Languages over {0,1}

Lemma: There are uncountably many languages over {0,1}

Proof idea: we represent a language over {0,1} as an infinite sequence of 0's and 1's. The ith entry tells whether the ith string is in the language or not. We diagonalize over strings.

Proof: Suppose there were a 1-1 corresp. from N.

1 0110110.....
2 1001010....
3 0000111.....
:

We define a new language over {0,1} as follows:

Languages over {0,1}

1 0110110.....
2 1001010....
3 0000111.....
:

The ith bit of the language is defined as follows:

0 if the ith bit of the ith language in list is 1

1 if the ith bit of the ith language in list is 0

This language is not on the list, as it differs from every language on the list in at least 1 bit.

Contradiction.

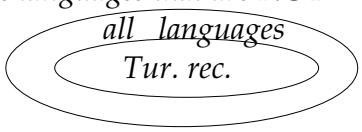
This proves that the languages over {0,1} are uncountable.

TM's over {0,1}

Lemma: There are countably many TM's over {0,1}

Proof: we encode TM M as a string <M> over {0,1}.
We can enumerate the TM's as follows:
Enumerate strings over {0,1}. If a generated string is not an encoding of a TM, omit it and go on to the next one. Eventually, get the next.

Countable no. of TM's *Uncountable no. of languages*
Therefore, there are some languages that are not accepted by any TM.
Anotherwords, there are languages that are NOT Turing-recognizable.



The Halting Problem is Undecidable

Th: A_{TM}={<M,w> | M is a TM accepting w} is undec.

Proof: Assume A_{TM} is decidable. Let H be a decider for it. We construct a new TM D with H as subr.

(Idea: D will call H to determine if M accepts <M>. once H returns, D will do the opposite.)

D: on input <M>

1. **Run H on input <M>.**
2. **If H accepts, D rejects; if H rejects, D accepts.**

Now consider input <D> to D.

D(<D>) accepts if D does not accept <D>
rejects if D does accept <D>

Contradiction.

Diagonalization in the Proof

	M1	M2	M3	....
<M1>	a	a		
<M2>		r		
<M3>	r		a	
:				

Empty slots are when don't halt. H fills in the empty slots with r's.

	M1	M2	M3	....
<M1>	a	a	r	
<M2>	r	r	r	
<M3>	r	r	a	
:				

Running D computes the opposite of H along diag. Get contradiction from D's entry in the table.

5

A Turing-Unrecognizable Language

A_{TM} *won't work, since we showed that it IS Turing-recog.*

Th. A language L is decidable iff.

both L and its complement $\overline{L}$ are Turing-recognizable.

→ : If L is decidable, it is certainly Tur.- rec.

If L is decidable, we know that $\overline{L}$ is decidable, and therefore Turing-rec.

← : If L and $\overline{L}$ are Tur.-rec., let M1 be recognizer for L, and M2 for $\overline{L}$. M will decide L as follows:

M on input w:

1. Run M1 and M2 on w, on separate track, one step at a time on each.
2. If M1 accepts, M accepts. If M2 accepts, M rejects.

6

Universal TM U is not a decider

U has no way to determine if M halts on input w.

Lemma: A_{TM} is Turing-recognizable.

Proof: U is the recognizer.

Halting Problem: A_{HALT} = {<B,w> | B halts on input w}

A_{HALT} is decidable ⟷ A_{TM} is decidable

We show A_{TM} is undecidable (then A_{HALT})

Doesn't mean that we CAN'T determine for a particular <B,w> whether B halts (accepts) on input w.

Does mean that there is no algorithm that works for all input <B,w>. Gives fundamental limitation of Algorithms/TM.

7

A Turing-Unrecognizable Language

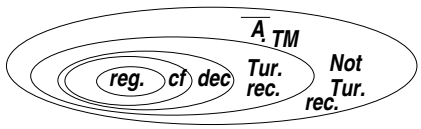
M decides L: Every w in in L or $\overline{L}$. Either M1 or M2 accepts w. But M halts when M1 or M2 accept. So M always halts, and so is a decider.
Further, M accepts all strings accepted by M1 (L)

Cor.: $\overline{A}_{TM}$ is not Turing-recognizable.

Proof: A_{TM} is Turing-recognizable. If $\overline{A}_{TM}$ were,

then by previous Th. A_{TM} would be decidable.

But we've shown its not decidable, CONTR.



8

Other Undecidable Problems

Th. $HALT_{TM}$ = {<M,w> | M is a TM that halts on w} is undecidable.

Proof idea: by contradiction. We assume that $HALT_{TM}$ is decidable, and show then that A_{TM} would be decidable.

Proof: If $HALT_{TM}$ were decidable, then there is a TM H that decides it. We construct TM S to decide A_{TM} as follows:

S on input <M,w>:

1. **Run H on <M,w>.** *(Note H always halts)*
2. **If H rejects, then S rejects.** *(M doesn't halt on w)*
3. **If H accepts, then simulate M on w until it halts.**
4. **if M accepts then S accepts; M rejects, S rej.**