THE CHURCH-TURING THESIS

Intuitive Notion of Algorithms EQUALS Turing Machines
UNDECIDABILITY II: REDUCTIONS
TUESDAY Feb 18
\[ A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \} \]

**\[ A_{TM} \text{ is undecidable: (constructive proof & subtle) } \]**

Assume machine H **semi-decides** \[ A_{TM} \] (such exist, why?)

\[
H( (M,w) ) = \begin{cases} 
\text{Accept} & \text{if } M \text{ accepts } w \\
\text{Rejects or loops} & \text{otherwise}
\end{cases}
\]

Construct a new TM \[ D_H \] as follows: on input M, run H on (M,M) and output the “**opposite**” of H whenever possible.
\[ D_H ( D_H ) = \begin{cases} 
\text{Reject if } D_H \text{ accepts } D_H \\
\text{(i.e. if } H( D_H, D_H ) = \text{Accept}) \\
\text{Accept if } D_H \text{ rejects } D_H \\
\text{(i.e. if } H( D_H, D_H ) = \text{Reject}) \\
\text{loops if } D_H \text{ loops on } D_H \\
\text{(i.e. if } H( D_H, D_H ) \text{ loops})
\end{cases} \]

\textbf{Note:} It must be the case that \( D_H \) loops on \( D_H \)

There is \textbf{no} contradiction here!

Thus we have \textit{effectively} constructed an instance which does not belong to \( A_{TM} \) (namely, \((D_H, D_H)\)) but \( H \) fails to tell us that.
That is:

Given any semi-decision machine $H$ for $A_{TM}$ (and thus a potential decision machine for $A_{TM}$), we can effectively construct an instance which does not belong to $A_{TM}$ (namely, $(D_H, D_H)$) but $H$ fails to tell us that.

So $H$ cannot be a decision machine for $A_{TM}$.
In most cases, we will show that a language $L$ is undecidable by showing that if it is decidable, then so is $A_{TM}$.

We reduce deciding $A_{TM}$ to deciding the language in question $A_{TM} \ 	ext{"<"} \ L$.
THE HALTING PROBLEM

$\text{HALT}_{\text{TM}} = \{ (M,w) \mid M \text{ is a TM that halts on string } w \}$

**Theorem:** $\text{HALT}_{\text{TM}}$ is undecidable

**Proof:** Assume, for a contradiction, that TM $H$ decides $\text{HALT}_{\text{TM}}$

We use $H$ to construct a TM $D$ that decides $A_{\text{TM}}$

On input $(M,w)$, $D$ runs $H$ on $(M,w)$

- If $H$ rejects then reject
- If $H$ accepts, run $M$ on $w$ until it halts:
  - Accept if $M$ accepts and
  - Reject if $M$ rejects
If $M$ doesn't halt: REJECT

If $M$ halts:

Does $M$ halt on $w$?

ACCEPT if halts in accept state
REJECT otherwise
In most cases, we will show that a language $L$ is undecidable by showing that if it is decidable, then so is $A_{TM}$.

We **reduce** deciding $A_{TM}$ to deciding the language in question.

$$A_{TM} \text{ } \text{“<“} \text{ } L$$

So, $A_{TM} \text{ } \text{“<“} \text{ } \text{Halt}_{TM}$

Is $\text{Halt}_{TM} \text{ } \text{“<“} \text{ } A_{TM}$?
\[ A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \} \]

\[ \text{HALT}_{TM} = \{ (M, w) \mid M \text{ is a TM that halts on string } w \} \] (*)

\[ E_{TM} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \} \] (*)

\[ \text{REG}_{TM} = \{ M \mid M \text{ is a TM and } L(M) \text{ is regular} \} \] (*)

\[ \text{EQ}_{TM} = \{ (M, N) \mid M, N \text{ are TMs and } L(M) = L(N) \} \] (*)

\[ \text{ALL}_{PDA} = \{ P \mid P \text{ is a PDA and } L(P) = \Sigma^* \} \] (*)

ALL UNDECIDABLE

(*) Use Reductions to Prove Which are SEMI-DECIDABLE?

What about complements?
$E_{\text{TM}} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \}$

**Theorem:** $E_{\text{TM}}$ is undecidable

**Proof:** Assume, for a contradiction, that TM $Z$ decides $E_{\text{TM}}$. Use $Z$ as a subroutine to decide $A_{\text{TM}}$.

Algorithm for deciding $A_{\text{TM}}$: On input $(M,w)$:

1. Create $M_w$
2. Run $Z$ on $M_w$

So, $L(M_w) = \emptyset \iff M(w)$ does not accept

$L(M_w) \neq \emptyset \iff M(w)$ accepts
Erase $s$, run $M(w)$

So, $L(M_w) = \emptyset \iff M(w)$ does not accept

Decision Machine for $A_{TM}$

Accepts if $M$ does not accept $w$
Rejects, otherwise

REVERSE accept/reject
Theorem: REGULAR_{TM} is undecidable

Proof: Assume, for a contradiction, that TM \( R \) decides REGULAR_{TM}

Use \( R \) as a subroutine to decide \( A_{TM} \)

1. Create \( M'_w \)

2. Run \( R \) on \( M'_w \)

So, \( L (M'_w) = \Sigma^* \iff M(w) \) accepts

\( L (M'_w) = \{0^n1^n\} \iff M(w) \) does not accept
If $s = 0^n1^n$, accept
Else run $M(w)$

$L(M_w') = \Sigma^*$ if $M(w)$ accepts
{$0^n1^n$} otherwise

$L(M_w')$ is regular $\iff M(w)$ accepts

Is $L(M_w')$ regular?

Yes $\iff M$ accepts $w$
**MAPPING REDUCIBILITY**

$f : \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.

A language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B$$

$f$ is called a **reduction** from $A$ to $B$.

Think of $f$ as a "**computable coding**".
A is mapping reducible to B, $A \leq_m B$, if there is a computable $f : \Sigma^* \rightarrow \Sigma^*$ such that $w \in A \iff f(w) \in B$

Also, $\neg A \leq_m \neg B$, why?
Theorem: If \( A \leq_m B \) and \( B \) is decidable, then \( A \) is decidable.

Proof: Let \( M \) decide \( B \) and let \( f \) be a reduction from \( A \) to \( B \).

We build a machine \( N \) that decides \( A \) as follows:

On input \( w \):

1. Compute \( f(w) \)
2. Run \( M \) on \( f(w) \)
Theorem: If $A \leq_m B$ and $B$ is (semi) decidable, then $A$ is (semi) decidable

Proof: Let $M$ (semi) decide $B$ and let $f$ be a reduction from $A$ to $B$

We build a machine $N$ that (semi) decides $A$ as follows:

On input $w$:

1. Compute $f(w)$
2. Run $M$ on $f(w)$
All undecidability proofs from today can be seen as constructing an \( f \) that reduces \( A_{TM} \) to the proper language.

(Sometimes you have to consider the complement of the language.)
All undecidability proofs from today can be seen as constructing an $f$ that reduces $A_{TM}$ to the proper language $A_{TM} \leq_m Halt_{TM}$ (So also, $\neg A_{TM} \leq_m \neg Halt_{TM}$):

Map $(M, w) \rightarrow (M', w)$
where $M'(w) = M(w)$ if $M(w)$ accepts
loops otherwise

So $(M, w) \in A_{TM} \iff (M', w) \in Halt_{TM}$
$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

$E_{TM} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \}$

**CLAIM:** $A_{TM} \leq_m \neg E_{TM}$

**CONSTRUCT** $f : \Sigma^* \rightarrow \Sigma^*$

$f : (M,w) \rightarrow M_w$ where $M_w(s) = M(w)$

So, $M(w)$ accepts $\iff L(M_w) \neq \emptyset$

So, $(M, w) \in A_{TM} \iff M_w \in \neg E_{TM}$

So $\neg E_{TM}$ is NOT DECIDABLE, but it is SEMI-DECIDABLE (why?) Is $E_{TM}$ SEMI-DECIDABLE?
$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

$REG_{TM} = \{ M \mid M \text{ is a TM and } L(M) \text{ is regular} \}$

CLAIM: $A_{TM} \leq_m REG_{TM}$  So $REG_{TM}$ is UNDECIDABLE

CONSTUCT $f : \Sigma^* \rightarrow \Sigma^*$

$f: (M,w) \rightarrow M'_w \text{ where } M'_w(s) = \text{ accept if } s = 0^n1^n M(w) \text{ otherwise}$

So, $L(M'_w) = \Sigma^*$ if $M(w)$ accepts

$\{0^n1^n\}$ if not

So, $(M, w) \in A_{TM} \iff M'_w \in REG_{TM}$

Is REG SEMI-DECIDABLE? ($\neg$ REG is not. Why?)
$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

$REG_{TM} = \{ M \mid M \text{ is a TM and } L(M) \text{ is regular} \}$

CLAIM: $\neg A_{TM} \leq_{m} REG_{TM}$ So $REG_{TM}$ is NOT SEMI-DECIDABLE

CONSTRUCT $f : \Sigma^* \rightarrow \Sigma^*$

$f: (M,w) \rightarrow M''_w$ where $M''_w(s) = \text{accept if } s = 0^n1^n$ and $M(w)$ accepts 

Loop otherwise

So, $L(M'_w) = \{0^n1^n\}$ if $M(w)$ accepts 

$\emptyset$ if not

So, $(M, w) \notin A_{TM} \iff M''_w \in REG_{TM}$

So, $REG$ NOT SEMI-DECIDABLE
\[ A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \} \]

\[ \text{HALT}_{TM} = \{ (M, w) \mid M \text{ is a TM that halts on string } w \} \]

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

\[ \text{REG}_{TM} = \{ M \mid M \text{ is a TM and } L(M) \text{ is regular} \} \]

\[ \text{EQ}_{TM} = \{ (M, N) \mid M, N \text{ are TMs and } L(M) = L(N) \} \]

\[ \text{ALL}_{PDA} = \{ P \mid P \text{ is a PDA and } L(P) = \Sigma^* \} \]

ALL UNDECIDABLE

Which are SEMI-DECIDABLE?

What about complements?
\[ E_{TM} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \} \]

\[ EQ_{TM} = \{ (M, N) \mid M, N \text{ are TMs and } L(M) = L(N) \} \]

CLAIM: \( E_{TM} \leq_m EQ_{TM} \) \( \Rightarrow \) \( EQ_{TM} \) is UNDECIDABLE

CONSTRUCT \( f : \Sigma^* \rightarrow \Sigma^* \)

\[ f : M \rightarrow (M, M \emptyset) \text{ where } M \emptyset (s) = \text{Loops} \]

\[ \text{So, } M \in E_{TM} \iff (M, M \emptyset) \in EQ_{TM} \]

Is \( EQ_{TM} \) SEMI-DECIDABLE? \( \text{NO, since,} \]

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

What about \( \neg EQ_{TM} \)?
$A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \}$

$EQ_{TM} = \{ (M, N) \mid M, N \text{ are TMs and } L(M) = L(N) \}$

CLAIM: $A_{TM} \leq_m EQ_{TM}$

So, $\neg EQ_{TM}$ is not semi-decidable

CONSTRUCT $f : \Sigma^* \rightarrow \Sigma^*$

$f : (M, w) \rightarrow (M_w, M_A)$

Where for each $s$ in $\Sigma^*$,

$M_w(s) = M(w)$ and $M_A(s)$ always accepts

So, $(M, w) \in A_{TM} \iff (M_w, M_A) \in EQ_{TM}$
Undecidable given a TM to tell if the language it recognizes is empty. It’s not even semi-decidable, altho it is semi-decidable to tell if the language is non-empty.

Undecidable given a TM to tell if it is equivalent to a FSM. It’s not even semi-decidable, nor is it semi-decidable to tell if it is not equivalent to a FSM.

Undecidable given 2 TMs to tell if they are equivalent. It’s not even semi-decidable, nor is it semi-decidable to tell if they are not
A_{TM} = \{ (M,w) | M \text{ is a TM that accepts string } w \} \\
ALL_{PDA} = \{ P | P \text{ is a PDA and } L(P) = \Sigma^* \} \\

CLAIM: A_{TM} \leq_m \neg ALL_{PDA} \\

CONSTRUCT f : \Sigma^* \rightarrow \Sigma^* \\

Idea! More subtle construction

Map \((M,w)\) to a PDA \(P_w\) that recognizes \(\Sigma^*\) if and only if \(M\) does not accept \(w\)

So, \((M, w) \notin A_{TM} \iff P_w \in ALL_{PDA}\)

\(P_w\) will recognize all (and only those) strings that are NOT accepting computation histories for \(M\) on \(w\)
CONFIGURATIONS

11010q_700110

q_7

11010100001110
COMPUTATION HISTORIES

An accepting computation history is a sequence of configurations $C_1, C_2, \ldots, C_k$, where

1. $C_1$ is the start configuration,
2. $C_k$ is an accepting configuration,
3. Each $C_i$ follows from $C_{i-1}$

An rejecting computation history is a sequence of configurations $C_1, C_2, \ldots, C_k$, where

1. $C_1$ is the start configuration,
2. $C_k$ is a rejecting configuration,
3. Each $C_i$ follows from $C_{i-1}$
An **accepting computation history** is a sequence of configurations $C_1, C_2, \ldots, C_k$, where

1. $C_1$ is the start configuration,
2. $C_k$ is an accepting configuration,
3. Each $C_i$ follows from $C_{i-1}$

An **rejecting computation history** is a sequence of configurations $C_1, C_2, \ldots, C_k$, where

1. $C_1$ is the start configuration,
2. $C_k$ is a rejecting configuration,
3. Each $C_i$ follows from $C_{i-1}$

$M$ accepts $w$ if and only if there exists an accepting computation history that starts with $C_1 = q_0w$
P will recognize all strings (read as sequences of configurations) that:

1. Do not start with $C_1$ or
2. Do not end with an accepting configuration or
3. Where some $C_i$ does not properly yield $C_{i+1}$

Non-deterministic checks for 1, 2, and 3.
P will reject all strings (read as sequences of configurations) that:

1. Start with $C_1$ and
2. End with an accepting configuration and
3. Where each $C_i$ properly yields $C_{i+1}$

Non-deterministic checks for 1, 2, and 3.
\{ 0^{2^n} \mid n \geq 0 \}
\{ 0^{2n} \mid n \geq 0 \}
P recognizes all strings except accepting computation histories:

$$\text{#C}_1 \text{# C}_2^R \text{#C}_3 \text{#C}_4^R \text{#C}_5 \text{#C}_6^R \text{#...# C}_k$$

If $i$ is odd, put $C_i$ on stack and see if $C_{i+1}^R$ follows properly:

For example,

If $uaq_i bv$ and $\delta (q_i, b) = (q_j, c, R)$,

then $C_i$ properly yields $C_{i+1} \iff C_{i+1} = uacq_jv$
P recognizes all strings except accepting computation histories:

#\text{C}_1# \text{C}_2^R \#\text{C}_3 \#\text{C}_4^R \#\text{C}_5 \#\text{C}_6^R \#\ldots\# \text{C}_k

If \(i\) is odd, put \text{C}_i on stack and see if \text{C}_{i+1}^R follows properly:

For example,

If \(=u\text{aq}_i\text{b}v\) and \(\delta (q_i, b) = (q_j, c, L),\)
then \(C_k\) properly yields \(C_{k+1} \Leftrightarrow C_{k+1} = u\text{q}_j\text{a}c\text{v}\)
P recognizes all strings except accepting computation histories:

#C₁# C₂^{R} #C₃ #C₄^{R} #C₅ #C₆^{R} #....# Cₖ

If i is even, put Cᵢ^{R} on stack and see if Cᵢ₊₁ follows properly.
EVEN
EVEN

$q_00000$

$q_1000$

$q_300$

$xq_300$

$x0q_40$

$x0xq_3$

$x0q_2x$

$xq_20x$

$q_2x0x$

$q_00000#000q_1#xq_300#0q_40x#x0xq_3#...#$
\[ A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \} \]

\[ \text{ALL}_{PDA} = \{ P \mid P \text{ is a PDA and } L(P) = \Sigma^* \} \]

CLAIM: \( A_{TM} \leq_m \neg \text{ALL}_{PDA} \)

CONSTRUCT \( f : \Sigma^* \rightarrow \Sigma^* \)

\[ f : (M, w) \rightarrow P_w \text{ where } \]

\[ P_w (s) = \text{accept} \iff s \text{ is NOT an accepting computation of } M(w) \]

So, \((M, w) \notin A_{TM} \iff P_w \in \text{ALL}_{PDA} \)

So, \((M, w) \in A_{TM} \iff P_w \in \neg \text{ALL}_{PDA} \)

EXPLAIN THE PROOF TO YOUR NEIGHBOR
\( A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \} \)

\( \text{HALT}_{TM} = \{ (M,w) \mid M \text{ is a TM that halts on string } w \} \)

\( E_{TM} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \} \)

\( \text{REG}_{TM} = \{ M \mid M \text{ is a TM and } L(M) \text{ is regular} \} \)

\( \text{EQ}_{TM} = \{ (M, N) \mid M, N \text{ are TMs and } L(M) = L(N) \} \)

\( \text{ALL}_{PDA} = \{ P \mid P \text{ is a PDA and } L(P) = \Sigma^* \} \)

**ALL UNDECIDABLE**

Which are SEMI-DECIDABLE?

What about complements?
Read chapter 5.1-5.3 of the book for next time
THE PCP GAME

ba
---
a

a
---
ab

b
---
bcb

b
---
a
GENERAL RULE #1

If every top string is longer than the corresponding bottom one, there can’t be a match
GENERAL RULE #2

If there is a domino with the same string on the top and on the bottom, there is a match
POST CORRESPONDENCE PROBLEM
Given a collection of dominos, is there a match?
PCP = \{ P \mid P \text{ is a set of dominos with a match} \}

PCP is undecidable!
THE FPCP GAME

... is just like the PCP game except that a match has to start with the first domino
FPCP

aaa
---
a

-----
c

-----
aa

-----
a

-----
c

-----
a
Theorem: FPCP is undecidable

Proof: Assume machine C decides FPCP

We will show how to use C to decide $A_{TM}$
Given \((M, w)\)

we will construct a set of dominos \(P_{M,w}\) where a match is an accepting computation history for \(M\) on \(w\)

\[
P_{M,w} = \begin{array}{ccc}
\text{caa} & \text{aba} & \text{...} \\
\text{c} & \text{bb} & \text{d}
\end{array}
\]
\{ 0^{2^n} \mid n \geq 0 \}

Diagram:

- States: q_0, q_1, q_2, q_3, q_4
- Transitions:
  - 0 \rightarrow \square, R
  - x \rightarrow x, R
  - x \rightarrow x, L
  - 0 \rightarrow 0, L
  - 0 \rightarrow 0, R
\{ 0^{2n} \mid n \geq 0 \}
Given \((M,w)\), we will construct an instance \(P_{M,w}\) of FPCP in 7 steps.

Assume \(M\) on \(w\) never attempts to move off the left hand edge of tape.
STEP 1

Put \#q_0w_1w_2\ldots w_n\# into P

For start configuration

START
**STEP 2**

If $\delta(q,a) = (p,b,R)$ then add $\underbrace{qa}_{bp}$

**STEP 3**

If $\delta(q,a) = (p,b,L)$ then add $\underbrace{cqa}_{pcb}$ for all $c \in \Gamma$
$$\{ 0^{2^n} \mid n \geq 0 \}$$
**STEP 4**

For all \( a \in \Gamma \) add:

For tape cells not adjacent to head.

**STEP 5**

For configuration separator add:

To simulate the blanks on the right hand side of tape.

CONTINUE
**STEP 4**

Add $a$ for all $a \in \Gamma$

**STEP 5**

Add $#$ for all $a \in \Gamma$

**STEP 6**

Add $aq_{\text{acc}}$ for all $a \in \Gamma$
STEP 7

add $q_{acc}##$

END
Given \((M,w)\), we can construct an instance of FPCP that has a match if and only if \(M\) accepts \(w\).
Can convert an instance of FPCP into one of PCP:

Let $u = u_1u_2\ldots u_n$, define:

\[ \star u = \star u_1 \star u_2 \star u_3 \star \ldots \star u_n \]

\[ u\star = u_1 \star u_2 \star u_3 \star \ldots \star u_n \star \]

\[ \star u\star = \star u_1 \star u_2 \star u_3 \star \ldots \star u_n \star \]

FPCP:

\[
\begin{array}{ccc}
  t_1 & & t_k \\
  \hline \\
  b_1 & & b_k
\end{array}
\]

PCP:

\[
\begin{array}{ccc}
  \star t_1 & & \star t_k \\
  \hline \\
  \star b_1 & & \star b_k
\end{array}
\]

\[
\begin{array}{c}
  \star \diamond
\end{array}
\]
Given \((M, w)\), we can construct an instance of PCP that has a match if and only if \(M\) accepts \(w\)