Theory of Computation//The Halting Problem

Arjun Chandrasekhar

Unsolvable Problems

- Can we train the java compiler to detect your infinite loops *before you run your code*?
- Can we create the perfect virus detection software?
- Can we get computers to tell us which mathematical conjectures are true/false?

Programs taking other programs as input

- Can pass one program description as input to another program
- Example: Let even.java be a program that takes an a string w as a command line argument
  - If w has even length, output "ACCEPT"
  - Otherwise, output "REJECT"
- We could pass the source code of even.java as the input to even.java
  - Pass the source code as one long string
  - What will this do?
  - This would check if even.java contains an even number of characters in its source code

Programs taking other programs as input

- Let strange.java be a program that takes the name of a java source code file program.java as input and does the following:
  1. Make one long string out of the source code of program.java
  2. Pass this string to even.java
  3. If even.java outputs ACCEPT, strange.java outputs REJECT
  4. If even.java outputs REJECT, strange.java outputs ACCEPT

What happens if we pass strange.java as the input to strange.java?

Programs taking other programs as input

What happens if we pass strange.java as the input to strange.java?

1. Create a string s out of the source code of strange.java
2. Pass s as the argument to even.java
3. If even.java outputs ACCEPT, strange.java outputs REJECT
4. If even.java outputs REJECT, strange.java outputs ACCEPT

strange.java checks if its own source code has an even length

Undecidable Languages

- We are now ready to show that certain languages are undecidable
- No computer program will EVER solve these problems
- We will make use of diagonalization, as well as machines that take other machines as input
  - "If we could recognize this language, we could construct a that machine contradicts every machine in the world - including itself"

The Halting Problem

Raise your hand if you have ever written an infinite loop

- Wouldn't it be nice if the compiler could detect these ahead of time?

The Halting Problem

The Halting Problem

Raise your hand if you have ever written an infinite loop

- Wouldn't it be nice if the compiler could detect these ahead of time?

Theorem: It is impossible to write a compiler that can detect infinite loops with 100% accuracy

The Halting Problem - Proof Idea

Theorem: It is impossible to write a compiler that can detect infinite loops with 100% accuracy

- **Proof idea:** if we could do this, we could write a program that literally contradicts itself
- We will write a program that runs this compiler on itself and then does the opposite of what it is "supposed" to do
- Our program will "fool" the compiler, thus proving the compiler doesn't actually perform as advertised

The Halting Problem - Starting Assumption

Assume for sake of contradiction we have a program called halt.java

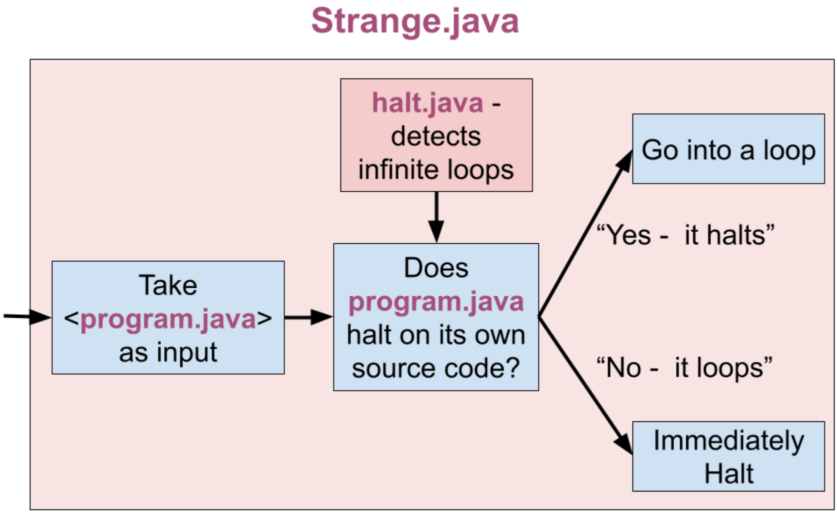
- halt.java takes two command line arguments: program.java and w
- halt.java prints ACCEPT if program.java halts on input w
- halt.java prints REJECT if program.java goes into an infinite loop on input w

The Halting Problem - Strange Program

Let's create a program called strange.java

1. strange.java takes one command line argument: program.java
2. strange.java creates a string w out of the source code of program.java
3. strange.java runs halt.java and passes $\langle program.java, w \rangle$ as command line arguments
4. If halt.java prints ACCEPT then strange.java goes into an infinite loop
5. If halt.java prints REJECT then strange.java immediately halts

The Halting Problem - Strange Program



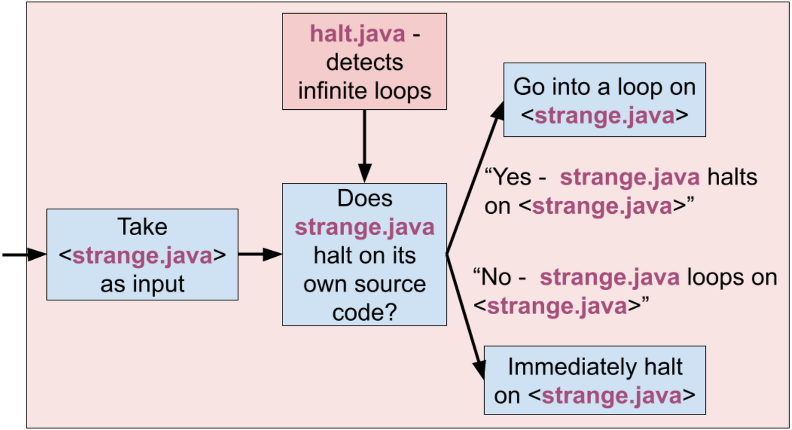
The Halting Program - Counterexample

What does strange.java do if it receives its own source code strange.java as input?

1. strange.java creates a string w out of the source code of strange.java
2. strange.java passes $\langle \textit{strange.java}, w \rangle$ to halt.java
3. If halt.java prints ACCEPT, strange.java goes into an infinite loop
4. If halt.java prints REJECT, strange.java halts

The Halting Program - Counterexample

Let's feed **Strange.java** its own source code



The Halting Problem - Contradiction

- If halt.java says strange.java will halt on its own source code, strange.java goes into an infinite loop
- If halt.java says strange.java will loop on its own source code, strange.java will immediately halt
- THIS IS A CONTRADICTION!!!
- We conclude that halt.java is not detecting infinite loops correctly.

The Halting Problem - Follow Up

Some notes:

- The point of this argument is not that we *want* to write strange.java
- The point is that it shouldn't even be *possible* to write a program like strange.java
- It's only possible to create strange.java if we assume that halt.java exists
- We conclude that halt.java doesn't exist, because paradoxical programs don't exist

HALT is Undecidable

Let's prove the same theorem using Turing machines

$$\mathrm{HALT} = \{\langle M, w\rangle | \text{ M halts on w}\}$$

- We receive two input arguments
  - The source code/description of machine M
  - Some string w
- We want to design a machine that can check if M will halt on w

Theorem: HALT is undecidable

HALT is Undecidable - Proof Idea

Proof idea: construct a machine that is self-contradictory

- AFSOC H is a machine that decides HALT
- We will construct a machine S that asks H what it is supposed to do and does the opposite
- By assuming that H exists, we can create a machine S that should not exist

Turing Machine Descriptions

- Let M be a Turing Machine
- $\langle M \rangle$ is a string that refers to the *description* of M
- Think of $\langle M \rangle$ as a source code file and M as an actual executable that can be run

HALT is Undecidable - Initial Assumption

AFSOC H decides HALT

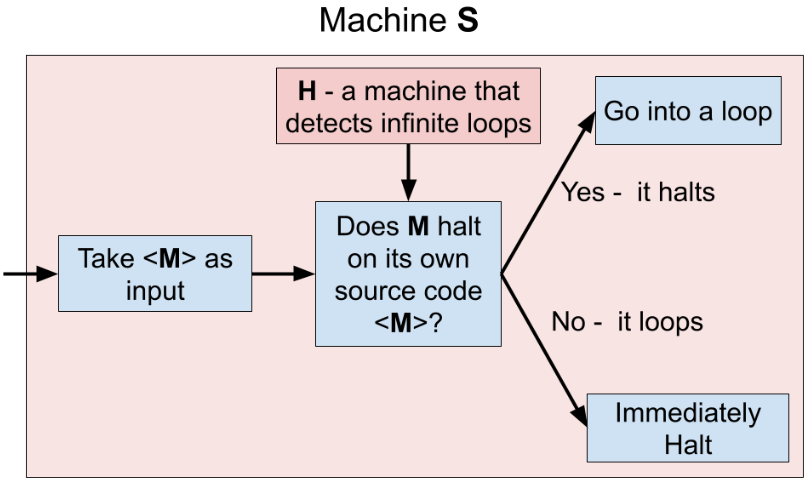
- H takes $\langle M, w \rangle$ as input
- H accepts if M halts on w
- H rejects if M loops on w

HALT is Undecidable - Strange Machine

Construct a machine S that does the following:

1. S takes a machine description $\langle M \rangle$
2. Run H on $\langle M, \langle M \rangle \rangle$
   - "Does M halt if it gets its own source code as input?"
3. S then "does the opposite" of what H says
   3.1 If H accepts $\langle M, \langle M \rangle \rangle$, S goes into a loop
   3.2 If H rejects $\langle M, \langle M \rangle \rangle$, then S immediately halts

HALT is Undecidable - Strange Machine



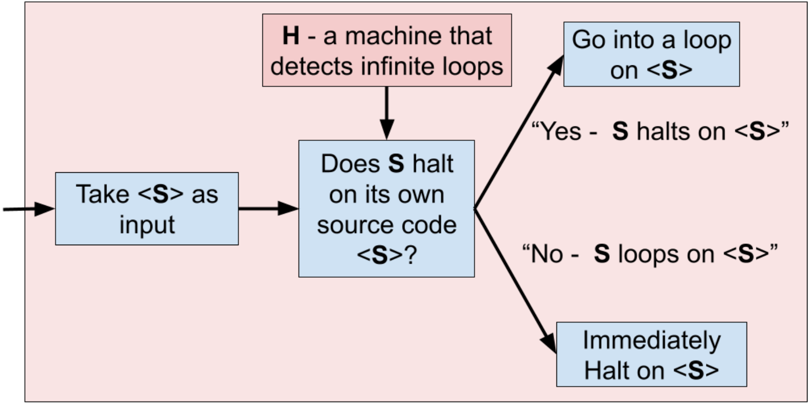
HALT is Undecidable - Contradiction

What happens if S receives $\langle S \rangle$ as input?

1. S runs H on $\langle S, \langle S \rangle \rangle$
2. If H accepts $\langle S, \langle S \rangle \rangle$, S loops on $\langle S \rangle$
3. If H rejects $\langle S, \langle S \rangle \rangle$, S halts and accepts $\langle S \rangle$

HALT is Undecidable - Contradiction

Let's feed **S** its own source code



HALT is Undecidable - Contradiction

What happens if S receives $\langle S \rangle$ as input?

1. S runs H on $\langle S, \langle S \rangle \rangle$
2. If H accepts $\langle S, \langle S \rangle \rangle$, S loops on $\langle S \rangle$
   - If S is supposed to halt on its own description, it loops!
3. If H rejects $\langle S, \langle S \rangle \rangle$, S halts and accepts $\langle S \rangle$
   - If S is supposed to loop on its own description, it halts!

There is no way that H is actually deciding HALT correctly!

Diagonalizing HALT

- We can interpret the preceding proof as a form of diagonalization
- We assumed that we could determine what every program does on every possible input
- We constructed a machine S that contradicted every program in the universe
  - But this means that S contradicts itself
- Thus we reject our original assumption

Diagonalizing HALT

	$<M_1>$	$<M_2>$	$<M_3>$	...	$<S>$
M_1	**HALT**	LOOP	LOOP	...	HALT
M_2	LOOP	**LOOP**	LOOP	...	HALT
M_3	HALT	LOOP	**HALT**	...	LOOP
...	...	...	...	...	...
S	LOOP	HALT	LOOP	...	**???**

Box (i, j):
"Does machine M_i halt or loop on source code $<M_j>$"?

The existence of **H**, a decider for **HALT**, allows us to fill in this table (and then construct the paradoxical machine S)

Construct **S** by taking the "opposite" of the **diagonals** until we reach a **contradiction**

HALT is Recognizable

HALT is not decidable. Is it at least recognizable?

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$

Let's design a machine H to recognize HALT

- If M halts on w then H needs to accept $\langle M, w\rangle$
- If M loops on w then H should reject or possibly loop on $\langle M, w\rangle$

HALT is Recognizable

Let's design a machine H to recognize

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$

H does the following on input $\langle M, w\rangle$:

1. Run M on w
   1.1 If M ever halts, then accept $\langle M, w\rangle$
   1.2 If M loops forever then H will loop forever

- If M does indeed halt on w then eventually H will accept $\langle M, w\rangle$
- If M loops forever on w, H will do the same, so it will not accept $\langle M, w\rangle$ (which is sufficient)

co-Recognizable Languages

A language L is **co-Turing Recognizable** if it's complement $\overline{L}$ is recognizable

- We can construct a machine $\overline{M}$ that recognizes $\overline{L}$
- If $w \in \overline{L}$ (i.e. $w \notin L$) then $\overline{M}$ will halt and accept
- If $w \notin \overline{L}$ (i.e. $w \in L$) then $\overline{M}$ will reject or loop forever

co-Recognizable Languages

A language L is **co-Turing Recognizable** if it's complement $\overline{L}$ is recognizable

- We sometimes say L is **co-recognizable**
- We can also say L is **co-Recursively Enumerable** or **co-RE**
- **Note:** In prior lectures we used L^c to denote the complement. For these topics, the convention is to use $\overline{L}$ to denote the complement

co-Recognizable Languages

Theorem: A language is decidable if and only if L is both recognizable and co-recognizable

1. ($\Rightarrow$) If a language is decidable it is both recognizable and co-recognizable
2. ($\Leftarrow$) If a language is both recognizable and co-recognizable, it is decidable

co-Recognizable Languages

($\Rightarrow$) If L is decidable then it is recognizable

- Let M be the machine that decides L
- Then M also recognizes L!
  - M always halts
  - If $w \in L$ then M will halt and accept
  - If $w \notin L$, M will not accept (in fact, it will halt and reject)

co-Recognizable Languages

($\Rightarrow$) If L is decidable then it is co-recognizable

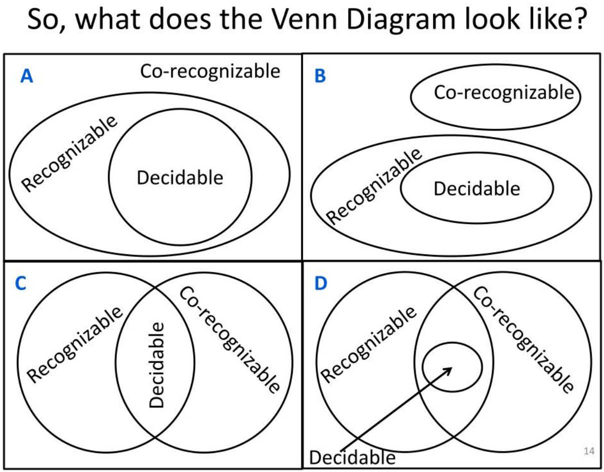
- Let M be the machine that decides L
- To recognize $\overline{L}$ we create a machine $\overline{M}$ that runs M and does the opposite
  - M always halts, so $\overline{M}$ always halts
  - If $w \in \overline{L}$ then M will halt and reject, so $\overline{M}$ will halt and accept
  - If $w \notin \overline{L}$, then $w \in L$. So M will halt and accept, and $\overline{M}$ will halt and reject

co-Recognizable Languages

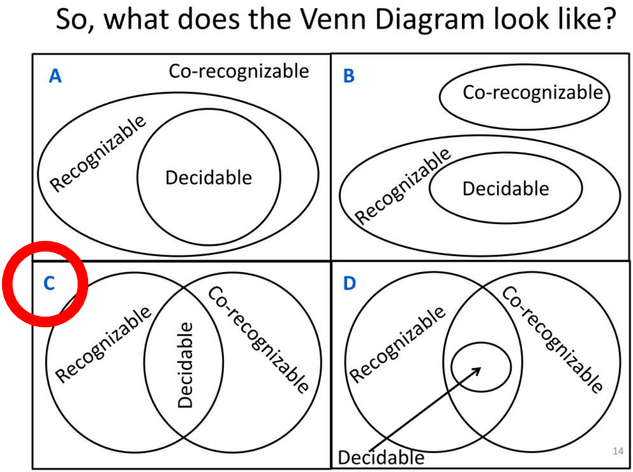
($\Leftarrow$) If L is both recognizable and co-recognizable, then L is decidable

- Let M recognize L and $\overline{M}$ recognize $\overline{L}$
- Construct a machine D to decide L
- D does the following on input w
  1. Run M and $\overline{M}$ in parallel
  2. If M accepts, accept
  3. If $\overline{M}$ accepts, reject
- Exactly one of the two machines has to eventually accept, so D always halts

Decidable vs. (co-)Recognizable Languages



Decidable vs. (co-)Recognizable Languages



The language $\overline{\text{HALT}}$

Consider the following language

$$\overline{\text{HALT}} = \{\langle M, w\rangle | M \text{ loops on } w\}$$

- Pronounced "co-HALT"
- If M loops on w we accept $\langle M, w\rangle$
- If M halts on w we reject $\langle M, w\rangle$
- $\overline{\text{HALT}}$ is co-recognizable because its complement HALT is recognizable

Unrecognizability of $\overline{\mathrm{HALT}}$

Theorem: $\overline{\mathrm{HALT}}$ is not Turing-recognizable

- AFSOC $\overline{\mathrm{HALT}}$ is recognizable
- Then HALT is co-recognizable
- We know that HALT is also recognizable
- Then HALT would decidable, which is a contradiction!
- We conclude that $\overline{\mathrm{HALT}}$ is unrecognizable