6.045

Lecture 11:
Fun With Undecidability!

Announcements

- **If MIT is denying you resources you need and you're running out of options, please contact me personally.**

- **Pset now due Monday March 30: we will release pset solutions immediately after that**
- **For now, midterm is still Thursday April 2**
- **Practice midterm + solutions out tonight!**
- **There is candy!**

The Acceptance Problem for TMs

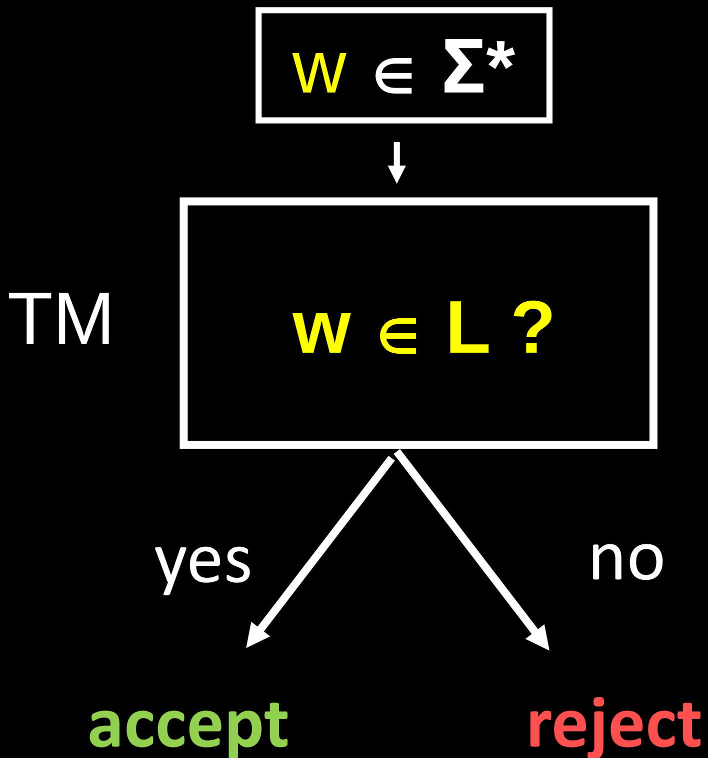
A_{TM} = { ⟨M, w⟩ | M is a TM that accepts string w }

Given: code of a Turing machine M and
an input w for that Turing machine,

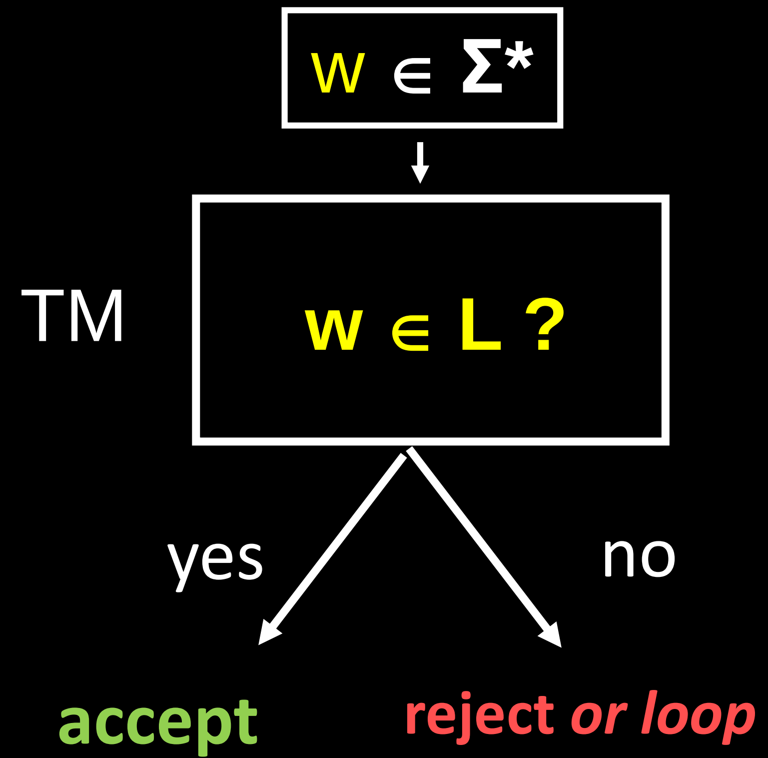
Decide: Does M accept w?

A_{TM} decidable ⇒ There is an algorithm ALG which,
given *any* code and input,
ALG determines in finite time
if the code will stop and accept the input

Theorem [Turing]:
A_{TM} is recognizable, but **NOT** decidable!



L is decidable



L is recognizable

Theorem: L is decidable iff both L and ¬L are recognizable

Theorem: L is **decidable** iff both L and $\neg$L are **recognizable**

Theorem: A_{TM} is recognizable but NOT decidable

Corollary: $\neg A_{TM}$ is not recognizable!

Theorem: $HALT_{TM}$ is not decidable

Reducing One Problem to Another

$f : \Sigma^* \rightarrow \Sigma^*$ 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 as $A \leq_m B$, if there is a computable $f : \Sigma^* \rightarrow \Sigma^*$ such that for every $w \in \Sigma^*$,

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

f is called a mapping reduction (or many-one reduction) from A to B

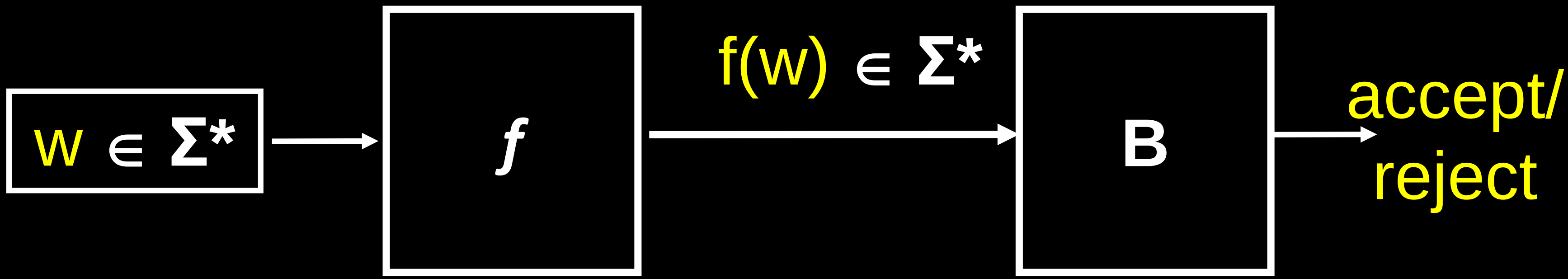
Theorem: If $A \leq_m B$ and B is decidable, then A is decidable

Corollary: If $A \leq_m B$ and A is undecidable, then B is undecidable

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

Corollary: If $A \leq_m B$ and A is unrecognizable, then B is unrecognizable

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



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

A recipe for proving undecidability!

To prove B is undecidable, find undecidable A and a mapping reduction from A to B.

A mapping reduction from A_{TM} to $HALT_{TM}$

Theorem: $A_{TM} \leq_m HALT_{TM}$

f(z) := Decode z into a pair $\langle M, w \rangle$. Write down the description of a TM M' with the spec:

"M'(w) = Run M on w.
If M accepts, then *accept,* else *loop forever*"
Output the encoding $\langle M', w \rangle$

Then, $z = \langle M, w \rangle \in A_{TM} \Leftrightarrow$ M accepts w
$\Leftrightarrow$ M' halts on w $\Leftrightarrow \langle M', w \rangle \in HALT_{TM}$

Corollary: $HALT_{TM}$ is undecidable

Theorem: $A_{TM} \leq_m HALT_{TM}$

Corollary: $\neg A_{TM} \leq_m \neg HALT_{TM}$

Corollary: $\neg HALT_{TM}$ is unrecognizable!

Proof: If $\neg HALT_{TM}$ were recognizable, then $\neg A_{TM}$ would also be recognizable, because $\neg A_{TM} \leq_m \neg HALT_{TM}$. But $\neg A_{TM}$ is not!

Question: $A_{TM} \leq_m \neg A_{TM}$?

Theorem: $HALT_{TM} \leq_m A_{TM}$

Theorem: $\mathrm{HALT}_{TM} \le_m A_{TM}$

Proof: Define a mapping reduction f:

f(z) := Decode z into a pair $\langle M, w \rangle$

Write down a TM M' with the specification:

"M'(w) = Run M on w. If M halts, *accept*"

Output $\langle M', w \rangle$

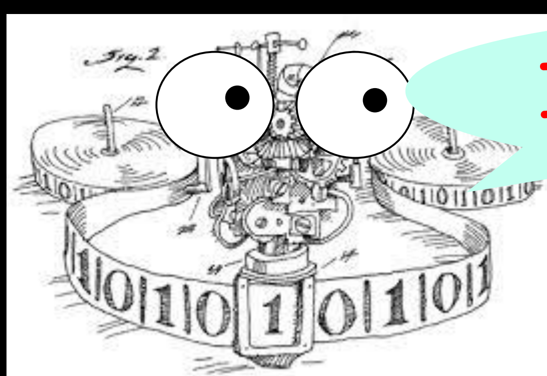
Observe $z=(M, w) \in \mathrm{HALT}_{TM} \iff (M', w) \in A_{TM}$

Corollary: $HALT_{TM} \equiv_m A_{TM}$

Yo, T.M.! I can give you the magical power to either compute the halting problem, or the acceptance problem. Which do you want?

Wow, hm, so hard to choose…

I can't decide!



Another Reduction Example

EQ_{DFA} = { (D_1,D_2) | D_1 and D_2 are DFAs, $L(D_1)=L(D_2)$}

EQ_{REGEX} = { (R_1,R_2) | R_1 and R_2 are regexps, $L(R_1)=L(R_2)$}

Theorem: $EQ_{REGEX} \leq_m EQ_{DFA}$

Proof: Mapping reduction *f* from EQ_{REGEX} to EQ_{DFA}:

f: On input z, decode z into a pair (R_1,R_2),
Convert R_1,R_2 into NFAs N_1,N_2,
Convert NFAs N_1,N_2 into DFAs D_1,D_2. Output (D_1,D_2)

Then, $(R_1,R_2) \in EQ_{REGEX} \Leftrightarrow L(D_1)=L(R_1)=L(R_2)=L(D_2)$
$\Leftrightarrow L(D_1)=L(D_2) \Leftrightarrow (D_1,D_2) \in EQ_{DFA}$

So *f* is a mapping reduction from EQ_{REGEX} to EQ_{DFA}

The Emptiness Problem for TMs

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

Given a program, does it reject or loop on all inputs?

Theorem: EMPTY_{TM} is *unrecognizable*

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

f(z) := Decode z into $\langle M, w \rangle$. Output code of the TM:

"M'(x) := if (x = w) then run M(w) and output answer, else reject"

Observe: EITHER $L(M') = \varnothing$ OR $L(M') = \{w\}$

$z=(M,w) \notin A_{\text{TM}} \Leftrightarrow$ M doesn't accept w

$\Leftrightarrow L(M') = \varnothing$

$\Leftrightarrow \langle M' \rangle \in \text{EMPTY}_{\text{TM}} \Leftrightarrow f(z) \in \text{EMPTY}_{\text{TM}}$

The Emptiness Problem for Other Models

$\text{EMPTY}_{\text{DFA}} = \{ \langle M \rangle \mid M \text{ is a DFA such that } L(M) = \varnothing \}$

Given a DFA, does it reject every input?

Theorem: $\text{EMPTY}_{\text{DFA}}$ is decidable

Why?

$\text{EMPTY}_{\text{NFA}} = \{ \langle M \rangle \mid M \text{ is a NFA such that } L(M) = \varnothing \}$

$\text{EMPTY}_{\text{REX}} = \{ \langle R \rangle \mid M \text{ is a regexp such that } L(M) = \varnothing \}$

The Equivalence Problem

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

Do two programs accept exactly the same strings?

Theorem: EQ_{TM} is *not recognizable*

Proof: Reduce $EMPTY_{TM}$ to EQ_{TM}

Let $M_\varnothing$ be a TM that always rejects immediately, so $L(M_\varnothing) = \varnothing$

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

Then $M \in EMPTY_{TM} \iff L(M) = L(M_\varnothing)$

$\iff \langle M, M_\varnothing \rangle \in EQ_{TM}$

Moral:

Analyzing Programs is Really, Really Hard for Programs to Do.

(Sometimes)

Computing With Oracles: Another Kind of Reduction

***We do not condone smoking. Don't do it. It's bad. Kthxbye**

Oracle Turing Machines

q_{YES}

Is (M, w) in A_{TM}?

yes!

A	N	P	U	T					

INFINITE TAPE

Now leaving reality for a moment….

Oracle Turing Machines

An **oracle Turing machine M** is equipped with a set $B \subseteq \Gamma^*$ and a special oracle tape, on which M may ask membership queries about B

Formally, M enters a special state $q_?$ to ask a query

and the TM receives a query answer in **one step**

[Formally, the transition function on $q_?$ is defined in terms of the *entire oracle tape*:

State $q_?$ changes to q_{YES}

if the string y written on the oracle tape is in B,

else $q_?$ changes to q_{NO}]

This notion makes sense even if B is not decidable!

How to Think about Oracles?

Think in terms of Turing Machine pseudocode!

An oracle Turing machine M with oracle B $\subseteq$ Γ* lets you include the following kind of if-then statement:

```
"if (z in B) then <do something>
              else <do something else>"
```

where `z` is some string defined earlier in pseudocode. We define the oracle TM to that it can always check the condition `(z in B)` in one step

This notion makes sense even if B is not decidable!

Deciding one problem with another

Definition: A is decidable with B
if there is an *oracle TM M with oracle B*
that accepts strings in A and rejects strings not in A

Language A "Turing-Reduces" to B

$$A \leq_T B$$

A_{TM} is decidable with $HALT_{TM}$ ($A_{TM} \leq_T HALT_{TM}$)

We can decide if M accepts w
using an ORACLE for the Halting Problem:

On input (M,w),
If (M,w) is in $HALT_{TM}$ then
run M(w) and output its answer.
else REJECT.

(This is exactly like our proof that
$HALT_{TM}$ is undecidable, from last lecture!)

$HALT_{TM}$ is decidable with A_{TM} ($HALT_{TM} \leq_T A_{TM}$)

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

1. If (M,w) is in A_{TM} then ACCEPT

2. Else, swap the accept and reject states of M to get a machine M′. If (M′,w) is in A_{TM} then ACCEPT

3. REJECT

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

Corollary: If $A \leq_T B$ and A is undecidable, then B is undecidable

Proof: Exactly the same proof as the one for mapping reductions!

If $A \leq_T B$ then there is a TM M with oracle B that decides A. If B is decidable, then we can replace every oracle call to B with a TM that decides B. Now M is a TM with no oracle!

$\leq_T$ versus $\leq_m$

Theorem: If $A \leq_m B$ then $A \leq_T B$

Proof (Sketch):

$A \leq_m B$ means there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every w,

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

To decide A on an input w with oracle B, just compute f(w), then call B on f(w) and return answer

Theorem: $\neg A_{TM} \leq_T A_{TM}$

D ($\langle M,w \rangle$): If ($\langle M,w \rangle$ in A_{TM}) then *reject* else *accept*

Theorem: $\neg A_{TM} \not\leq_m A_{TM}$

Limitations on Oracle TMs!

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

SUPERHALT = { (M,x) | TM M, with an oracle for the Halting Problem, halts on x}

We can use the original proof by diagonalization!

Assume H (with HALT oracle) decides SUPERHALT

Define D(X) := "if H(X,X) (with HALT oracle) accepts then LOOP, else ACCEPT."

(D uses a HALT oracle to simulate H)

But D(D) halts ⇔ H(D,D) accepts ⇔ D(D) loops…

(by assumption on H) (by def of D)

Limitations on Oracle TMs!

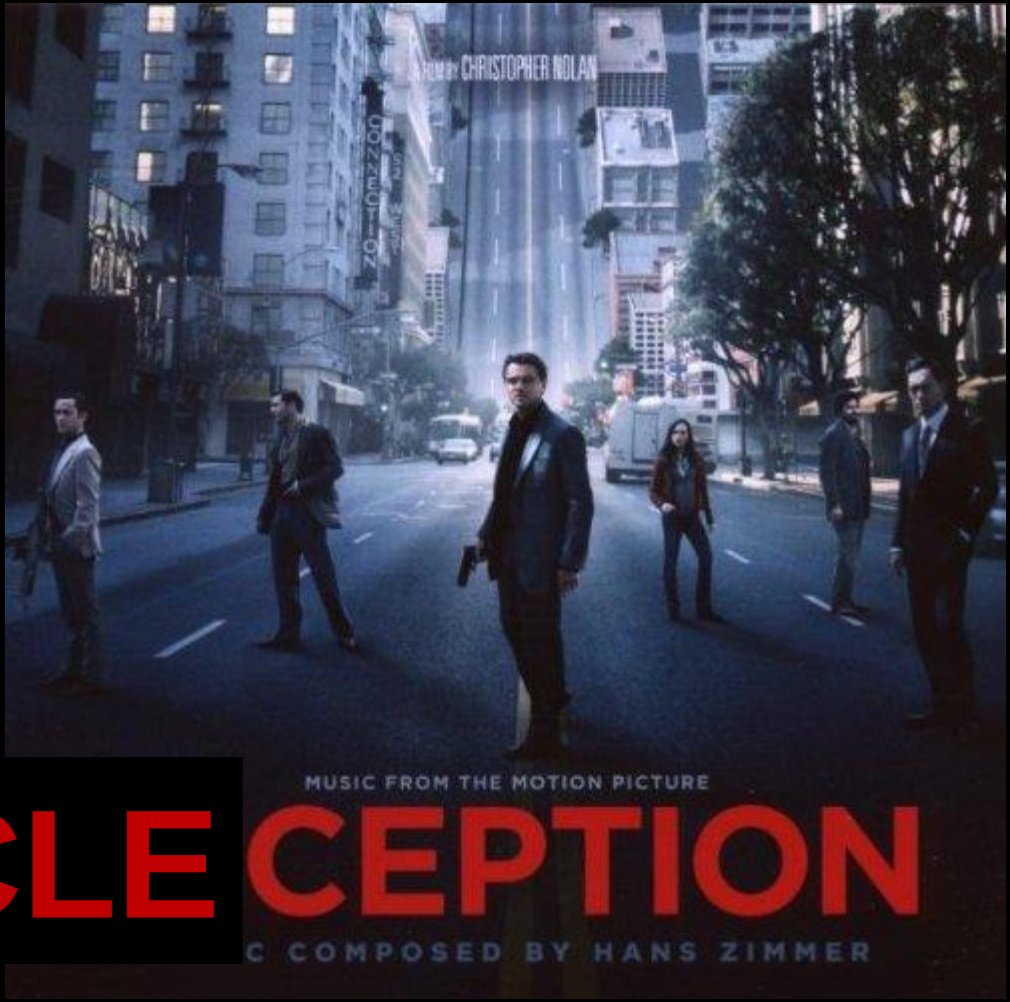
There is an *infinite hierarchy* of unsolvable problems!

Given ANY oracle A, there is always a <u>harder</u> problem that cannot be decided with that oracle A

$\text{SUPERHALT}^0 = \text{HALT} = \{ (M,x) \mid M \text{ halts on } x\}$.

$\text{SUPERHALT}^1 = \{ (M,x) \mid M, \text{ with an oracle for } \text{HALT}_{TM}, \text{ halts on } x\}$

$\text{SUPERHALT}^n = \{ (M,x) \mid M, \text{ with an oracle for } \text{SUPERHALT}^{n-1}, \text{ halts on } x\}$


CHRISTOPHER NOLAN
MUSIC FROM THE MOTION PICTURE
CEPTION
C COMPOSED BY HANS ZIMMER

ORACLE

A Puzzle About Oracles

Given three instances
$(M_1, w_1), (M_2, w_2), (M_3, w_3)$
of the Halting Problem,

It's easy to decide all three of them, using three oracle calls to HALT.

Can you decide $(M_i, w_i) \in$ HALT for all i, with only TWO oracle calls to HALT?

The Busy Beaver Function

How much work can a little TM do?

The Busy Beaver Function

Define a simple Turing machine to be one with input alphabet {1}, tape alphabet {1,□}, and a "halt state". Besides the "halt state", our TMs have n other states.

Define BB(n) to be the maximum number of steps taken on input ε by any n-state simple TM that halts.

BB(1) = 1: For a 1-state TM running on blank tape, it either halts in the first step, or it runs forever!

BB(2) = 6

BB(3) = 21

BB(4) = 107

BB(5) $\geq$ 47,176,870

BB(6) > $10^{36,534}$

BB(7) > $10^{10^{10^{10^{18,000,000}}}}$

The Busy Beaver Function

Theorem: BB(n) is not computable!

BB(n) grows so ridiculously fast that no computable function whatsoever *(no function you have ever seen)* can even *upper bound* it!!

First Idea: If you could compute BB(n), then you could solve the Halting problem for simple TMs running on blank tape!

Second Idea: It is impossible to decide that Halting problem

The Busy Beaver Function

Theorem: BB(n) is not computable!

Theorem: Assuming there is a TM computing BB(n), we can solve the Halting problem for simple TMs on ε.

Proof: Here's pseudocode for the Halting problem:

On the input ⟨M⟩ [code of a TM M]

Count the number of states in M, call it n

Compute t = BB(n) [in binary or unary]

Run M on blank tape for t steps.

If it halts, then accept. Otherwise, reject!

Theorem: There is NO computable function $f : \mathbb{N} \to \mathbb{N}$ such that for all n, $f(n) \geq$ BB(n).

The Busy Beaver Function

You can encode arbitrary math conjectures in simple TMs!

Theorem: There is a 1919-state simple TM that halts iff ZFC (set theory) is inconsistent!

There is a 744-state simple TM that halts iff the Riemann hypothesis is false.

There is a 43-state simple TM that halts iff Goldbach's conjecture is false

Good luck verifying if those halt!

Two Problems

Problem 1 Undecidable

{ (M, w) | M is a TM that on input w, tries to move its head past the left end of the tape at some point }

Problem 2 Decidable

{ (M, w) | M is a TM that on input w, moves its head left at some point}

Problem 1 **Undecidable**

L' = { (M, w) | M is a TM that on input w, tries to move its head past the left end of the tape }

Proof: **Reduce A_{TM} to L'**

On input **(M,w)**,
make a TM N that shifts w over one cell,
puts a special symbol # on the leftmost cell,
then simulates M(w) on its tape.
If M's head moves to the cell with # but has *not yet accepted*, N moves the head back to the right.
If M accepts, N tries to move its head past the #.

(M,w) is in A_{TM} if and only if (N,w) is in L'

Problem 2 **Decidable**

{ (M, w) | M is a TM that on input w, moves its head left at some point}

On input (M,w), run M on w for
$|Q| + |w| + 1$ steps,
where $|Q|$ = number of states of M

Accept If M's head moved left at all
Reject Otherwise

(Why does this work?)

Thank you all …

… see you in June, hopefully?

Be safe!