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 2018

Agenda of Day 30

- About Final Exam
- Solution and Feedback of Quiz 10
- No summary for today!
- Lecture 27: Teaching …
  - Computability

About Final Exam

Reminder 2

- **Value**: 20%
- **Topics**: Everything covered from the beginning of the semester
- **Type**: Closed all materials

	Section 1	Section 2	Section 3
Date	Thursday, May 17th	Thursday, May 17th	Tuesday, May 22nd
Time	2:45 - 5:00 pm	5:15 - 7:30 pm	2:45 - 5:00 pm
Venue	MH 233	MH 233	SCI 311

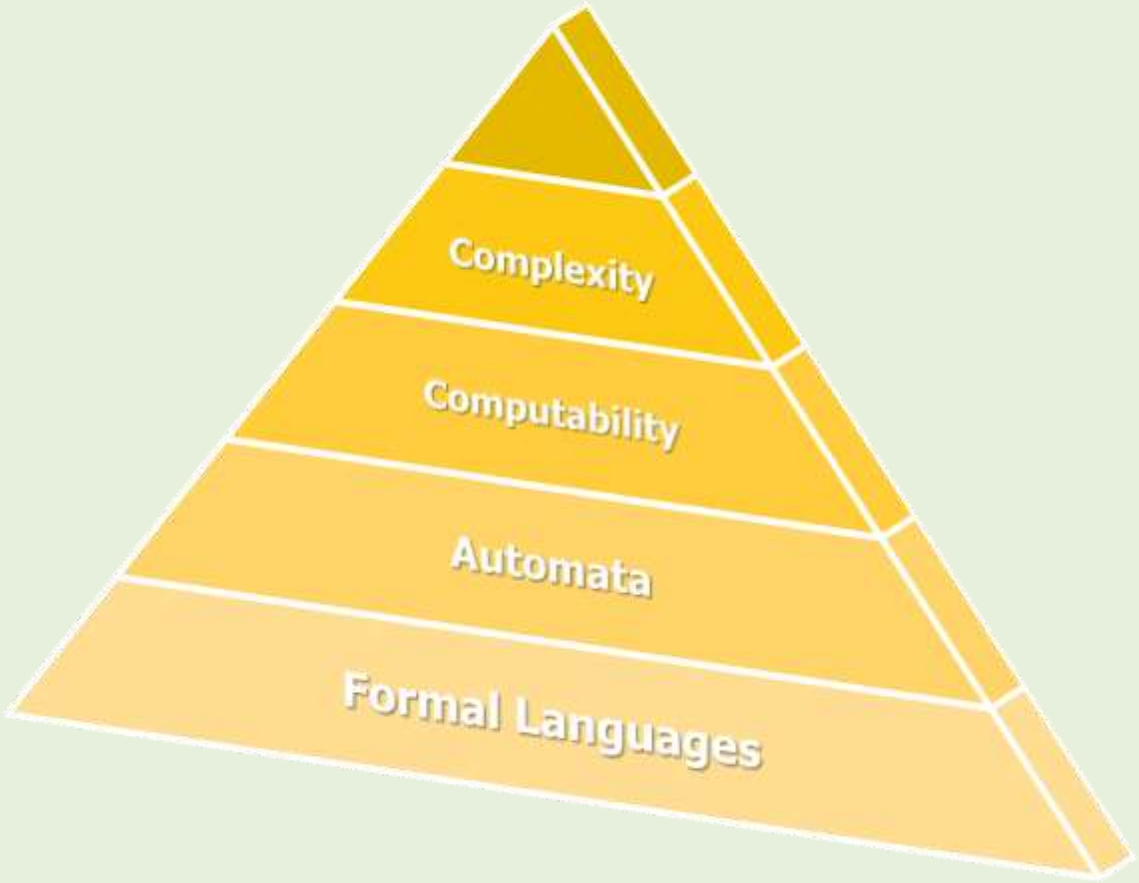
- We won't need whole 2:15 hours.
- As usual, I'll announce officially the type and number of questions via Canvas. (study guide)

Solution and Feedback of Quiz 10 (Out of 30)

Metrics	Section 1	Section 2	Section 3
Average	26	26	28
High Score	29	30	30
Low Score	17	21	17

Big Picture: Computer Science Foundation

Recap

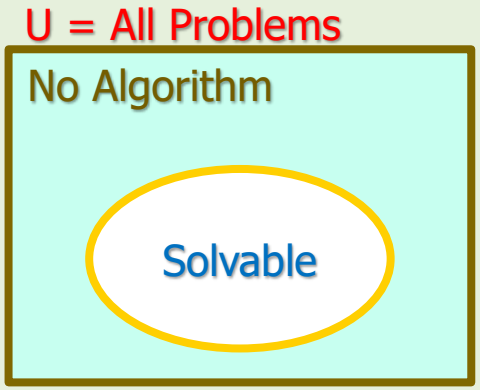


Objective of This Lecture

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

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

  Some problems are beyond the theoretical limits of computation.



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: Notes

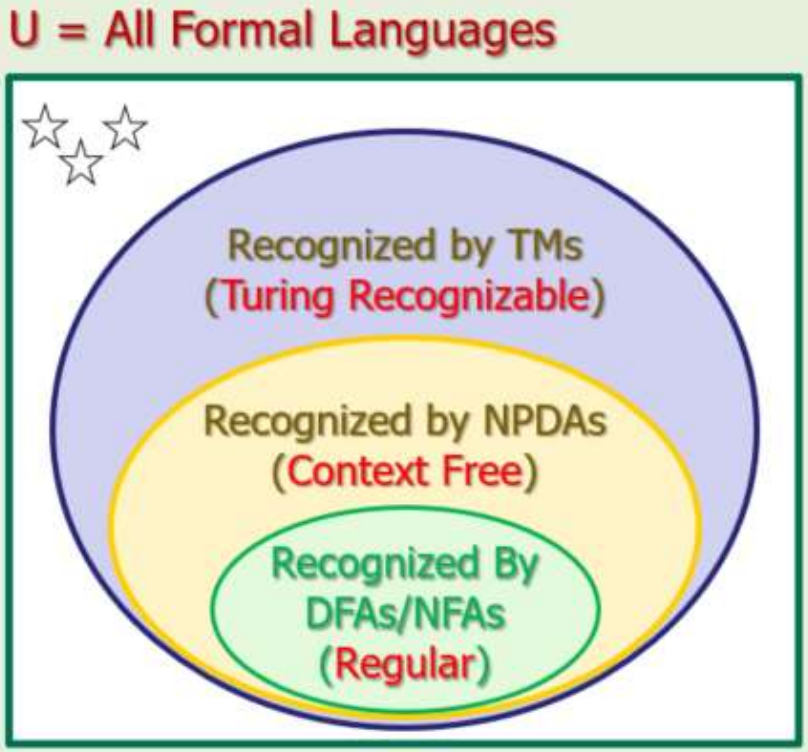
1. In Spring 2017, we simulated a microprocessor as a term project to prove Turing thesis experimentally.

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

Decidability

Turing Recognizable Languages

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



Turing Recognizable Languages

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

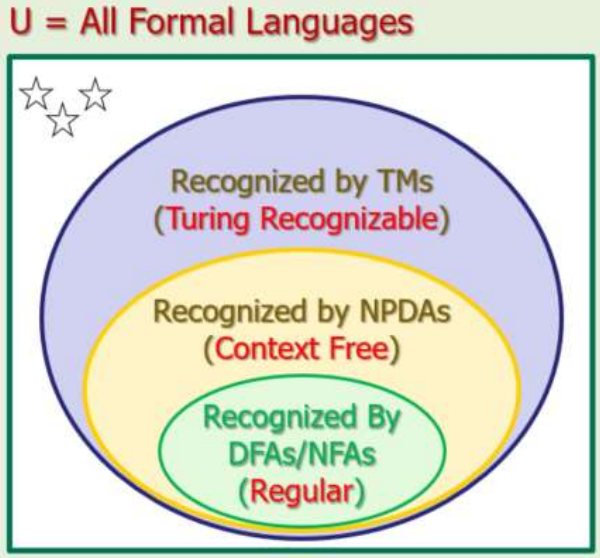
- But to reject the strings of $\overline{L}$, it ...

  ... halts in a non-accepting-state,

  OR

  ... gets stuck in an infinite-loop!

- An this is a problem! Why?



Problem of Turing Recognizable Languages

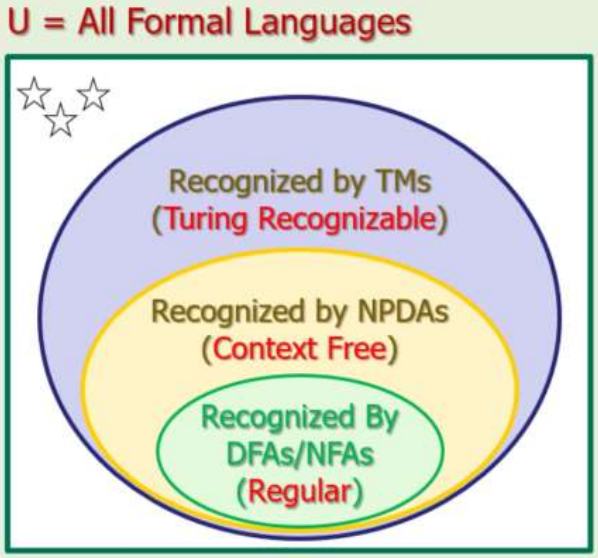
- Because we learned that ...
- It is impossible to distinguish whether ....

  ... 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.



Turing Decidable Languages

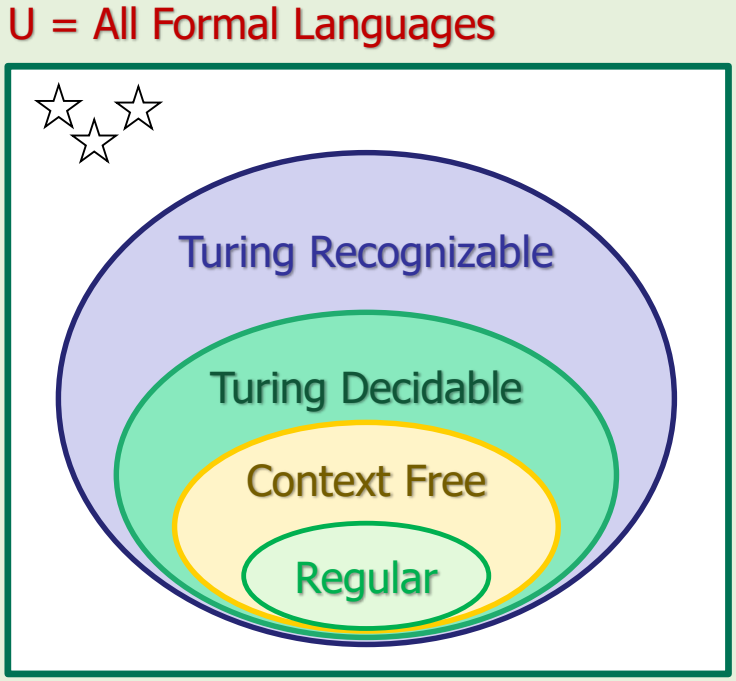
- Those TMs that always halt (for accepting and rejecting) are called "deciders".
  - Because they can always make a decision.

- 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

- The new hierarchy of languages!



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

Encoding

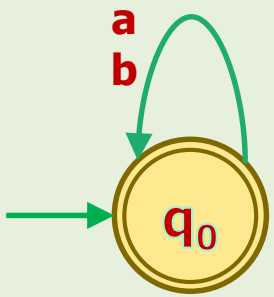
- The input to a TM should be a string.
- So, if we want to input "objects" (e.g. graphs) other than strings, then we have to convert them to strings.
- Converting an object to a string is called "encoding".

- 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 $\langle A \rangle$.
- If we have several types of objects, e.g. A_1, A_2, ..., 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 that we input to it is denoted by $\langle M, w \rangle$.

Encoding: Notes

1. Using this technique, we can easily encode all known objects in CS such as:

   Polynomials, graphs, grammars, automata, formal languages, and/or any combination of these objects.

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

   For example, we could encode DFAs by using binary numbers.

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

   The result of the computation would be the same regardless of the way we encode the objects.

Decidable Problems: Examples

Example 3

- The set of the following strings ⟨M, w⟩ can be considered as a language.

  L_{DFA} = {⟨M, w⟩ : M is a DFA that accepts input string w}

- Is this a decidable language?
- The term project of Fall 2017 semester showed that:

  This is a decidable language.

- Since the DFA always halts and never falls into an infinite-loop, so, the TM that simulates the DFA always halts too.
- The TM always can decide whether the DFA accepts w or not.

Decidable Problems: Examples

Example 4

- Is the following language decidable?

  L_{REX} = {⟨R, w⟩ : R is a REGEX that generates string w}

- There is a theorem (we did not cover) by which we can convert a REGEX to an NFA.
- Also, there is a theorem (we mentioned it but did not prove) 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 too.

Undecidable Problems: 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 that simulates it (let's call it UTM) will fall into an infinite-loop too.

- Let's describe what could happen ...

Undecidable Problems: 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 falls into an infinite-loop and so as UTM.

- So, UTM has the possibility of falling into an infinite-loop.
- 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 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.

- Players don't turn their head around and just look straight.

- 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)

- Question: should the barber shave himself?

The Barber Paradox

- If the barber shave, then he would belong to group B that they should 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!

The Barber Paradox

- What is the resolution of this paradox?

- 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!

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.
- $R = \{x : x \text{ is a set}, x \notin x\}$

- 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.
- $R = \{A_1, A_2, A_3, \ldots\}$
- $R = \{x : x \text{ is a set}, x \notin x\}$
- Does R contain R (itself)?

- 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!

A Classical Paradox

- Is the following statement true or false?

  p ≡ This sentence is false!

- aka "I always tell lie!"

- If p is true, then it should be false!
- If p is false, then it should be true!

- These kind of paradoxes are called "self-referential paradoxes".
- It means, something is referring to itself.

Halting Problem

- Now we're ready to understand "halting problem".
- Is there any TM H that decides the following language?

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

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

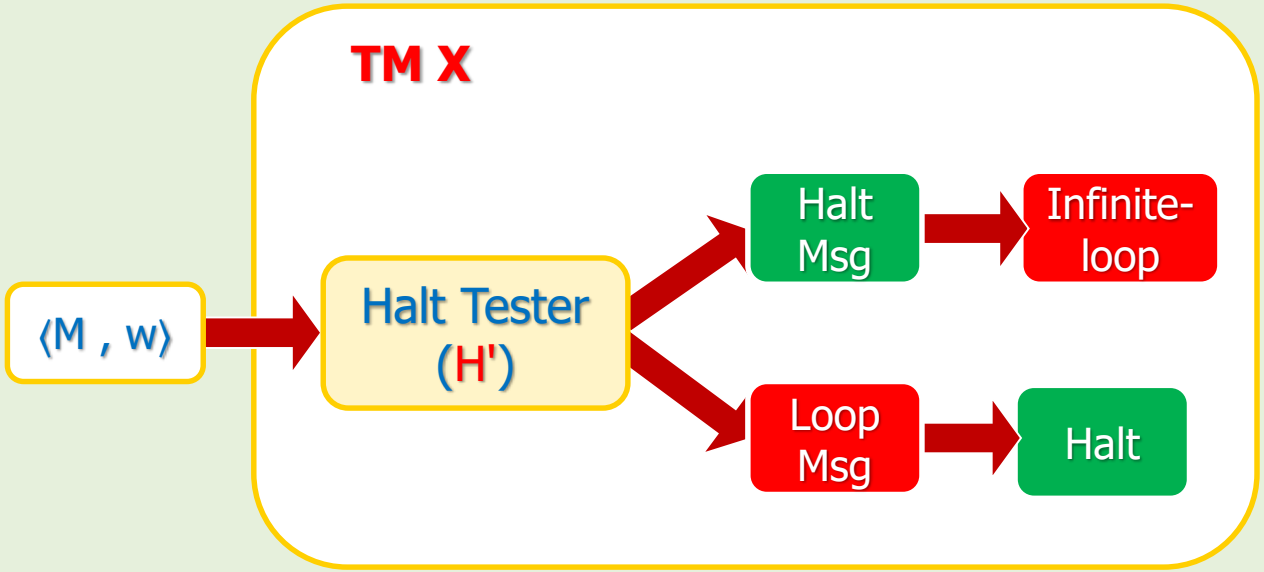
Halting Problem

- Through a theorem, Alan Turing showed H does NOT exist.
- To prove this theorem, we use "proof-by-contradiction".

- Assume H exists.
- Now, we construct another TM as a test-case for H.

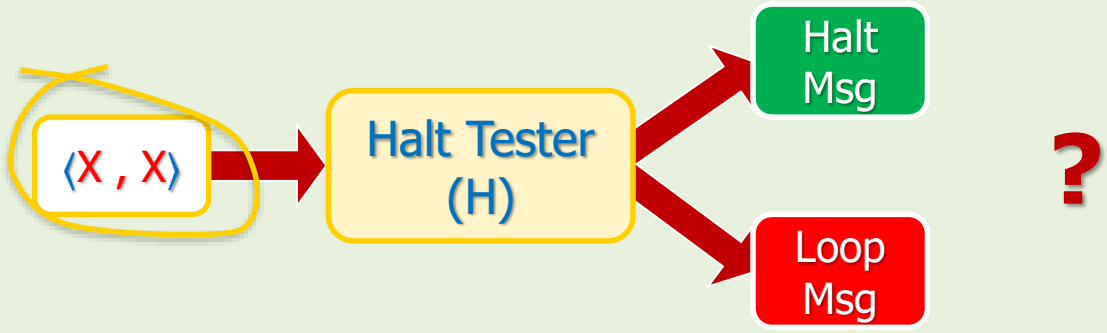
Constructing a Test-Case for H

- Use a copy of H, let's call it H', as a subroutine and construct another TM, called 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.



Halting Problem Proof

- H can decide about any TM and w.
- X, our test-case, is a TM and can be encoded.
- Now we input X, both as M and also its input w.
- It means, we input ⟨X , X⟩ to H!
- What is the answer of H when we input ⟨X , X⟩?



Halting Problem Proof

- As you see, there are two possible answers:

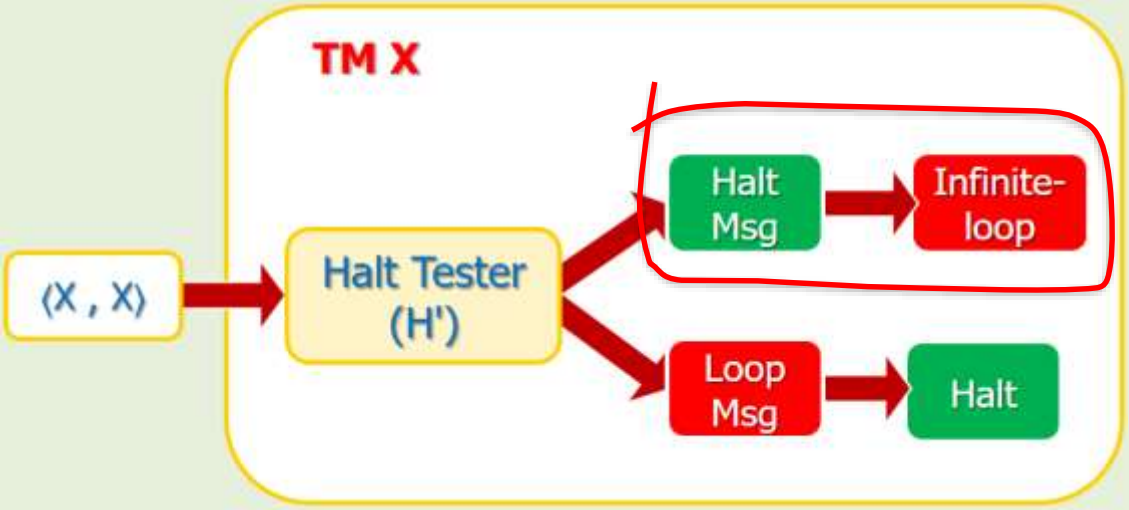
1. X halts on X.
2. X loops on X.

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

- Let's examine each case separately ...

Case 1: X halts on X

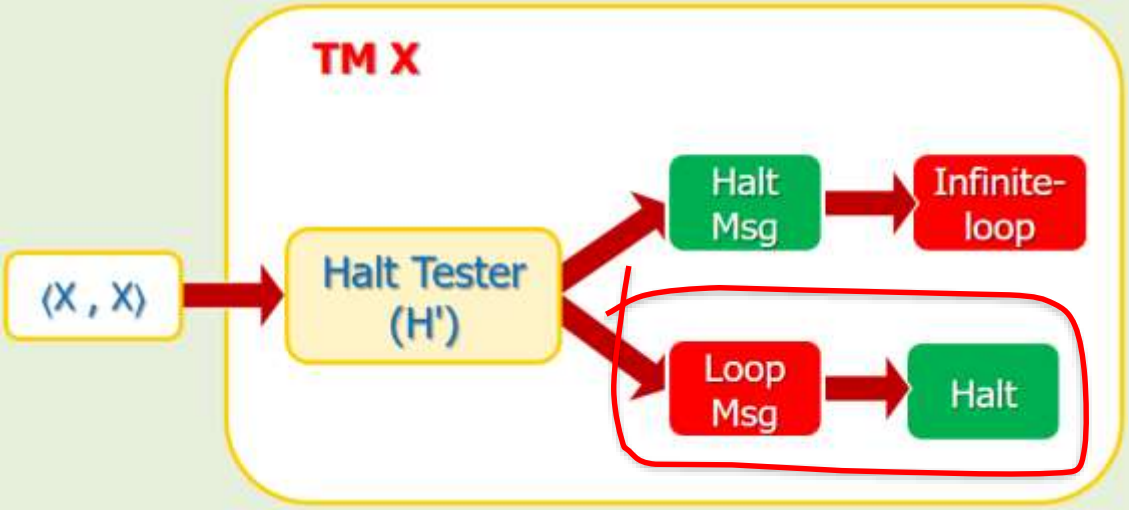
- To verify H's response, let's examine the X's behavior by inputting it ⟨X , X⟩.



- Recall that H' is a copy of H and should respond the same.
- 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!

Case 2: X Loops on X

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



- H' responds "Loop Msg", then it halts.
- Therefore, H was wrong again!

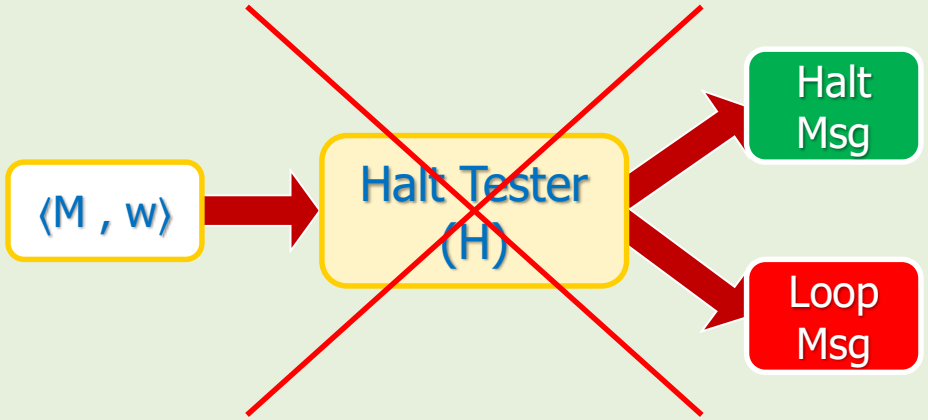
Halting Problem Proof

Conclusion

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