Halting, Again

At long last we are able to prove the undecidability of

$$A_{\mathsf{TM}} = \{\langle M, w\rangle | M \text{ is a TM that accepts } w\}.$$

Proof: By contradiction. Suppose TM H is a decider for for A_{TM}.

On input $\langle M, w\rangle$, where M is a TM and w is a string, H halts and accepts if and only if M accepts w. Furthermore, H halts and rejects if M fails to accept w.

Halting

On input $\langle M, w \rangle$, where M is a TM and w is a string, H halts and accepts if and only if M accepts w.
Furthermore, H halts and rejects if M fails to accept w.

$$H(\langle M, w \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

Halting

Now we construct a new TM D with H as a subroutine.

D does the following

- calls H to determine what M does when the input to M is its own description.
- When D determines this, it does the opposite.

It rejects if M accepts, and accepts if M rejects.

Halting

More precisely, D does the following:

- Run H on input $\langle M, \langle M \rangle \rangle$.
- Output the opposite of what H outputs:
  - If H accepts, *reject*, and
  - If H rejects, *accept*.

Halting

Don't be confused by the notion of running a machine on its own description!

Actually, you should get used to it.

- Notion of self-reference comes up again and again.
- Read "Gödel, Escher, Bach" an Eternal Golden Braid, by Douglas Hofstadter.
- This notion of self-reference is the basic idea behind Gödel's revolutionary result.

Compilers do this all the time . . . .

Halting

In summary,

$$D(\langle M \rangle) = \begin{cases} \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \\ \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \end{cases}$$

The Punch Line

What happens if we run D on its own description?

$$D(\langle D \rangle) = \begin{cases} \textit{reject} & \text{if } D \text{ accepts } \langle D \rangle \\ \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \end{cases}$$

Uh, oh.

Once Again

- Assume that TM H decides A_{TM}.
- Then use H to build a TM D, that when given M, accepts exactly what M does not accept.
- Run D on its own description.
- D does:
  - 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 $\langle D\rangle$.

  Last step does not compute!

Diagonalization

Claim: This proof is diagonalization in transparent disguise.

Let's make a table.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\ldots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	
M_3					
M_4	*accept*	*accept*			
$\vdots$					

Entry is *accept* if M_i accepts $\langle M_i \rangle$, blank if M_i rejects or loops.

Diagonalization

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\ldots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	
M_3					
M_4	*accept*	*accept*			
$\vdots$					

Run H on on corresponding inputs.

Entry (i,j) states whether H accepts $\langle M_i, \langle M_j \rangle \rangle$.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\ldots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$		$\vdots$	$\vdots$	$\vdots$	

Diagonalization

Now we add D to the table.

- H is a TM and so is D.
- It occurs on the list $M_1, M_2, \ldots$ of all TMs.
- D computes the opposite of the diagonal entries.
- At its own diagonal entry, it computes its own opposite!

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$		$\ldots$
M_1	$\boxed{\textit{accept}}$	*reject*	*accept*		
M_2	*accept*	$\boxed{\textit{accept}}$	*accept*		
M_3	*reject*	*reject*	$\boxed{\textit{reject}}$		
M_4	*accept*	*accept*	*reject*		
$\vdots$		$\vdots$		$\ddots$	
D	$\boxed{\textit{reject}}$	$\boxed{\textit{reject}}$	$\boxed{\textit{accept}}$		$\boxed{???}$
$\vdots$		$\vdots$		$\ldots$	

A Non-enumerable Language

We have already seen a non-decidable language: A_{TM}.

Can we do better (i.e., worse)?

Mais, oui!

We now display a language that isn't even enumerable . . . .

A Non-enumerable Language

Theorem: If L and $\overline{L}$ are both enumerable, then L is decidable.

Corollary: If L is not decidable, then either L or $\overline{L}$ is not enumerable.

Definition: A language is co-enumerable if it is the complement of an enumerable language.

Theorem

Theorem: A language is decidable if and only if it is both enumerable and co-enumerable.

Proof: We must prove two directions:

- If A is decidable, then both A and $\overline{A}$ are enumerable.
- If A and $\overline{A}$ are both enumerable, then A is decidable,

One Direction

Claim: If A is decidable, then both A and $\overline{A}$ are enumerable.

Proof: Pop quiz!

Other Direction

Claim: If A and $\overline{A}$ are both enumerable, then A is decidable,

- Let M_1 be the acceptor for A, and
- M_2 the acceptor for $\overline{A}$.

$M =$ On input w

1. Run both M_1 and M_2 in parallel.
2. If M_1 accepts, *accept*; if M_2 accepts, *reject*.

Parallel?

Question: What does it mean to run M_1 and M_2 in parallel?

- M has two tapes
- M alternates taking steps between M_1 and M_2.

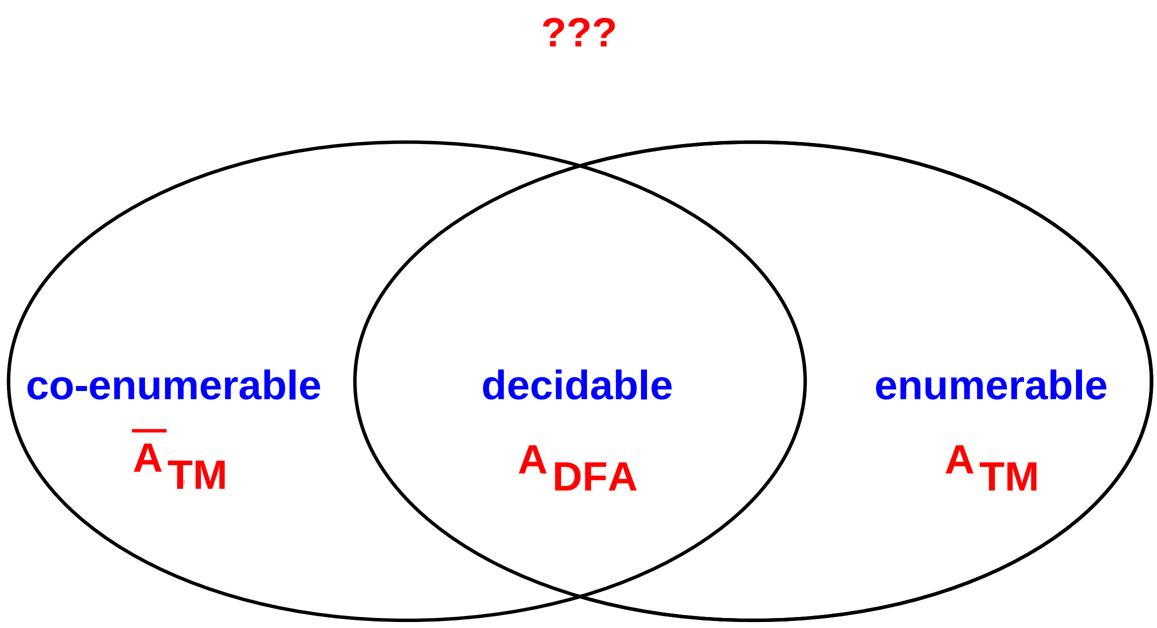
Claim

We claim that M decides A.

- Every string is in A or $\overline{A}$.
- Either M_1 or M_2 accepts input w.
- Because M halts whenever M_1 or M_2 accepts M always halts, and is a decider.
- Moreover, M accepts strings in A and rejects strings in $\overline{A}$.

Therefore, M decides A, and A is decidable.

Languages



Question: Are there any languages in the area marked ????

Answer: Yes, lots. Why?