Diagonalization and the Halting Problem

We use diagonalization to prove that some languages are hard.

The Goal Again

Is there for every problem an algorithm: that is, some procedure that gives the right answer, is clear and completely described, and is guaranteed to terminate? Is there a TM for every language?

Do Not Read This

Example: Self-Denial

Define the self-denying machines by

$$S_{tm} = \{ \langle M \rangle : M \text{ is TM not accepting } \langle M \rangle \}$$

Note that S_{tm} is a valid language. If we were omniscient, we could see whether w is in S_{tm} by parsing w as a TM, and then checking whether that TM accepted w.

The Language S_{tm} is Not R.e.

Self-denial. *S_{tm} is not r.e.*

Proof by Contradiction. Suppose some TM accepts this language: call it M'.

Does M' accept string $\langle M' \rangle$? Well, $\langle M' \rangle$ is in S_{tm} if and only if M' does not accept $\langle M' \rangle$. That is, M' accepts $\langle M' \rangle$ if and only if M' does not accept $\langle M' \rangle$.

A contradiction. Since the logic is correct, the problem is the supposition: M' does not exist.

Diagonalization in TMs

Here’s the diagonalization argument in TMs. Recall that we encode a TM in binary; thus we can list them in lexicographic (dictionary) order.

Diagonalization in TMs

Create a table with each row labeled by a TM and each column labeled by a string that encodes a TM. The entries say whether TM M_i accepts the string $\langle M_j \rangle$.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\ldots$
M_0	*Acc*	*Not*	*Not*	
M_1	*Not*	*Not*	*Not*	
M_2	*Not*	*Acc*	*Acc*	
$\vdots$				

Diagonalization Produces Non-R.e. Language

Now apply diagonalization; that is, go down the diagonal and change every *Acc* to a *Not* and vice versa. If one writes down all those strings that now have an *Acc* on diagonal, one has a language. This language is. . . S_{tm}, the self-denying machines.

But this diagonal is different from every row. That is, this diagonal behaves differently from every TM. That is, the language is not the language of any TM.

The Acceptance Problem

It would be useful to have an algorithm that takes as input a TM and string, and tells one whether the TM will halt on that input. Unfortunately, such an algorithm does not exist.

The Acceptance Problem

Define $A_{tm} = \{ \langle M, w \rangle : M$ is TM that accepts $w \}$.

It is easy to show by simulation that:

A_{tm} *is r.e.*

The Acceptance Problem is Undecidable

Theorem. *A_{tm} is not recursive. That is, the acceptance problem is undecidable.*

The theorem says that one cannot build a TM that will always halt and tell one whether a given machine accepts a given word or not. Using Church's thesis, this means that there is no algorithm that one can use to test beforehand whether a given machine/program on given input will halt.

We give two proofs.

Proof by Diagonalization

To prove that A_{tm} is undecidable, we build a TM D that *for every i* does the opposite of i^{th} machine on input $\langle M_i \rangle$. So when we try to find D on our list of TMs, it is not there!

Suppose there were machine H that on *every* input $\langle M, w \rangle$ would tell one whether or not M accepted w. Then, build a new TM $D \ldots$

Building a New TM

Build a new TM D that does the following:

```
$D$: On input $w$
1. Determine the TM $S$ that $w$ encodes.
2. Run $H$ on $\langle S, w \rangle$.
3. If $H$ accepts then reject; else accept.
```

A Contradiction

But wait. What happens if input is description of D, say $w' = \langle D \rangle$ (not that D notices)?

Well, D writes $\langle D, w' \rangle$ on the tape, and feeds to H. If H says accept, then D rejects, and vice versa. That is, if H claims that D accepts $w' = \langle D \rangle$, then D rejects w'. If H says D rejects $w' = \langle D \rangle$, then D accepts w'. Huh?

This is a contradiction. Everything we did was fine except possibly that H exists. Conclusion: H does not exist.

Alternative Proof

Suppose there were TM H that decided A_{tm}. Then one could use H as a subroutine to decide the language S_{tm}. But that language is not recursive. Contradiction.

The Halting Problem is Undecidable

It is no easier if all you want to know is whether the program will halt or not—called the ***halting problem***.

For, one can easily adjust a TM so that instead of entering h_r to reject, it enters a state that keeps its head moving to the right forever. Solving the halting problem is thus just as hard as solving the acceptance problem. That is, the halting problem is undecidable.

Summary

The language S_{tm} (self-denial) is not r.e. The acceptance language A_{tm} and the halting problem are r.e. but not recursive. The proof uses self-reference.