DECIDABILITY

DISCRETE MATHEMATICS AND THEORY 2
MARK FLORYAN

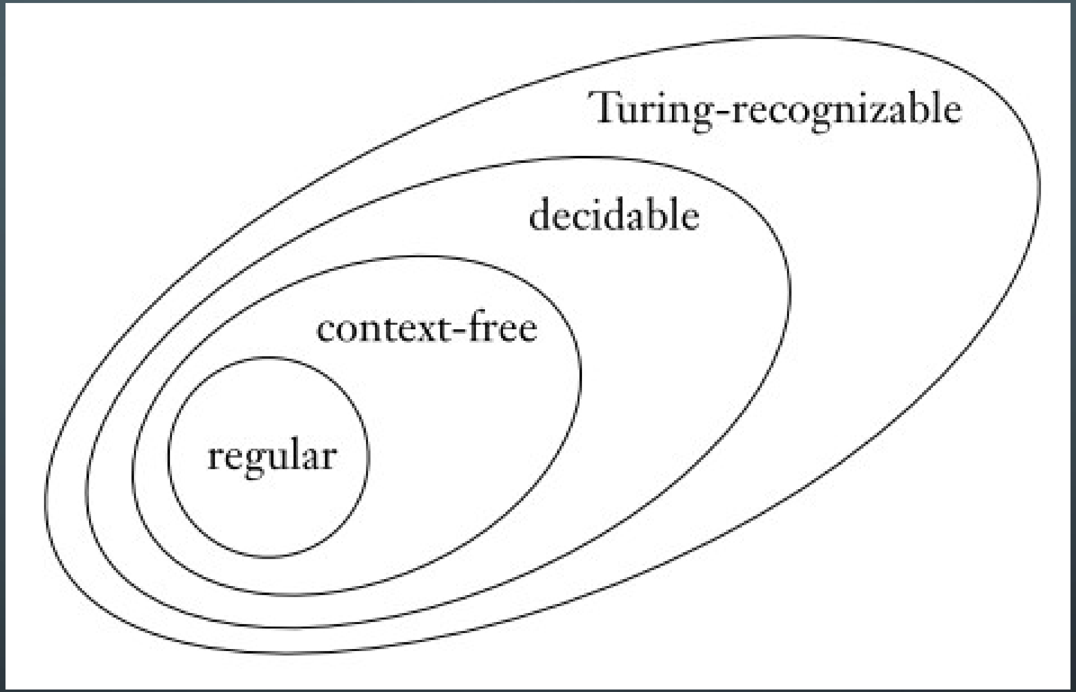
GOALS!

1. Let's revisit the concept of ***decidable languages***, and find some.

2. Let's find some examples of ***undecidable languages***, and even some examples of ***unrecognizable languages***.

3. Let's introduce the concept of reductions, which can expedite / simplify proofs that certain problems are ***undecidable*** or ***unrecognizable***.

THE BIG PICTURE!



PART 1: DECIDABLE LANGUAGES

DECIDABLE LANGUAGES

Recall that a ***decidable language*** is a language for which a Turing Machine exists that computes it and always halts.

Let's look at a few more decidable languages and eventually start discovering some undecidable languages.

DECIDABLE LANGUAGES

Example:

Can you describe a Turing Machine that decides this language?

DECIDABLE LANGUAGES

Example:

M = “On input <B,w>:

1. Simulate B on input w
2. If B ends in accept state, ***accept***. Otherwise, ***reject***.”

Let’s briefly discuss some of the implementation details involved in this.

w is finite, B is also guaranteed to halt. So the simulation must be possible and it must halt.

DECIDABLE LANGUAGES

Example:

How about this one? How would you design the machine this time?

DECIDABLE LANGUAGES

Example:

N = “On input <B,w>:

1. Convert NFA B into DFA C using procedure given previously.
2. Run Turing Machine M from previous slide on <C,w>
3. If M accepts, then ***accept***. Otherwise, ***reject***.”

MORE DECIDABLE LANGUAGES

All of the following languages are similarly decidable:

$$A_{REX} = \{ R, w \mid R \text{ is a reg. exp. that generates } w \}$$

Does a given expression generate this string?

$$E_{DFA} = \{ A \mid A \text{ is a } DFA \text{ and } L(A) = \varnothing \}$$

Is language of the DFA empty?

$$EQ_{DFA} = \{ A, B \mid A, B \text{ are } DFA \wedge L(A) = L(B) \}$$

Do two DFAs recognize the same language?

…and analogous languages for Context-Free Grammars (CFGs)

PART 2: UNDECIDABLE LANGUAGES

DO UNDECIDABLE LANGUAGES EXIST?

Are there problems that are unsolvable by computers (Turing Machines)?

DO UNDECIDABLE LANGUAGES EXIST?

Are there problems that are unsolvable by computers (Turing Machines)?

Many of these problems are recognizable, but not decidable.

Yes! In fact, many simple and common problems are undecidable.

This has profound philosophical implications in Computer Science. Some things are fundamental limitations that computers cannot overcome.

DO UNDECIDABLE LANGUAGES EXIST?

Theorem: The language is undecidable.

This language ***is Turing-Recognizable*** though. Here is how:

U = “On input <M,w>:

1. Simulate M on input w
2. If M ever accepts, then ***accept***.
3. If M ever reject, then ***reject***.

Note that if M loops forever, then so will U

DO UNDECIDABLE LANGUAGES EXIST?

Theorem: The language is undecidable.

Okay, let's prove it. Intuitively, what is the potential issue here?

This is one of the most famous proofs in Computer Science

DO UNDECIDABLE LANGUAGES EXIST?

Theorem: The language is undecidable.

Step 1: For the sake of contradiction, assume is decidable

$$H(\langle M, w\rangle) = \begin{cases} \mathit{accept} & \text{if } M \text{ accepts } w \\ \mathit{reject} & \text{if } M \text{ does not accept } w. \end{cases}$$

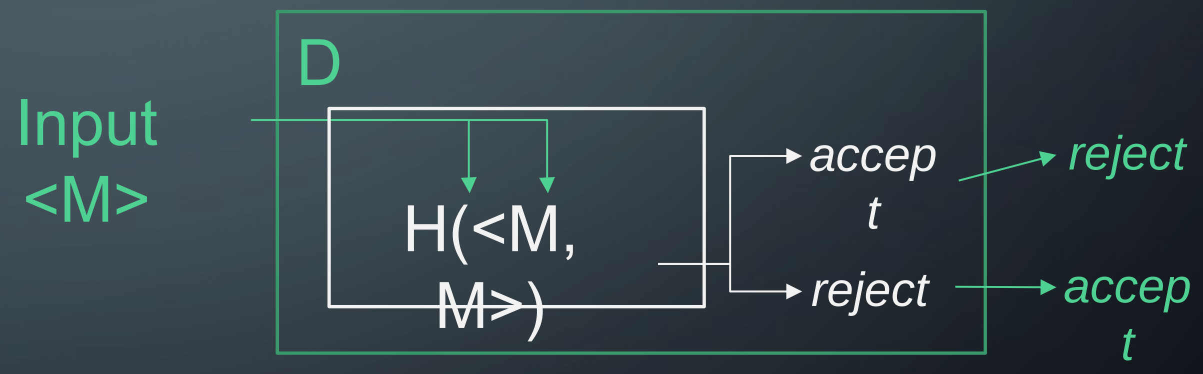
If is decidable, then there must exist a machine that decides it. Let's call that machine H

DO UNDECIDABLE LANGUAGES EXIST?

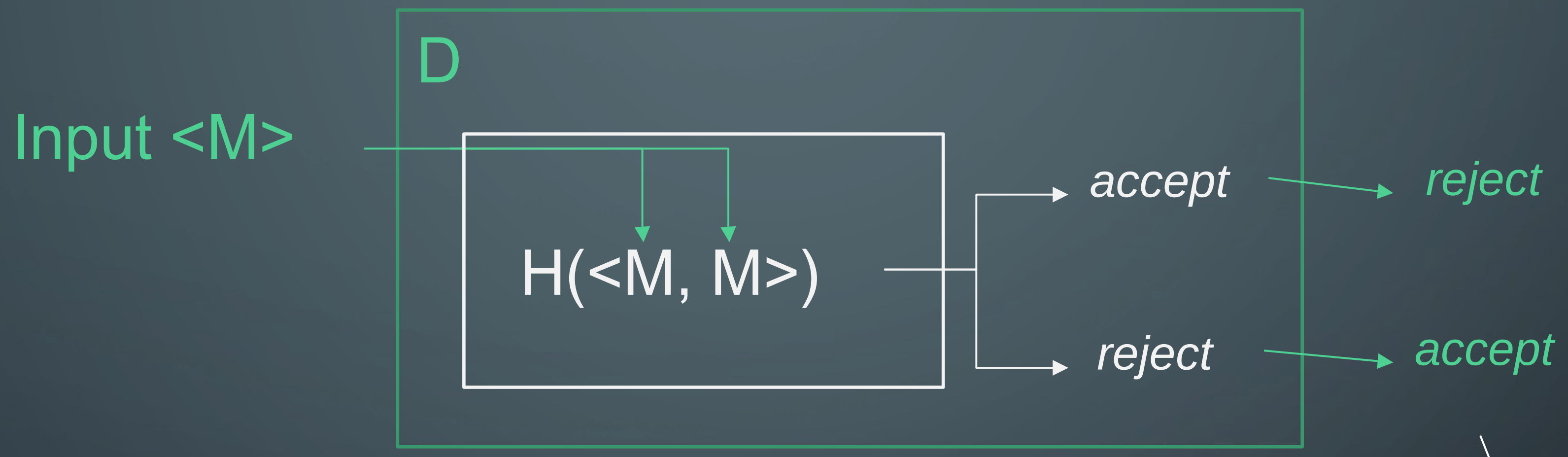
Theorem: The language is undecidable.

Step 2: Construct a new machine D that uses H as a subroutine.

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w. \end{cases}$$



SOME COMMENTS ON MACHINE D

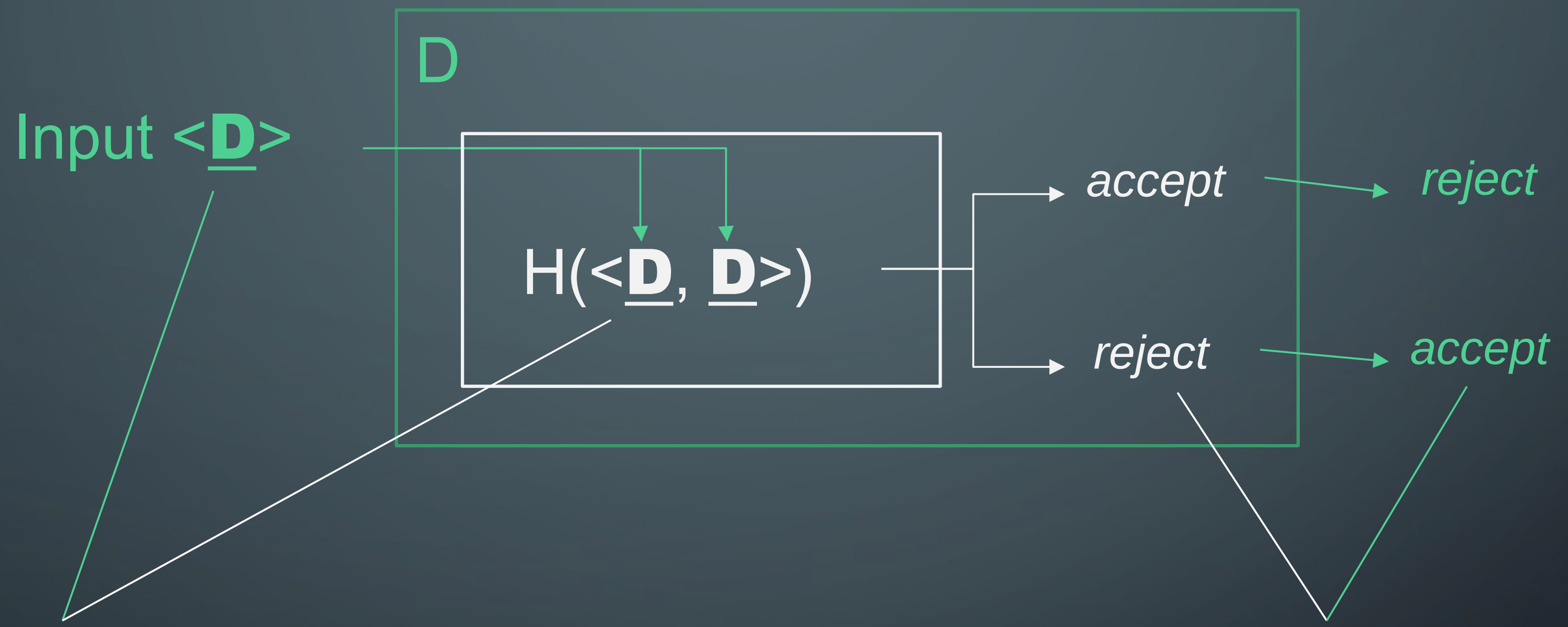


What does it mean to run a machine with itself as input? Is this even possible?

Notice that we flip the output here. This will be important for creating the contradiction

SOME COMMENTS ON MACHINE D

Step 3: *Run the machine D with itself (D) as input. What happens?*

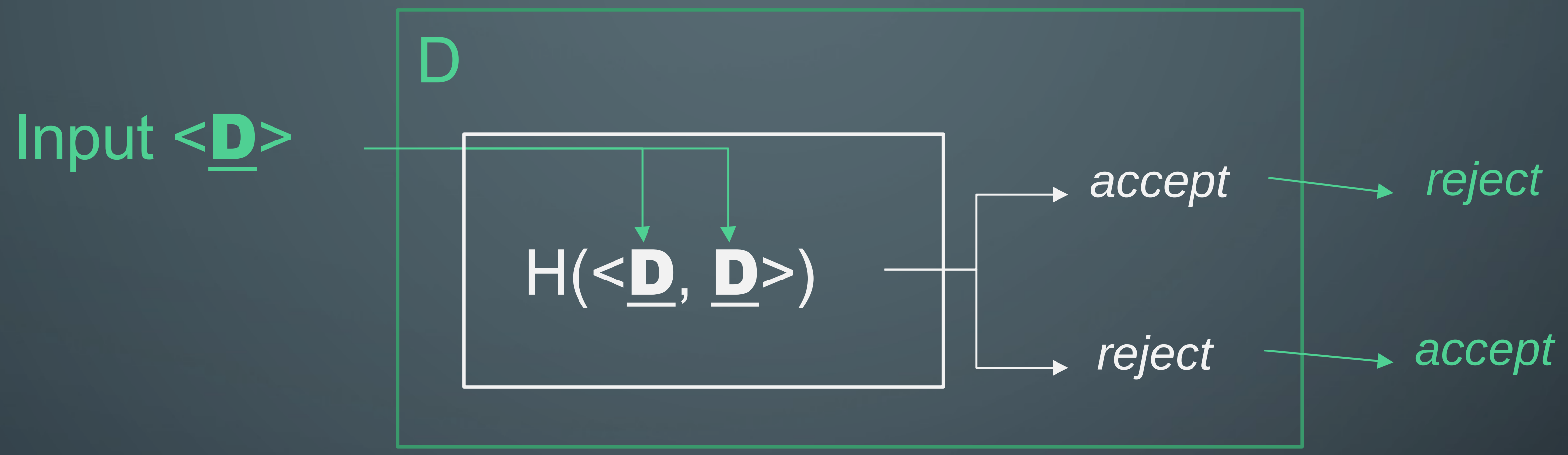


Notice that D is running with itself as input in two places, once overall (green square) and once simulated inside of H

Which means these outputs should match because they are the output of the exact same thing (D running on D as input)

SOME COMMENTS ON MACHINE D

Step 3: *Run the machine D with itself (D) as input. What happens?*



Q.E.D: *This is a contradiction because if H exists (is decidable), then there is at least one set of inputs where H produces the wrong answer (well, it cannot produce the right answer by definition).*

DO UNDECIDABLE LANGUAGES EXIST?

This is really a proof by diagonalization

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$			$\vdots$		$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		?	
$\vdots$			$\vdots$				$\ddots$

DO UNDECIDABLE LANGUAGES EXIST?

This is really a proof by diagonalization

Each entry is a machine's output when another machine's description is given as input

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$			$\vdots$		$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		*?*	
$\vdots$			$\vdots$				$\ddots$

But this entry has to be both accept and reject at the same time, leading to the contradiction

D is defined to be the machine that has the opposite output from the corresponding diagonal (see green outlines)

DO UNDECIDABLE LANGUAGES EXIST?

Theorem: The language is undecidable.

Thus it is proven, and there is at least one undecidable language

PART 3: NON-RECOGNIZABLE LANGUAGES?

NON-TURING RECOGNIZABILITY?

Is it possible to find languages that are NOT Turing recognizable?

Yes, but we will need to discuss the idea of the complement of a language first.

DEFINITION: COMPLEMENT OF A LANGUAGE

The ***complement*** of a language is the set of strings that do NOT belong to . In other words,

Some Examples:

$\mathscr{L}(A)$	$\overline{\mathscr{L}(A)}$
Strings containing less than ten 1's	Strings containing ten or more 1's
DFA <D> accepts string <w>	DFA <D> rejects string <w>
TM <M> halts on input <w>	TM <M> loops forever on input <w>

MORE ON COMPLEMENTS

TM <M> halts on input <w>

Some Turing Machine Executes on input / tape

Accept: *Input on tape is in language*

In language above, Accepts (Yes) is easy because if machine halts we are sure it is a Yes

Reject: *Input on tape is NOT in language*

Loop: *TM runs forever, never reaching accept or reject state*

However, distinguishing between Reject (No) and Looping Forever is difficult to ascertain. Is the machine just taking a long time?

MORE ON COMPLEMENTS

TM <M> loops forever on input <w>

Some Turing Machine Executes on input / tape

Accept: *Input in language*

Reject: *Input Not in language*

Loop: *TM runs forever*

Now, Rejecting (No) is easy. If we halt then we output Reject (No).

Now, distinguishing between Accept (Yes) and Looping Forever is hard. Is the machine just taking a long time and we should really reject or is it actually looping forever?

CO-TURING RECOGNIZABLE

A language is ***co-Turing recognizable*** iff the complement is Turing recognizable.

$\mathcal{L}(A)$

TM <M> loops forever on input <w>

→

$\overline{\mathcal{L}(A)}$

TM <M> halts on input <w>

→

Is recognizable as shown earlier

ANOTHER WAY TO DEFINE DECIDABILITY

<u>Theorem</u>: A language is decidable if and only if it is Turing-recognizable and it is co-Turing-recognizable

How to prove this?

<u>Direction 1</u>: If language is decidable → It is T-Rec. and Co-T-Rec.

<u>Direction 2</u>: If language is T-Rec. and Co-T-Rec. → It is decidable

PROVING THE THEOREM

Theorem: A language is decidable if and only if it is Turing-recognizable and it is co-Turing-recognizable

Direction 1: If language is decidable → It is T-Rec. and Co-T-Rec.

Assume language A is decidable → Thus, a machine that decides A (let's call it D) must exist

Recognizer

Input w → D

- If D accepts, ***accept***
- If D rejects, ***reject***

Co-Recognizer

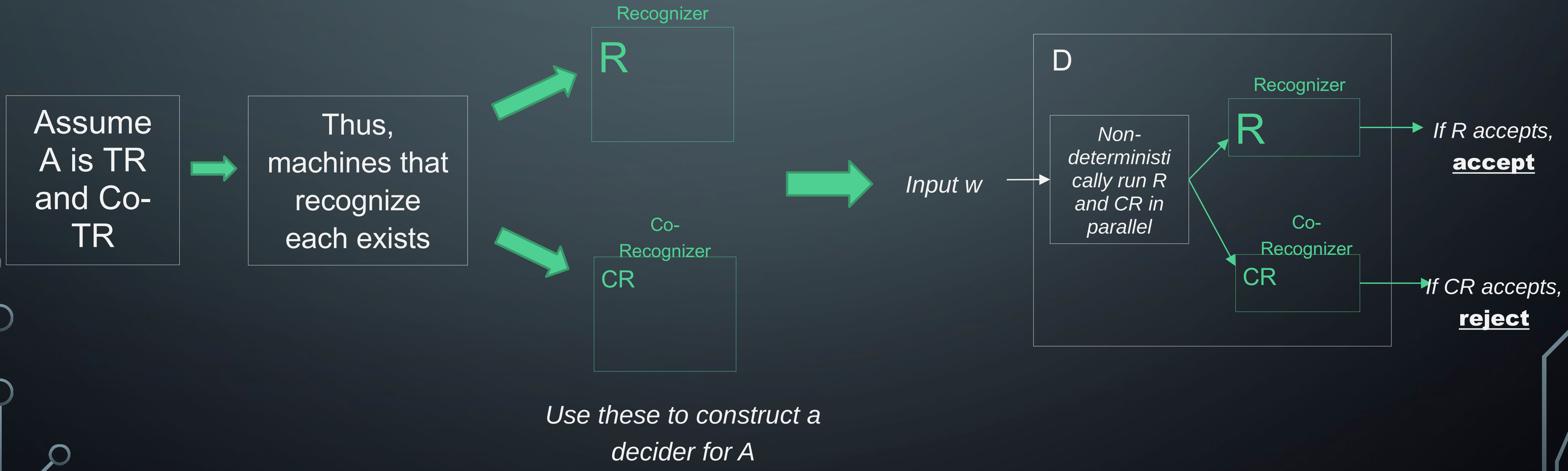
Input w → D

- If D accepts, ***reject***
- If D rejects, ***accept***

PROVING THE THEOREM

Theorem: A language is decidable if and only if it is Turing-recognizable and it is co-Turing-recognizable

Direction 2: If language is T-Rec. and Co-T-Rec. → It is decidable



UNRECOGNIZABILITY!

Finally ready to find an unrecognizable language

Theorem: is unrecognizable.

Recall that , thus:

UNRECOGNIZABILITY!

Finally ready to find an unrecognizable language

Theorem: is unrecognizable.

Can you prove it?

Assume for sake of contradiction that is recognizable

This means that (assumed) and (proven earlier) are both recognizable

Thus, is decidable by earlier theorem (both it and complement are recognizable)

Contradiction! is undecidable as proven earlier.

PART 4: INTRODUCTION TO REDUCIBILITY

WHAT IS REDUCIBILITY?

Reduction: A process through which problems are related to one another through comparison. This comparison establishes that one problem can be solved if the other is.

Problem: I wanted to visit Japan (during Covid)!
Wife's Obaachan was sick.

- Get Covid Visa Exception
- Book ticket(s)
- Travel on plane, etc.
- Go visit Obaachan

This one was ***hard***!

These are all ***easy***!

WHAT IS REDUCIBILITY?

Reduction: A process through which problems are related to one another through comparison. This comparison establishes that one problem can be solved if the other is.

Problem: I wanted to visit Japan (during Covid)! Wife's Obaachan was sick.

- Get Covid Visa Exception
- Book ticket(s)
- Travel on plane, etc.
- Go visit Obaachan

Reduces to →

Problem: Get Covid Visa Exception.

- Prove marriage to citizen
- Acquire invite from citizen
- Bring paperwork to embassy in DC
- …

Now this one is hard!

REDUCTION PROCESS

Reduction: A reduction exists between problems ***A*** and ***B*** if a solution to ***B*** can be used to develop a solution for ***A***.

Problem ***A***

Solve problem B

Do easy work

Do more easy work

…

Reduces to

Problem ***B***

Solve problem B

THE HALTING PROBLEM

The Halting Problem: Given a Turing machine, does it halt:

$$A_{TM}(M, w)$$

Accept if M accepts

Reject if M rejects

Reject if M loops forever

Reduces to

$$Halt_{TM}(M, w)$$

Does M halt on w?

If I can solve the problem in green, then I can solve both of these problems!!

THE HALTING PROBLEM

$A_{TM}(M, w)$

Accept if M accepts

Reject if M rejects

Reject if M loops forever

Reduces to

$Halt_{TM}(M, w)$

Does M halt on w?

Assume for the sake of contradiction, that is decidable. Thus, some machine R exists that decides it.

Machine M, on input w:

*- Invoke R on (M,w) to see if M halts. If not, __**reject**__.*

- Else simulate M on input w:

*- If M accepts, then __**accept**__.*

*- If M rejects, then __**reject**__.*

Then, this machine would decide Atm, but that contradicts our theorem that ATM is undecidable. Thus, halt is also undecidable

THE HALTING PROBLEM

Theorem: is undecidable

Proof was simplified by using a proof by contradiction via a valid reduction from

TM EMPTINESS TESTING

Emptiness Test: Can you use a similar reduction to show is undecidable?

In other words, test whether the given machine never accepts

TM EMPTINESS TESTING

Emptiness Test: Can you use a similar reduction to show is undecidable?

Step 0: Assume, for sake of contradiction, a machine R decides

Step 1: Modify M:

= "on input x:
, <u>reject</u>
otherwise, run M on w, <u>accept</u> iff M does"

**Notice that w is hardcoded into description of*

Why is this helpful?

TM EMPTINESS TESTING

Emptiness Test: Can you use a similar reduction to show is undecidable?

Step 0: Assume, for sake of contradiction, a machine R decides

Step 1: Modify M:

= "on input x:
, <u>reject</u>
otherwise, run M on w, <u>accept</u> iff M does"

Step 2: Solve Atm

= "on input (M,w):
Construct as described
Run R on input
Flip the output of R"

*Key Idea: M1 can **only** accept w*

So, testing emptiness on M1 = testing acceptance of M on w

TM EMPTINESS TESTING

Emptiness Test: Can you use a similar reduction to show is undecidable?

Thus, is undecidable via reduction from !!

ONE MORE EXAMPLE

Regular?: Prove this language is undecidable through reduction.

If we can decide this, can we use it to decide ?

Similar idea!
Construct a machine that recognizes non-regular languages

ONE MORE EXAMPLE

Regular?: Prove this language is undecidable through reduction.

Step 0: For sake of contradiction, assume is decidable, thus a machine R exists that decides it.

Similar idea!
Construct a machine that recognizes non-regular languages

Step 1: Construct :

= "on input x:
if x has form , accept
else, run M on w and accept iff M accepts"

ONE MORE EXAMPLE

Regular?: Prove this language is undecidable through reduction.

Step 0: For sake of contradiction, assume is decidable, thus a machine R exists that decides it.

Step 1: Construct :

= "on input x:
if x has form , accept
else, run M on w and accept iff M accepts"

Observe:

If M accepts w, then accepts

If M accepts w, then accepts

ONE MORE EXAMPLE

Regular?: Prove this language is undecidable through reduction.

Step 0: For sake of contradiction, assume is decidable, thus a machine R exists that decides it.

Step 2: Recognize

S = on input (M,w):
Construct as described earlier
Run R on
Accept IFF R accepts

Why does this work? will be regular if M accepts w?