Decidability - Overview

- Every question about regular languages is decidable.
- Some questions about CFLs are decidable, but some are not.
- The "Halting Problem" is not decidable.
- Some languages are not Turing recognizable.
- Many questions about Turing machines are not decidable, and some are not even Turing recognizable.
The "Halting Problem"

Given a program... 
Will it halt?

Given a Turing Machine, will it halt when run on some particular given input string?

Given some program written in [Java/C/any other language], will it ever get into an infinite loop? Or will it always terminate?
Answer

In general, we can't always know!

The best we can do is run the program and see whether it halts... but...

For many programs, we can prove "It will always halt." [or "It may sometimes loop."]

But for programs in general, the question is **undecidable**.
Problem:
Given a D.F.A. and a string, will the DFA accept?

Is this problem decidable?

Languages are decidable.
We must express the problem in terms of languages!

"x is a member of the language iff
the answer to the problem is yes."
**Theorem**

\[
A_{DFA} = \{ \langle B, w \rangle \mid \text{B is a DFA that accepts string } w \} \\
\text{... is Decidable.}
\]

**Possible Confusion**

- **String** \(w\)
- **Language** \(DFA = \text{Regular Language}\)

- **String** \(\langle B, w \rangle\)
- **Language** \(A_{DFA}\)

Not regular, not CFG, but it is Decidable
PROOF THAT

\[ A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts string } w \} \]

IS DECIDABLE.

Provide a TM that decides it.

The TM is given as input \( \langle B, w \rangle \) a DFA and a string \( w \).

The TM checks to make sure B is a valid representation.

The TM then simulates B on \( w \).

If B reaches a final state at the end of \( w \), then the TM will accept.

Otherwise, the TM will reject.

This TM will always halt.

(We could prove that the TM will always halt, if necessary.)
Consider the question of whether a nondeterministic finite state automaton accepts a given string.

The language...

\[ A_{\text{NFA}} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts string } w \} \]

...is decidable.

Proof

Construct a TM that takes as input the representation of an NFA \( B \) and a candidate string \( w \).

**Approach 1:** Simulate the NFA on \( w \).

**Approach 2:** Convert the NFA to a DFA.

This algorithm was described in Chapter 1.

We could program a TM to do it.

Simulate the DFA on \( w \).

We could program a TM to do it.

[We did that for \( A_{\text{DFA}} \).

Accept if the simulation accepts, otherwise reject.
Given a regular expression $R$ and a string $w$, can we decide whether $R$ generates $w$?

The language...

$A_{\text{rex}} = \{ <R,w> | R \text{ is a regular expression that generates string } w \}$

...is decidable.

How do we know?

We can build a TM / we can write a program that, given a reg. expression $R$ and string $w$ as input, will determine whether $<R,w>$ is in $A_{\text{rex}}$.

And that algorithm / TM will always halt.

Halting?

Often this is self-evident. Sometimes we ought to prove it carefully. Many programs can be proven to always halt. But this does not mean the halting problem is decidable!
\[ A_{\text{REX}} = \left\{ \langle R, w \rangle \mid R \text{ is a regular expression that generates } w \right\} \]

**ALGORITHM/ TM**

**STEP 1:** Convert \( R \) into a NFA, \( B' \).

*(RECALL THE ALGORITHM)*

\[
\begin{align*}
  \text{X} & \mid \text{Y} \implies \quad \text{Diagram of NFA for X|Y} \\
  \text{X} \cdot \text{Y} & \implies \quad \ldots \\
  \text{X}^* & \implies \quad \text{Diagram of NFA for X*}
\end{align*}
\]

**STEP 2:** Construct the string \( \langle B', w \rangle \)

**STEP 3:** Use the TM from the last theorem to decide whether this is in

\[ A_{\text{NFA}} = \left\{ \langle B, w \rangle \mid B \text{ is a NFA that accepts } w \right\} \]
Acceptance Testing

Is string $w$ in the language?

$A_{DFA} = \{ \langle B, w \rangle \mid \ldots \}$

Emptiness Testing

Is the language empty?

$L = \emptyset$?

$E_{DFA} = \{ \langle B \rangle \mid \ldots \}$

Equality Testing

Are two languages the same?

$\text{EQ}_{DFA} = \{ \langle A, B \rangle \mid \ldots \}$
The language...

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

... is decidable.

Algorithm/TM to decide it:

Given a DFA "A", can we go from the initial state to a final state?

If so, then the DFA could generate some string.

⇒ Is any final state reachable from the initial state?

A graph problem:

- Mark the initial state
- Repeat until no new states get marked...
  - Mark any state where there is a transition to it from a marked state
- Check to see if any final state got marked.
THEOREM

THE LANGUAGE

\[ EQ_{\text{DFA}} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \} \]

IS DECIDABLE.

PROOF

Let \( C \) be the "SYMMETRIC DIFFERENCE" between \( A \) and \( B \).

"Anything in \( A \) or \( B \) but not both."

\[ C = (A \cap \overline{B}) \cup (\overline{A} \cap B) \]

NOTE: IF \( A = B \) THEN THE SYMMETRIC DIFFERENCE WILL BE \( \emptyset \).
Given...

\[ A = \text{DFA to accept } L(A) \]
\[ B = \text{DFA to accept } L(B) \]

\[ \ldots \text{we know how to combine DFA's.} \]

\[ \overline{L(A)} \]
\[ L(A) \cup L(B) \]
\[ L(A) \cap L(B) \]

Build DFA \( C \) to accept the symmetric difference.

Use the TM from previous theorem \[ E_{\text{DFA}} = \text{empty language} \] to test.

Proof

Construct a TM

Input: \( <A, B> \) (2 DFA's)

Construct a DFA \( C \) to accept

\[ L(C) = \overline{(L(A) \cap L(B))} \cup \overline{(L(A) \cap L(B))} \]

Use the previous TM to test whether the language that \( C \) accepts is empty. Accept if so; reject otherwise.
CONTEXT-FREE LANGUAGES

Given a CFG, will it generate a given string, \( w \)? *Decidable!*

Given a CFG, is the language it generates empty? *Decidable!*

Given two CFGs, do they accept the same language? *Not Decidable!*

\[ \begin{align*}
\{ & \text{Is a CFG ambiguous?} \\
& \text{Do two CFGs have any strings they generate in common?} \\
& \text{Is the complement of a CFG also a CFG?} \\
\} \quad \text{Not Decidable!} 
\]
THEOREM

The Language

\[ A_{CFG} = \{ (G, w) \mid G \text{ is a CFG that generates string } w \} \]

is decidable.

In other words:

"Given a CFG and a string, we can write a program that will always halt and will tell us yes/no whether the grammar generates the string."

All grammars are "parsable."

Some kinds of grammars (e.g., LL(k) or LR(k) grammars) can be parsed very efficiently.

But in general, the parser may take \( O(N^3) \) time.

(Where \( N \) is the length of the string.)
**PROOF**

**IDEA #1:**

Enumerate all leftmost derivations. For each, test to see if it generates \( z \).

**Problem:** What if \( z \) is not in the language? 
\[ \Rightarrow \text{May not halt!} \]
Proof

Inputs: \( G = \text{a CFG.} \)
\( W = \text{a STRING.} \)

Step 1: Convert \( G \) into Chomsky Normal Form.

Derivations using CNF Grammars

At each step, the length grows by exactly 1.

\( S \rightarrow SS \)
\( S \rightarrow a \)

\( S \rightarrow SS \rightarrow SSS \rightarrow SSSS \rightarrow SSSSS \rightarrow SSSSSS \)

... Plus 1 additional step for each terminal symbol:

\( aSSSS \rightarrow aaaSSS \rightarrow aaaaSS \rightarrow aaaaaS \)

\[ \vdash \text{Every derivation has exactly } 2N-1 \text{ steps.} \]
Step 2:
Let N be the length of w.
List all derivations of length 2N-1.
(There are only finitely many.)
Check each derivation to see if it generates w.
If any derivation generates w,
then accept.
else reject.
THEOREM

The language

\[ E_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \} \]

is decidable.

To prove \( E_{\text{CFG}} \) is decidable...

To prove some problem is decidable...

Give an algorithm (i.e., a T.M.)

Verify the algorithm is correct.

* Always halts.
* Gives the right answer.
• Consider this grammar.
• Which non-terminals can generate a string of terminals?

\[
\begin{align*}
S & \rightarrow ABCD \\
A & \rightarrow BCA \\
A & \rightarrow xyz \\
B & \rightarrow CA \\
B & \rightarrow AB \\
B & \rightarrow BBB^w \\
C & \rightarrow CB \\
C & \rightarrow w^w \\
D & \rightarrow DD \\
D & \rightarrow BD \\
D & \rightarrow DC
\end{align*}
\]
**Algorithm**

**Input:** A CFG "G".

A "Marking" Algorithm:

Mark all Terminal Symbols

Repeat

Look for a Rule:

\[ A \rightarrow XXXX \]

Where all symbols on right side have been marked.

\[ B \rightarrow CA \]

Mark the nonterminal on the left side.

Until nothing more can be marked.

If the start symbol is marked not

Then accept \( (L(G) = \emptyset) \)

Else reject \( (L(G) \neq \emptyset) \)

Does it terminate?

Is it correct?
Theorem

The language

\[ A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a Turing Machine and } M \text{ accepts } w \} \]

is Turing recognizable.

Given the description of a Turing Machine and some input, can we determine whether the machine accepts it?

Sure!

Just simulate/run the TM on the input.

\[ M \text{ accepts } w \]

Our algorithm will halt & accept

\[ M \text{ rejects } w \]

Our algorithm will halt & reject.

\[ M \text{ loops on } w \]

Our algorithm will not halt!

So it is not a decider!
THE "UNIVERSAL TURING MACHINE"

**Input:** $M =$ the description of some T.M.
$w =$ an input string for $M$.

**Action:**
- Simulate $M$.
- Behave just like $M$ would.
  (May ACCEPT, REJECT, or LOOP)

THE UNIVERSAL TURING MACHINE

is a **recognizer** (but

not a **DECIDER**) for

$$A_{TM} = \{ <M,w> \mid M \text{ is a TM and } M \text{ accepts } w \}$$
**DEFINITIONS**

**"ONE-TO-ONE"**

If $a \neq b$ then $f(a) \neq f(b)$

**NO!**

**"ON-TO"**

Every element in $B$ is "hit."

**NO!**

**"CORRESPONDENCE"**

One-to-one and onto
INFINITY: COUNTABLE AND UNCOUNTABLE

GEORG CANTOR: "Two sets have the same size iff there exists a correspondence between them."

A set is "COUNTABLE" iff

It has a finite size, or

There is a correspondence with $\mathbb{N}$

$$\mathbb{N} = \{1, 2, 3, \ldots\}$$

ODD NUMBERS = $\{1, 3, 5, \ldots\}$

COUNTABLY INFINITE

ALSO A SUBSET OF $\mathbb{N}$

RATIONAL NUMBERS = $\{m/n \mid m \text{ and } n \in \mathbb{N}\}$

COUNTABLY INFINITE

IRRATIONAL NUMBERS/REAL NUMBERS UNCOUNTABLY INFINITE
The set of rational numbers is countably infinite.

Proof: Find a way to list them. Must include all of them.

Rationals: \[ \left\{ \frac{m}{n} \mid m \text{ and } n \in \mathbb{N} \right\} \]

\[
\begin{array}{ccccccccc}
\frac{1}{7} & \frac{4}{3} & \frac{22}{29} & \frac{1}{2} & \frac{17}{3} & \frac{4}{2} & \ldots \\
1 & 2 & 3 & 4 & 5 & 6 & \ldots \\
\end{array}
\]
The set of irrational numbers is **UNCOUNTABLY INFINITE**!

\[
\begin{align*}
\pi &= 3.14159265 \ldots \\
\sqrt{2} &= 1.4142135 \ldots \\
e &= 2.718281828 \ldots \\
&= 5.67932043 \ldots \\
\frac{1}{3} &= 0.3333333333 \ldots < 0.333333, 5.71, 29 \ldots \\
&= 0.3333334, 0, 0, \ldots
\end{align*}
\]

**Proof:** Assume it is **COUNTABLY INFINITE**.

\[
\begin{array}{cccccccc}
1 & 3 & 1 & 4 & 1 & 5 & 9 & \ldots \\
2 & 1 & 4 & 1 & 4 & 2 & 1 & \ldots \\
3 & 2 & 7 & 1 & 8 & 2 & 8 & \ldots \\
4 & .5 & 6 & 7 & 9 & 3 & 2 & \ldots \\
5 & .7 & 4 & 2 & 5 & 3 & 1 & \ldots \\
6 & .3 & 9 & 2 & 4 & 5 & 0 & \ldots \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
.4 & 5 & 2 & 8 & 4 & 1 & \ldots \\
\end{array}
\]

This Number is NOT in the table!
WE CAN ENUMERATE THE SET OF ALL TURING MACHINES.

**Approach #1**

- Every Turing machine can be encoded into a string (of finite length)
- Every string is either a valid TM representation or not.
- Generate all strings, one after the other.
- Check to see if it is a valid TM.

**Approach #2**

The alphabets are finite.
There are a finite number of kinds of transitions

\[ a \Rightarrow b, R \]

For \( i = 1, 2, 3, \ldots \)

- There is a finite number of directed graphs with \( i \) nodes.
- There is a finite number of ways to label the edges.
- Generate them all.

END
THEOREM

The set of all infinite length strings over \( \Sigma = \{0, 1\} \) is uncountably infinite.

PROOF (by diagonalization method)

Assume the set of infinite binary strings can be enumerated. [i.e., correspondence with \( \mathbb{N} \)]

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
\vdots \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
\end{array}
\]

is an infinite length binary string which is not equal to any string on this list!
The set of all finite length strings over $\Sigma = \{a, b\}$ is countable.

$\epsilon \ a \ b \ ab \ ba \ aa \ bb \ aaa \ aab \ ...$

A LANGUAGE contains some of those strings and not others.

$\epsilon \ a \ b \ ab \ ba \ aa \ bb \ aaa \ aab \ ...$

A LANGUAGE can be fully specified by giving an infinite length binary string.

$1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ ...$

There are uncountably many infinite length binary strings.

**THEREFORE**

The number of languages is **UNCOUNTABLY INFINITE!**
Theorem

The set of all Turing Machines is countably infinite.

Corollary

The set of all Turing-recognizable languages is countably infinite.

Theorem

The set of all languages is uncountably infinite.

Corollary

Some languages are not Turing recognizable!

A whole lot, too!
**THE HALTING PROBLEM**

\[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a Turing Machine and } M \text{ accepts } w \} \]

\[ \uparrow \text{ THIS SET IS UNDECIDABLE.} \]

**PROOF**

Assume \( A_{TM} \) is decidable.

Let \( H \) be the algorithm that decides \( A_{TM} \).

\[ H(\langle M, w \rangle) = \begin{cases} 
\text{ACCEPT, if } M \text{ accepts } w \\
\text{REJECT, if } M \text{ does not accept } w 
\end{cases} \]

\[ \uparrow \text{ Doesn't really exist; we are headed toward a contradiction.} \]

Using \( H \), construct a new machine, \( D \) \( \text{"Devil Machine"} \).

Try to run \( D \).

Find a contradiction.

\[ \therefore H = \text{A decider for Halting Problem cannot exist.} \]
PROOF - MORE DETAIL

HALTING DECIDER

\[ H(\langle M, w \rangle) = \begin{cases} 
\text{ACCEPT, if } M \text{ accepts } w. \\
\text{REJECT, if } M \text{ does not accept } w.
\end{cases} \]

DEVIL MACHINE

Input to D: \( \langle M \rangle = \) The description of a TM, \( M \).

Action:

Run \( H \) as a subroutine.

Ask whether machine \( M \) would accept if given a description of itself as input.

Output: Do the opposite.

\( H \) accepts \( \Rightarrow \) REJECT
\( H \) rejects \( \Rightarrow \) ACCEPT.
RECAP OF D'S BEHAVIOR

Run $H(\langle M, M \rangle)$

Given a machine $M$, ask whether that machine would accept when given a description of itself as input.

$D(\langle M \rangle) = \begin{cases} 
\text{ACCEPT, if } M \text{ does not accept } \langle M \rangle. \\
\text{REJECT, if } M \text{ accepts } \langle M \rangle.
\end{cases}$

Now run $D$ on itself!

$D(\langle D \rangle) = \begin{cases} 
\text{ACCEPT, if } D \text{ does not accept } \langle D \rangle. \\
\text{REJECT, if } D \text{ accepts } \langle D \rangle.
\end{cases}$

PARADOX!

The next sentence is true.
The previous sentence is false.
H accepts $\langle M, w \rangle$ exactly when $M$ accepts $w$.

D rejects $\langle M \rangle$ exactly when $M$ accepts $\langle M \rangle$.

D rejects $\langle D \rangle$ exactly when $D$ accepts $\langle D \rangle$.
"If a language $L$ is decidable, then $L$ is Turing recognizable and its complement $\overline{L}$ is Turing recognizable."

- Every decidable language is Turing recognizable.
- Want to recognize $\overline{L}$?
  i.e., is $x$ in $\overline{L}$?
  Just run the decider for $L$ and give the opposite answer.
"If a language $L$ is Turing recognizable and its complement $\overline{L}$ is also Turing recognizable, then $L$ is decidable."

Want to decide $L$?

i.e., Is $x$ in $L$?

Let $M_1$ be the recognizer for $L$.

Let $M_2$ be the recognizer for $\overline{L}$.

Run $M_1$ and $M_2$ in parallel.

$x$ is in either $L$ or $\overline{L}$.

Either $M_1$ or $M_2$ (or both) will eventually halt and one of them will accept.

If $M_1$ accepts, then accept.

If $M_2$ accepts, then reject.

Either way, you'll eventually halt with a decision.
**Theorem**

A language $L$ is decidable iff $L$ and $\overline{L}$ are Turing recognizable.

**Definition**

A language is "co-Turing recognizable" if its complement is Turing recognizable.

**Restating the Theorem**

A language is decidable iff it is Turing recognizable and co-Turing recognizable.
RECALL:

\[ A_T \text{ is Turing Recognizable.} \]
\[ A_T \text{ is NOT Decidable.} \]

THEREFORE:

\[ \overline{A_T} \text{ is NOT Turing Recognizable.} \]

PROOF

Assume \[ A_T \] is Turing Recognizable.

If a Language and its complement are both Turing Recognizable, then the language is decidable.

But we know \[ A_T \] is not decidable.