Reducibility:
A technique for proving undecidability
REDUCIBILITY

How can we prove that some problems are undecidable?

New Technique:

"Reduce" one problem to another problem.

Results:

The equivalence of two Turing machines? ... undecidable!

Is a given program/algorithm guaranteed to halt? ... undecidable!

Will a given Turing machine accept any string? ... undecidable!
An engineer and a mathematician were hiking when they were suddenly attacked by a bear. The engineer grabbed a stick and, yelling and stabbing wildly with the stick, managed to fight off the bear.

The next day, when they were hiking, they were attacked again by a bear.

The mathematician picked up a nearby stick and handed it to the engineer, thereby “REDUCING” the problem to a previously solved problem.
**Reducibility**

We want to "reduce" a hard problem into an easier problem. A solution to the easier problem can then be used to solve the harder problem.

**Hard Problem:**
Fly from Portland to Cairo.

We know there are direct flights from Portland to New York.

So we have "reduced" the problem to an easier problem.

**Easier Problem:**
Fly from New York to Cairo.

So... if we can find a solution to the easier problem, we can use it to solve the harder problem!
REVERSING THE LOGIC

REDUCE A HARD PROBLEM INTO AN EASIER PROBLEM.

BUT WHAT IF THE HARD PROBLEM IS KNOWN TO BE UNSOLVABLE?

THEN THE "EASIER" PROBLEM MUST ALSO BE UNSOLVABLE!

HARD PROBLEM: LIVE FOREVER

Known to be impossible?

"EASIER" PROBLEM: STOP AGING.

If we can find a solution to the "STOP AGING" problem, then we could solve the "LIVE FOREVER" problem.

BUT: WE KNOW "LIVING FOREVER" IS IMPOSSIBLE. SO WE CAN CONCLUDE THAT IT IS IMPOSSIBLE TO "STOP AGING."
**Known Fact:** $\text{ATM}$ is undecidable.

What about some other problem $P$... is $P$ undecidable?

**Theorem**

$P$ is undecidable.

**Proof Approach**

- Assume $P$ is decidable.
- Reduce $\text{ATM}$ (a "hard" problem) into $P$ (the "easier" problem).
- Use the solution of $P$ to solve $\text{ATM}$.
- Use the decidability of $P$ to find an algorithm to decide $\text{ATM}$.
- Build a TM to decide $\text{ATM}$ using the TM to decide $P$ as a subroutine.
- But we know that a decider for $\text{ATM}$ cannot exist.

\[\therefore\text{Contradiction: } P \text{ is not decidable!}\]
The Halting Problem
Proof of its Undecidability
THE "HALTING" PROBLEM

"Does a program halt when given a specific input?"

**Theorem**

The language

\[ \text{HALT}_M = \{ \langle M, w \rangle \mid M \text{ is a Turing Machine and } M \text{ halts on input } w \} \]

is UNDECIDABLE.

**Proof**

• Assume it is decidable.
  
  There is a Turing Machine \( R \) that decides \( \text{HALT}_M \).

• Use \( R \) to build another Turing Machine, \( S \), that decides \( \text{ATM} \).

**Reduce ATM to \( \text{HALT}_M \)**

• But \( \text{ATM} \) is undecidable.

• CONTRADICTION!
\[ \text{Atm} = \{ \langle M, w \rangle \mid M \text{ is a Turing Machine that accepts } w \} \]

To decide Atm...
- Given M and some input w,
  - If M accepts w, then \( \Rightarrow \) Accept.
  - If M rejects w or loops, then \( \Rightarrow \) Reject.

Deciders can never loop!
(Note: "S" is proven not to exist!)

**GIVEN**

\[ R = \text{A Turing Machine that decides:} \]

\[ \text{HALT}_{M} = \{ \langle M, w \rangle \mid M \text{ is a Turing Machine that halts on } w \} \]

To decide HALT_{M}
- Given M and some input w,
  - If M accepts or rejects w...
    - Then \( \Rightarrow \) Accept
  - If M loops...
    - Then \( \Rightarrow \) Reject
GOAL: Construct \( S \), a decider for \( \text{ATM} \)

\( R \) = Algorithm to \text{DECIDE} \( \text{HALT}_M \)
\( S \) = Algorithm to \text{DECIDE} \( \text{ATM} \).

\text{Here is Algorithm} \( S \):

\text{Input:} \( \langle M, w \rangle \)

Run \( R \) on \( \langle M, w \rangle \) to see if \( M \) halts or loops.

If \( R \) rejects, it means \( M \) loops.
\text{Then} \rightarrow \text{REJECT}

If \( R \) accepts, it means \( M \) halts.
(And \( R \) will never loop.)

Simulate/Run \( M \) on input \( w \).
When \( M \) halts...
If \( M \) accepts, \text{Then} \rightarrow \text{ACCEPT}
If \( M \) rejects, \text{Then} \rightarrow \text{REJECT}

So \( S \) is a \text{DECIDER} for \( \text{ATM} \).

\text{Contradiction:} \( \text{ATM} \) is \text{UNDECIDABLE}. 
Does a Turing Machine accept any strings?
**Theorem**

**Does a T.M. machine accept any string?** — UNDECIDABLE.

The language

\[ E_{TM} = \{ \langle M \rangle \mid M \text{ is a Turing Machine and } L(M) = \emptyset \} \]

is UNDECIDABLE.

**Proof:**

- Assume that \( R \) decides \( E_{TM} \).
- Use it to construct \( S \), a decider for \( A_{TM} \).
- Contradiction.

---

**Approach:**
We will reduce \( A_{TM} \) to \( E_{TM} \).
Goal:
- Construct an algorithm (S) which is given \(< M, w >\) and decides whether \(M\) accepts \(w\).

Step 1:
- Modify \(M\) a little bit.
- Call the new machine \(M'\)

\[ M' \]

(assume: \(w\) is a constant)
- Will reject all strings that do not match \(w\).
- May or may not accept \(w\).

\[ L(M') = \frac{3}{2}w \cup \emptyset \]

Algorithm for \(M'\)

**Input:** \(x\)

**If** \(x \neq w\), **then reject**.

Otherwise,

**Simulate** \(M\) on \(x\).

**If** \(M\) accepts, **then accept**.

**If** \(M\) rejects, **then reject**.
Algorithm for $S$: (to decide ATM)

Input: $M, w$

Step 1: Construct $M'$.

Take $M$.
Add a test $(x = w)$ in front of the initial state.
Then pass control to $M$.

Note:

$L(M') = \{w^3\}$ if $M$ accepts $w$.
$L(M') = \emptyset$ otherwise.

Step 2:

Use $R$ to decide whether $L(M')$ is empty or not.

$R$ accepts $\Rightarrow L(M')$ is empty $\Rightarrow M$ does not accept $w$

$R$ rejects $\Rightarrow L(M') = \{w^3\} \Rightarrow M$ accepts $w$.

Step 3: We now have decided ATM.

CONTRADICTION! ($E_{TM}$ is UNDECIDABLE)
Computable Functions
DEFINITION

A "COMPUTABLE" FUNCTION:

A function

\[ f: \Sigma^* \rightarrow \Sigma^* \]

that can be computed by a Turing machine.

- The input x is on the tape.
- The TM runs and **always halts**!
- The result \( f(x) \) is left on the tape.
EQUIVALENCE OF TWO TURING MACHINES
**Theorem**

The language

\[ EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are Turing Machines and } L(M_1) = L(M_2) \}\]

is undecidable.

**In plain English?**

"Given two programs, you can not compare them to see whether they do the same thing."

"Any correct algorithm to compare the functionality of two programs will sometimes fail to halt."
\[ EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2) \} \]

**Proof Approach**

**Reduce \( ETm \) to \( EQ_{TM} \).**

We already know \( ETm \) is **undecidable**.

\[ A_{TM} \leq_m E_{TM} \leq_m EQ_{TM} \]

Symbol for "Mapping Reducible"

It means we can perform the transformation of one problem into another with **computable algorithm**.

(A Turing Machine that always halts)
Given an algorithm to solve $EQ_{TM}$, how do we proceed?

$R = \text{decider for } EQ_{TM}$

$S = \text{decider for } ET_{TM}$

**Input to $S$:** $M$

- Let $M_\emptyset$ be a Turing machine that always rejects.
- Write $M_\emptyset$ on the tape after $M$.
- Call $R$ to decide whether $L(M) = L(M_\emptyset)$
PROVING PROGRAM CORRECTNESS

"INFORMAL IDEA OF WHAT
YOU WANT THE PROGRAM
TO DO."

Subject to human error; artform?

"FORMAL SPECIFICATION"
\forall x. (x is a mouse click) \Rightarrow ...

"SPECIFICATION LANGUAGE"

UNDECIDABLE IN GENERAL,
But that does not mean
you can't find a proof
in some cases.

"CODE"

Program/algorithm expressed
in some verifiable
programming language.
ANOTHER APPROACH

"INFORMAL IDEA OF WHAT YOU WANT THE PROGRAM TO DO."

"SPECIFICATION"

CODE, IMPLEMENTATION #1

Team 1

CODE, IMPLEMENTATION #2

Team 2

IDEA:
Try to prove these algorithms are equivalent.
(Must use human guidance to)
direct search.
Problems in finding a proof?
May indicate ambiguities, or
lack of detail in the specification.
REDUCE ONE LANGUAGE TO ANOTHER.
Reducing one problem into another

Defn

Reducing languages.

Language $A$ "reduces" to language $B$...

$A \leq_m B$

if there exists a computable function $f: \Sigma^* \rightarrow \Sigma^*$

such that, for every $x$

$x \in A$ iff $f(x) \in B$
MORE ON REDUCIBILITY

If A reduces to B and B is decidable,
THEN A is decidable.

If A reduces to B and A is undecidable,
THEN B is undecidable.

If A reduces to B and B is Turing Recognizable,
THEN A is Turing Recognizable.

If A reduces to B and A is not Turing Recognizable,
THEN B is not Turing Recognizable.
The Post Correspondence Problem
PCP: Post Correspondence Problem

Dominoes:

\[
\begin{array}{cccc}
B & A & CA & ABC \\
CA & AB & A & C \\
\end{array}
\]

We get as many of each type as we need.

Goal: Find a sequence of dominoes such that the top and bottom strings are the same.

\[
\begin{array}{cccc}
A & B & CA & A \\
AB & CA & A & AB \\
& & & C \\
\end{array}
\]

Does a solution exist?

This problem is unsolvable! (i.e., UNDECIDABLE)
An "instance" of the PCP problem

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>111</td>
</tr>
<tr>
<td>2</td>
<td>10111</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

A non-solution: ① ②

A: 11110111
B: 111111110

A solution: ② ① ③

A: 1011111110
B: 10.1111110
ANOTHER PCP INSTANCE

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>011</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>101</td>
<td>011</td>
</tr>
</tbody>
</table>

A: 10
B: 101

A: 011
B: 11

A: 101
B: 011

A: 101
B: 011

A: 101
B: 011

A: 101
B: 011

A: 101
B: 011

A: 101
B: 011

A: 101
B: 011

A: 101
B: 011

A: 101
B: 011

A: 101
B: 011

A: 101
B: 011
Proof that the Post Correspondence Problem is Undecidable
"Configuration"

\[ C : a b c d e g f d a c \]

A sequence of configurations:

\[ \cdots \Rightarrow C_4 \Rightarrow C_5 \Rightarrow C_6 \Rightarrow C_7 \Rightarrow \cdots \]

Step in the computation.

An "Accepting Computation History"

\[ C_1 \Rightarrow C_2 \Rightarrow C_3 \Rightarrow C_4 \Rightarrow \cdots \Rightarrow C_L \]

Legal steps for the machine

Initial starting configuration

A "Rejecting Computation History"

\[ C_1 \Rightarrow C_2 \Rightarrow C_3 \Rightarrow \cdots \Rightarrow C_L \]

Same, except the last configuration is a rejecting state.
A computation history is a finite sequence. If machine does not halt? \[\Rightarrow \text{no history}.\]

Deterministic machines: At most, one history (on a given input)

Non-deterministic machines: May have many histories.

From now on...

Assume our TM is deterministic.

There is either an accepting history, a rejecting history, or no history/machine loops.
Theorem

The post correspondence problem is undecidable.

Proof (Overview)

Reduce $A_M$ to a PCP.
If $A_M$ accepts, then $M$ computes on $w$ and accepts.
Then, there is a finite computation history which describes the computation of $M$ on $w$.

Encode $<M,w>$ into a PCP instance.
There is a solution to the PCP iff there is an accepting computation history.

If we could decide this instance of the PCP, then we could decide $A_M$.
But we can't decide $A_M$!
TILES:
\[
\begin{array}{c|c}
A & B \\
10111 & 10
\end{array}
\]
\[
10111 \\
\hline
10
\]
\[
\begin{bmatrix}
10111 \\
10
\end{bmatrix}
\]

PCP SOLUTIONS:

TURING MACHINE CONFIGURATIONS:

101 0 4 0 1 1

COMPUTATION HISTORY:

# 9_0 1 0 _1 # 1 9 4 0 1 # ... # 0 1 1 9 a 1 0 _1 #

INITIAL STATE AT LEFT END OF TAPE

THIS IS AN ACCEPTING HISTORY

\[q_a = q_{accept}\]
Given input \( \langle M, w \rangle \) to ATM,

construct an instance of the PCP...

(create a collection of tiles)

such that, if you can find

a solution to this PCP instance,

then you've found an accepting computation history.

A solution will look like this:

\[
\begin{array}{cccccccc}
# & 0 & 1 & 0 & 1 & # & 1 & 9 & 4 & 0 & 1 & # & 1 & 1 & 9 & 5 & 1 & # & \\
# & 0 & 1 & 0 & 1 & # & 1 & 9 & 4 & 0 & 1 & # & 1 & 1 & 9 & 5 & 1 & # & \\
# & 0 & 1 & 0 & 1 & # & 1 & 9 & 4 & 0 & 1 & # & 1 & 1 & 9 & 5 & 1 & # & \\
\end{array}
\]

\[\text{SPECIAL STARTING TILE}\]

\[1 \rightarrow 1, R\]

\[\text{ONE TILE FOREACH TRANSITION}\]
SPECIAL STARTING TILE:

INITIAL STATE

RIGHT MOVES:
\[ a \rightarrow b, R \]

LEFT MOVES

TILES TO "COPY" THE TAPE:

For every symbol in \[ \Gamma \]
Q: How do we accept?
A: Complete the match!

Add special tiles to allow \( q_{\text{accept}} \) to "eat" the symbols on the tape:

\[
\begin{align*}
\# & 1 0 \quad q_{\text{accept}} \quad 1 1 \quad \# \\
\# & 1 0 \quad q_{\text{accept}} \quad 1 1 \quad \# & 1 0 \quad q_{\text{accept}} \quad 1 \quad \#
\end{align*}
\]

For every symbol in \( I \)
EVENTUALLY, NOTHING REMAINS
EXCEPT GACC.

ADD A TILE TO FINISH THE MATCH.
REVIEW OF THE PROOF.

• If you can find a solution to this instance of the PCP, then you have found a legal accepting computation history, in which machine $M$ accepts string $w$.

• Does a solution exist?

• If you can decide the answer, then you can decide whether $M$ accepts $w$.

• We know that $A_{TM}$ is undecidable.

• Therefore, we have proven that the problem of finding a solution to the PCP is undecidable!

(In general).
LINEAR
BOUNDDED
AUTOMATA
LINEAR BOUNDED AUTOMATON (LBA)

A RESTRICTED FORM OF A TURING MACHINE.

The tape is limited in size to the size of the input.
(i.e., head not allowed to move beyond the input.)

LBAs are not as powerful as full Turing Machines.

But are really quite powerful.
Assume our tape alphabet can be larger than input alphabet.

We can use a larger tape alphabet to store more information in our limited tape.

Or equivalently...

We can restrict our machine to using only a small portion of the tape.

"A linear function of the input size"

"Small" problem

CONTROL

01_____

"Small" problem = short input length

"Large" problem

CONTROL

1010_____

Tape size limited to 3x input size
THEOREM

THE LANGUAGE

\[ A_{LBA} = \{ <M, w> \mid M \text{ is an LBA and } M \text{ accepts } w \} \]

IS DECIDABLE!!
Consider some LBA...

$q = \text{Number of states}$

$G = \text{Size of tape alphabet}$

$n = \text{Length of tape.}$

How many distinct configurations are there?

This may be a big number but it is finite.
ALBA is DECIDABLE.

\[ \exists \langle M, w \rangle \mid M \text{ is an LBA that accepts } w \]

**Proof**

Just simulate \( M \) on \( w \).

\( M \) could accept, reject, or loop.

How can we detect looping?

If \( M \) ever enters a configuration it has already been in, then it must loop forever.

There are only finitely many possible configurations.

\[ q^n q^n \]

If the simulation goes that long, it must be looping.

So just run the simulation \( q^n q^n \) steps.

If it has not halted by then, then reject.
Linear bounded automata (LBAs) are very powerful.

Not full Turing machine power, but...

What problems can they solve?

$A_{DFA}$
$A_{CFG}
E_{DFA}$
$E_{CFG}$

Decidable by an LBA.