Reducions and Undecidability

A Concrete Undecidable Problem

\( A_{TM} = \{ (M, w) | M \text{ is a TM that accepts string } w \} \)

Theorem: \( A_{TM} \) is recognizable but **NOT** decidable

Corollary: \( \neg A_{TM} \) is not recognizable

The Halting Problem

\( HALT_{TM} = \{ (M, w) | M \text{ is a TM that halts on string } w \} \)

Theorem: \( HALT_{TM} \) is undecidable

Proof: Assume, for a contradiction, that TM \( H \) decides \( HALT_{TM} \)

We use \( H \) to construct a TM \( D \) that decides \( A_{TM} \)

But \( A_{TM} \) is undecidable – contradiction.

One can often show that a language \( L \) is undecidable by showing that if \( L \) is decidable, then so is \( A_{TM} \)

We reduce \( A_{TM} \) to the language \( L \)

\( A_{TM} \leq L \)

We showed: \( A_{TM} \leq HALT_{TM} \)

Mapping Reductions

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

A language \( A \) is **mapping reducible** to language \( B \), written \( A \leq_{m} B \), if there is a computable \( f : \Sigma^* \rightarrow \Sigma^* \), such that for every \( w \),

\[ w \in A \iff f(w) \in B \]

\( f \) is called a mapping reduction (or many-one reduction) from \( A \) to \( B \)
Let $f : \Sigma^* \rightarrow \Sigma^*$ be a computable function such that $w \in A \iff f(w) \in B$.

Say: $A$ is mapping reducible to $B$
Write: $A \leq_m B$

Theorem: If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable

Proof:
Let $M$ decide $B$.
Let $f$ be a reduction from $A$ to $B$.
To decide $A$, we build a machine $M'$:

$M'(w)$:
1. Compute $f(w)$
2. Run $M$ on $f(w)$, output its answer

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

Theorem: If $A \leq_m B$ and $B$ is recognizable, then $A$ is recognizable

Proof:
Let $M$ decide $B$.
Let $f$ be a reduction from $A$ to $B$.
To decide $A$, we build a machine $M'$:

$M'(w)$:
1. Compute $f(w)$
2. Run $M$ on $f(w)$, output its answer if you ever receive one

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

Theorem: $A_{\text{TM}} \leq_m \text{HALT}_{\text{TM}}$

$f(M, w) := (M', w)$
where $M'(w) = \text{accept}$, if $M(w)$ accepts loops otherwise

So $(M, w) \in A_{\text{TM}} \iff (M', w) \in \text{HALT}_{\text{TM}}$

All undecidability proofs we've seen can be viewed as constructing an $f$ that reduces $A_{\text{TM}}$ to some language.
Is $\text{HALT}_{TM} \leq_m A_{TM}$?

Yes.

$f(M, w) := (M', w)$

where $M'(w) = \text{accept}$, if $M(w)$ ever halts loops otherwise

$(M, w) \in \text{HALT}_{TM} \iff (M', w) \in A_{TM}$

The Emptiness Problem

$\text{EMPTY}_{TM} = \{ M \mid M$ is a TM such that $L(M) = \emptyset \}$

Theorem: $\text{EMPTY}_{TM}$ is undecidable

Proof: Assume (for a contradiction) there is a TM $E$ that decides $\text{EMPTY}_{TM}$

We'll use it to get a decider $D$ for $A_{TM}$

$D(M, w) :=$ Build a TM $M'$ with the behavior:

"$M'(x) :=$ if $x \neq w$ reject else run $M(w)$"

Run $E(M')$.

If $E$ accepts, reject. If rejects, accept.

The Emptiness Problem

$\text{EMPTY}_{TM} = \{ M \mid M$ is a TM such that $L(M) = \emptyset \}$

Theorem: $\text{EMPTY}_{TM}$ is unrecognizable

Proof: Show that $\neg A_{TM} \leq_m \text{EMPTY}_{TM}$

$f(M, w) :=$ Output a TM $M'$ with the behavior:

"$M'(x) :=$ if $x \neq w$ reject else run $M(w)$"

$(M, w) \in A_{TM} \iff L(M') \neq \emptyset$

$\iff M' \not\in \text{EMPTY}_{TM}$

$\iff f(M, w) \not\in \text{EMPTY}_{TM}$

The Regularity Problem

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

Given a program, is it equivalent to some DFA?

Theorem: $\text{REGULAR}_{TM}$ is not recognizable

Proof: We show that $\neg A_{TM} \leq_m \text{REGULAR}_{TM}$

$f(M, w) :=$ Output a TM $M'$ with the behavior:

"$M'(x) :=$ if $(x = 0^n1^n)$ then sim $M(w)$ else reject"

$(M, w) \in A_{TM} \Rightarrow f(M, w) = M'$ accepts $\{0^n1^n\}$

$(M, w) \not\in A_{TM} \Rightarrow f(M, w) = M'$ accepts nothing

$(M, w) \not\in A_{TM} \iff f(M, w) \in \text{REGULAR}_{TM}$

The Equivalence Problem

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

Do two programs compute the same function?

Theorem: $\text{EQ}_{TM}$ is unrecognizable

Proof: Reduce $\text{EMPTY}_{TM}$ to $\text{EQ}_{TM}$

Let $M_\emptyset$ be some TM with no path from start state to accept state

Define $f(M) := (M, M_\emptyset)$

$M \in \text{EMPTY}_{TM} \iff L(M) = L(M_\emptyset) = \emptyset$

$\iff (M', M_\emptyset) \in \text{EQ}_{TM}$

Post's Correspondence Problem
The PCP Game, or “Domino Solitaire”

RULE #1
If every top string is longer than the corresponding bottom one, there can’t be a match
RULE #2
If there is a domino with the same string on the top and on the bottom, there is a match

Post's Correspondence Problem
Given a collection of domino types, can we build up a match?
PCP = { P | P is a set of dominos with a match }
Theorem: PCP is undecidable!

The FPCP Game
... just like the PCP game, except that a match has to start with the first domino type
Theorem: FPCP is undecidable
Use this to show PCP is undecidable

Theorem: FPCP is undecidable
Proof: We will reduce $A_{TM}$ to FPCP
Recall: an accepting computation history for $M$ on $w$ is a sequence of configs $C_0, C_1, ..., C_k$, where
1. $C_0$ is the start configuration $q_0w$,
2. $C_k$ is an accepting configuration,
3. Each configuration $C_i$ yields $C_{i+1}$
$M(w)$ accepts iff such a history exists.
We’ll build an instance of FPCP such that a match encodes an accepting computation history
Given \((M, w)\), we will construct an instance \(P\) of FPCP in seven steps.

**Step 1**
Put
\[
\text{#}
\]
\[
\text{#}q_0w_1w_2\ldots w_n#\]

The Start Domino

**Step 2**
If \(\delta(q, a) = (p, b, R)\) then add
\[
qa
bp
\]

**Step 3**
If \(\delta(q, a) = (p, b, L)\) then add
\[
cqa
pcb
\]
for all \(c \in \Gamma\)

The Transition Dominos

**Step 4**
add
\[
a
a
\]
for all \(a \in \Gamma\)

**Step 5**
add
\[
\#
\#
\#
\#
\]

Extra dominoes for filling in the configurations

**Step 4**
add
\[
a
a
\]
for all \(a \in \Gamma\)

**Step 5**
add
\[
\#
\#
\#
\#
\]

**Step 6**
add
\[
a_{\text{acc}}
q_{\text{acc}}
\]
for all \(a \in \Gamma\) (including \(\Box\))
Given \((M, w)\), we can effectively construct an instance of FPCP that has a match if and only if \(M\) accepts \(w\)

Therefore
\[ A_{TM} \leq_m \text{FPCP} \]
and FPCP is undecidable.

Can reduce FPCP to PCP:
For \(u = u_1 u_2 \ldots u_n\), where \(u_i \in \Gamma \cup Q \cup \{\#\}\), define:

\[
\begin{align*}
\star u &= * u_1 * u_2 * u_3 * \ldots * u_n \\
\star \star u &= * u_1 * u_2 * u_3 * \ldots * u_n *
\end{align*}
\]

FPCP:
\[
\begin{array}{cccc}
\star t_1 & \star t_2 & \ldots & \star t_k \\
\star b_1 & \star b_2 & \ldots & \star b_k
\end{array}
\]

PCP:
\[
\begin{array}{cccc}
\star t_1 & \star t_1 & \star t_2 & \ldots & \star t_k & \star \star \\
\star b_1 & \star b_1 & \star b_2 & \ldots & \star b_k & \star \star
\end{array}
\]

Oracle Turing Machines and Hierarchies of Undecidable Problems
Oracle Turing Machines

An oracle is a set B to which the TM may ask membership questions (formally: TM enters state \( q_1 \)) and the TM always receives a correct answer in one step (formally: if the string on the special oracle tape is in B, state \( q_1 \) is changed to \( q_{\text{YES}} \), otherwise \( q_{\text{NO}} \))

This makes sense even if B is not decidable!

HALT\(_{\text{TM}}\) is decidable in \( A_{\text{TM}} \)

On input \((M,w)\), decide if M halts on w as follows:

1. Ask the oracle for \( A_{\text{TM}} \) if M accepts w. If yes, then ACCEPT
2. Switch the accept and reject states of M to get \( M' \). Ask the oracle for \( A_{\text{TM}} \) if \( M' \) accepts w. If yes, then ACCEPT
3. REJECT

\( A_{\text{TM}} \) is decidable in HALT\(_{\text{TM}}\)

On input \((M,w)\), decide if M accepts w as follows:

Ask the oracle for HALT\(_{\text{TM}}\) if M halts on w. If yes, then run \( M(w) \) and output its answer. If no, then REJECT.
We say A is decidable in B if there is an oracle TM M with oracle B that decides A.

A “Turing Reduces” to B

\[ A \leq_T B \]

\[ \leq_T \text{ versus } \leq_m \]

Theorem: If \( A \leq_m B \) then \( A \leq_T B \)

Proof:

If \( A \leq_m B \) then there is a computable function \( f: \Sigma^* \rightarrow \Sigma^* \), where for every \( w \), \( w \in A \Leftrightarrow f(w) \in B \)

We can thus use an oracle for B to decide A.

Theorem: \( \neg \text{HALT}_{TM} \leq_T \text{HALT}_{TM} \)

Theorem: \( \neg \text{HALT}_{TM} \not\leq_m \text{HALT}_{TM} \)

Why?

Limits on Oracle TMs

The following problem cannot be decided by a TM with an oracle for the Halting Problem:

\[ \text{SUPERHALT} = \{ (M,x) \mid M, \text{with an oracle for the Halting Problem, halts on } x \} \]

Can still use diagonalization here!

Suppose H decides \( \text{SUPERHALT} \) (with oracle)

Define \( D(X) := \text{"if } H(X,X) \text{ accepts (with oracle) then LOOP, else ACCEPT."} \)

\( D(D) \) halts iff \( H(D,D) \) accepts iff \( D(D) \) loops...

Next Episode:

Adventures in Self-Reference

The Arithmetic Hierarchy

\[ \Delta^0_1 = \{ \text{decidable sets} \} \]

\[ \Sigma^0_1 = \{ \text{recognizable sets} \} \]

\[ \Sigma^0_{n+1} = \{ \text{sets recognizable in some } B \in \Sigma^0_n \} \]

\[ \Delta^0_{n+1} = \{ \text{sets decidable in some } B \in \Sigma^0_n \} \]

\[ \Pi^0_n = \{ \text{complements of sets in } \Sigma^0_n \} \]

\[ \Sigma^0_1 \cap \Pi^0_1 \]

Decidable Languages

\[ \Pi^0_1 \]

Co-R.E. Languages

R.E. Languages
Definition: A decidable predicate $R(x,y)$ is a proposition about the input strings $x$ and $y$, such that some TM $M$ implements $R$. That is, for all $x, y$, $R(x,y)$ is TRUE $\Rightarrow$ $M(x,y)$ accepts $R(x,y)$ is FALSE $\Rightarrow$ $M(x,y)$ rejects

We say $M$ “decides” the predicate $R$.

**EXAMPLES:**
$R(x,y) = “x + y \text{ is less than } 100”$
$R(N,y) = “$N$ halts on $y$ in at most 100 steps”

Note: A is decidable if and only if $A = \{x | R(x,\epsilon)\}$, for some decidable predicate $R$.

Theorem: A language $A$ is recognizable if and only if there is a decidable predicate $R(x, y)$ such that:

$$A = \{x | \exists y \ R(x,y)\}$$

Proof:
(1) If $A = \{x | \exists y \ R(x,y)\}$ then $A$ is recognizable
A TM can enumerate over all $y$’s and try them
(2) If $A$ is recognizable, then $A = \{x | \exists y \ R(x,y)\}$
Let $M$ recognize $A$ and define $R(x,y)$ to be true iff $M$ accepts $x$ in $y$ steps
(here, $y$ is interpreted as an integer)

$$\sum_1^0 = \{\text{recognizable sets}\}$$
= languages of the form $\{x | \exists y \ R(x,y)\}$

$$\Pi_1^0 = \{\text{complements of recognizable sets}\}$$
= languages of the form $\{x | \forall y \ R(x,y)\}$

$$\Delta_1^0 = \{\text{decidable sets}\}$$
= $\sum_1^0 \cap \Pi_1^0$

where $R$ is a decidable predicate

$$\sum_2^0 = \{\text{sets recognizable in some recognizable } B\}$$
= languages of the form $\{x | \exists y_1 \forall y_2 \ R(x,y_1,y_2)\}$

$$\Pi_2^0 = \{\text{complements of } \sum_2^0 \text{ sets}\}$$
= languages of the form $\{x | \forall y_1 \exists y_2 \ R(x,y_1,y_2)\}$

$$\Delta_2^0 = \sum_2^0 \cap \Pi_2^0$$

where $R$ is a decidable predicate

$$\sum_n^0 = \{\text{languages } \{x | \exists y_1 \forall y_2 \exists y_3 \ldots Qy_n \ R(x,y_1,\ldots,y_n)\}\}$$

$$\Pi_n^0 = \{\text{languages } \{x | \forall y_1 \exists y_2 \forall y_3 \ldots Qy_n \ R(x,y_1,\ldots,y_n)\}\}$$

$$\Delta_n^0 = \sum_n^0 \cap \Pi_n^0$$

where $R$ is a decidable predicate
Decidable predicate

\[ \Sigma_1^0 = \text{languages of the form } \{ x \mid \exists y R(x,y) \} \]

Theorem: \( A_{TM} \) is in \( \Sigma_1^0 \)

\[ A_{TM} = \{ (M,w) \mid \exists t [M \text{ accepts } w \text{ in } t \text{ steps}] \} \]

decidable predicate

\[ \Pi_1^0 = \text{languages of the form } \{ x \mid \forall y R(x,y) \} \]

Theorem: \( \text{EMPTY} = \{ M \mid L(M) = \emptyset \} \) is in \( \Pi_1^0 \)

\[ \text{EMPTY} = \{ M \mid \forall w \forall t [M \text{ doesn't accept } w \text{ in } t \text{ steps}] \} \]

two quantifiers??

**Pairing Functions**

Theorem. There is a 1-1 and onto computable function \(<,>: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*\) and computable functions \(\pi_1, \pi_2: \Sigma^* \rightarrow \Sigma^*\) such that

\[ z = <w, t> \text{ if and only if } \pi_1(z) = w \text{ and } \pi_2(z) = t \]

\[ \text{EMPTY} = \{ M \mid \forall w \forall t [M \text{ doesn't accept } w \text{ in } t \text{ steps}] \} \]

\[ \text{EMPTY} = \{ M \mid \forall z [M \text{ doesn't accept } \pi_1(z) \text{ in } \pi_2(z) \text{ steps}] \} \]
Theorem: TOTAL = \{ M | M halts on all inputs \} is in \( \Pi^0_2 \)
TOTAL = \{ M | \forall w \exists t [M halts on w in t steps] \}

Theorem: FIN = \{ M | L(M) is finite \} is in \( \Sigma^0_2 \)
FIN = \{ M | \exists n \forall w \forall t \left[ \text{Either } |w| < n, \text{ or } M \text{ doesn’t accept w in t steps} \right] \}

Limits on Oracle TMs
Theorem: The arithmetic hierarchy is strict: the nth level contains a language that isn’t in any of the levels below n.
Proof IDEA: Same idea as for SUPERHALT.
SUPERHALT^0 = HALT = \{ (M,x) | M halts on x \}.
SUPERHALT^1 = \{ (M,x) | M, with an oracle for the Halting Problem, halts on x \}
SUPERHALT^n = \{ (M,x) | M, with an oracle for \text{SUPERHALT}^{n-1}, halts on x \}