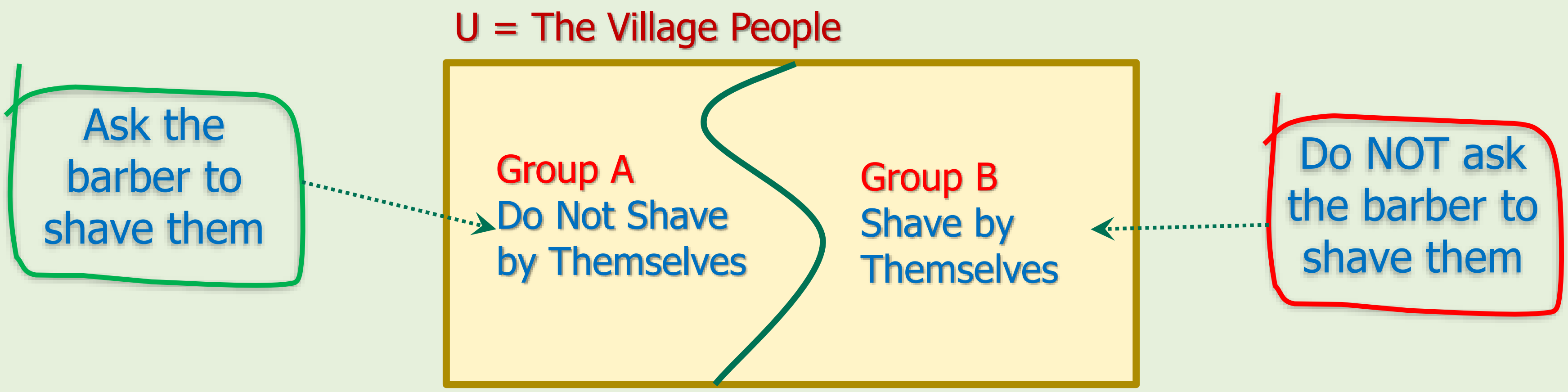
- There exists an isolated village with only one barber with two strict rules:
  1. If someone does NOT shave by himself, he should ask the barber to shave him. (Group A)
  2. If someone shave by himself, he should NOT ask the barber to shave him. (Group B)


U = The Village People
Ask the barber to shave them
Group A
Do Not Shave by Themselves
Group B
Shave by Themselves
Do NOT ask the barber to shave them

- Question: should the barber shave himself?

The Barber Paradox

- If the barber shaves, then he would belong to group B that they should not ask the barber to shave them.
- So, he should not shave himself!
- If the barber does not shave, then he would belong to group A that they should ask the barber to shave them.
- So, he should shave himself!


U = The Village People
Ask the barber to shave them
Group A
Do Not Shave by Themselves
Group B
Shave by Themselves
Do NOT ask the barber to shave them

The Barber Paradox

- What is the resolution of this kind of paradoxes?

- The important fact is that:

  This physical universe in which we are living,
  does not permit contradiction.

- So, if any assumption leads to a contradiction, then we have to conclude that the assumption was wrong.

- For this paradox, the assumption ...
  "there exists an isolated village ..." is wrong.

- In other words, there should not be any village with these stupid strict rules in this universe!

A Classical Paradox

- Is the following sentence true or false?

  p ≡ This sentence is false!

- If p is true, then it should be false!
  - Because the sentence itself is saying it is false!
- If p is false, then it should be true!
  - Because if p is false, then its negation should be true.
  - The negation of the sentence is "This sentence is true!".

Bertrand Russell's Paradox

- British philosopher, logician, mathematician, historian, writer, social critic, political activist, Bertrand A. W. Russell (1872-1970) introduced the following set in 1901:

- The set of all sets that do not contain themselves.

- To understand this set, let's take some examples.

Bertrand Russell's Paradox

Example 6

- The set of all pens in the world.

  A = {x : x is a pen}

- Does A contain itself?
- No, because the set A IS NOT a pen!

Example 7

- The set of all non-pens in the world.

  B = {x : x is a non-pen}

- Does B contain itself?
- Yes, because the set B IS a non-pen!

Bertrand Russell's Paradox

- Now let's get back to the Russell's set.
- The set of all sets that do not contain themselves.
- This set contains all sets like set A in example 6.
- $R = \{A_1, A_2, A_3, \dots\}$
- $R = \{x : x \text{ is a set}, x \notin x\}$
- Does R contain R (itself)? Can one of those A's be R?

- If $R \notin R$, then $R \in R$!
- If $R \in R$, then $R \notin R$!

$$R \in R \leftrightarrow R \notin R$$

- Again, the resolution of Russell's paradox is to declare that the set R cannot exist!

Self-Referential Paradoxes

- In all aforementioned paradoxes, something was referring to itself.
- That's why these kind of paradoxes are called "self-referential paradoxes".

- A rough conclusion would be:
- If in a problem, something is referring to itself, there could be a problem!

Halting Problem

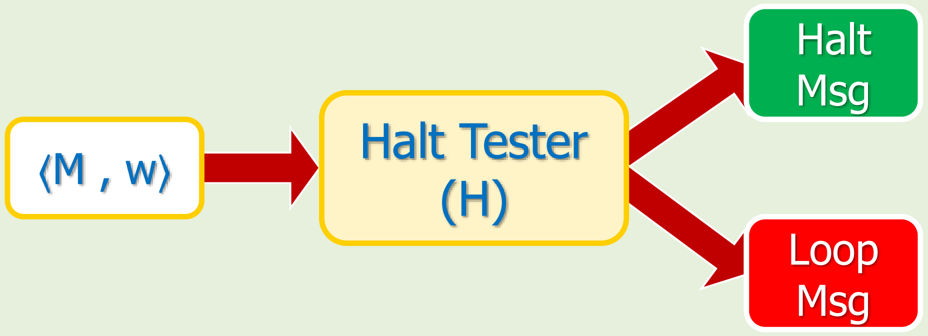
Halting Problem Statement

- Given the following language:

  $L_{HALT} = \{\langle M, w\rangle : M \text{ is a TM that halts on string } w\}$

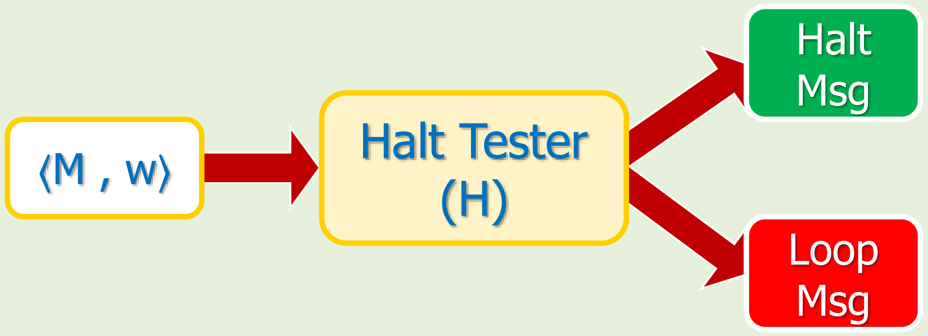
- Is there any decider for L_{HALT}?

- In other words:
- Is there a TM H (= algorithm) to decide whether an arbitrary TM M will halts on an arbitrary string w?



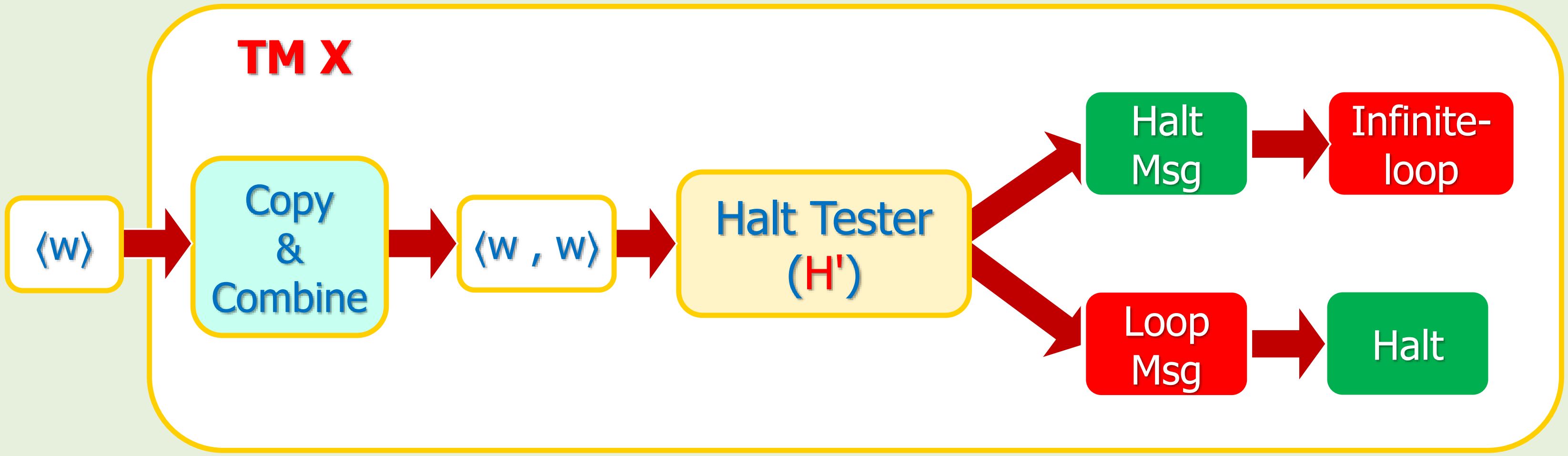
Halting Problem

- Through a theorem, Alan Turing showed TM H (= algorithm) does NOT exist.
- To prove this theorem, we use proof-by-contradiction:
- Assume TM H exists.
- Therefore, it should be able to decide about any arbitrary TM against any arbitrary string.
- Now, we construct another TM as a test-case for testing H's behavior.



Constructing a Test-Case for H

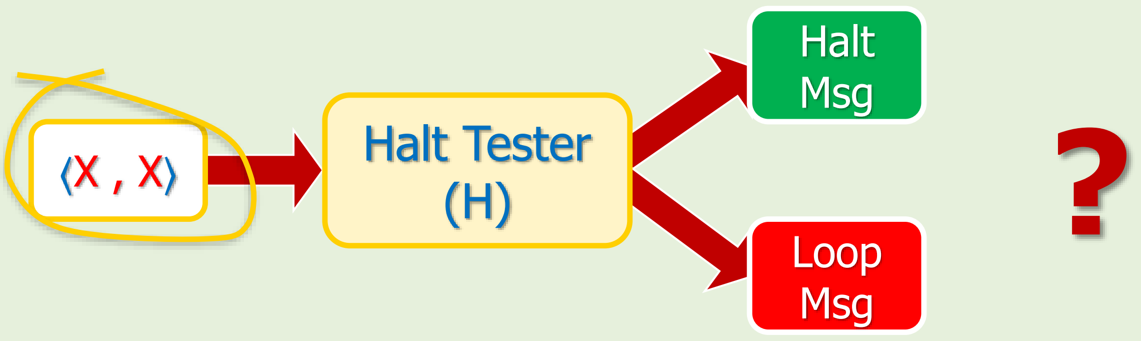
- Use a copy of H, let's call it H', as a subroutine and construct another TM, called TM X as follows:
- Negate the output of H'.
  - If H' output "Halt" message, then it goes to an artificial infinite-loop.
  - If H' output "Loop" message, then it halts and finishes the program.
- Put a "Copy & Combine" at the beginning to copy the input and combine it as <w, w>.



Halting Problem Proof

- H can decide about any arbitrary TM against any arbitrary string w.
- TM X, our test-case, is a TM too.
- Now we input encoded X, both as M and its input w.
- It means, we input ⟨X , X⟩ to H!

- What would be the answer of H for this input?

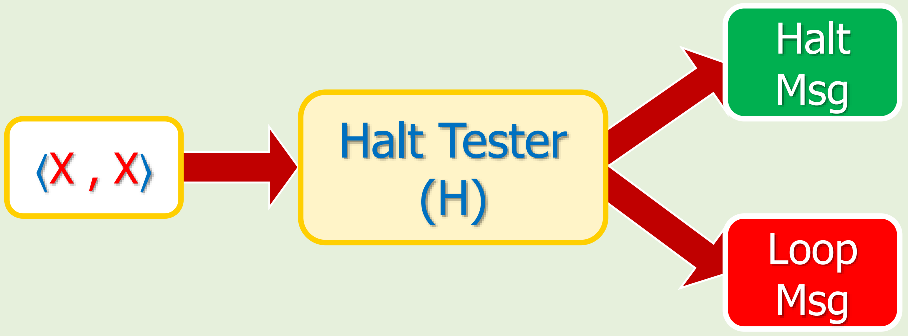


Halting Problem Proof

- There are only two possible answers for H:

1. TM X halts on string X.
2. TM X loops on string X.

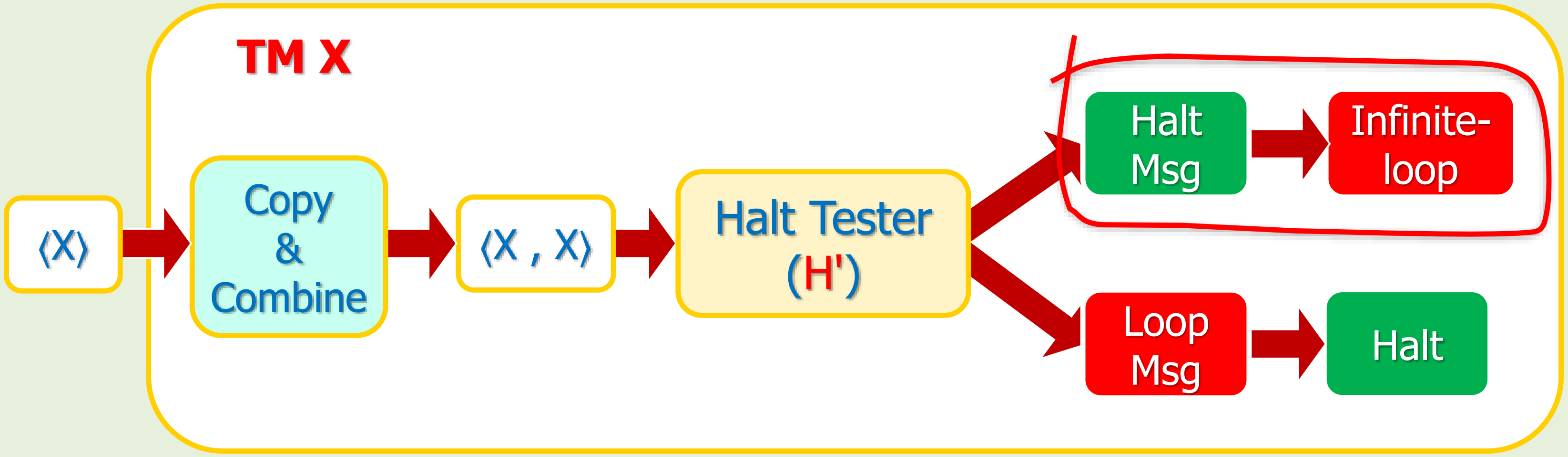
- We want to show that both cases lead to a contradiction.



- Let's examine each case separately ...

Case 1: H Responded "TM X halts on string X"

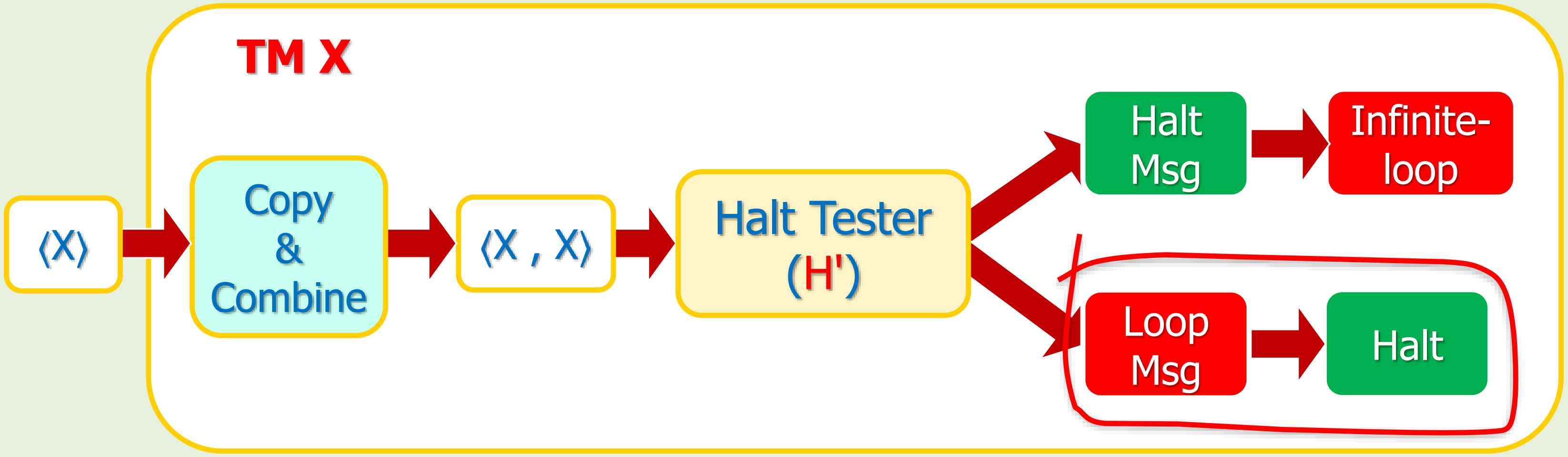
- To verify this response, let's examine X's behavior by inputting string X.



- Recall that H' is a copy of H and should respond the same way.
- So, H' responds "Halt Msg", then it goes to an artificial infinite-loop.
- So, X's behavior is falling in an infinite-loop.
- Therefore, H was wrong! Because it said "TM X halts on string X".

Case 2: H Responded "TM X Loops on string X

- Again, since H' is a copy of H, it should respond the same.



- H' responds "Loop Msg", then it halts.
- Therefore, H was wrong again! Because it said "TM X loops on string X".

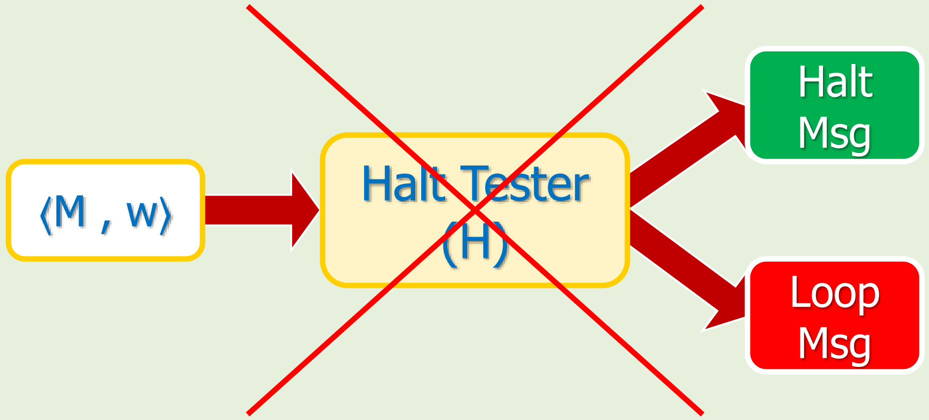
Halting Problem Proof

Conclusion

1. TM X halts on string X ⇒ TM X loops on string X
2. TM X loops on string X ⇒ TM X halts on string X

- Both cases led to contradiction!
- Therefore, our assumption "H exists" was wrong. So, ...

H does not exist!



References

1. Linz, Peter, "An Introduction to Formal Languages and Automata, 5th ed.," Jones & Bartlett Learning, LLC, Canada, 2012
2. Michael Sipser, "Introduction to the Theory of Computation, 3rd ed.," CENGAGE Learning, United States, 2013
   ISBN-13: 978-1133187790
3. Yanofsky, Nosan, "Paradoxes, Contradictions, and the Limits of Science," American Scientist, May 2016
4. YouTube, Proof That Computers Can't Do Everything (The Halting Problem), available at: https://www.youtube.com/watch?v=92WHN-pAFCs