Theorem

Let S be an infinite countable set. Then 2^S (powerset of S) is uncountable.

Proof: we will prove this using diagonalization.

Let $S = \{s_1, s_2, s_3, \ldots\}$

I need to show $P(S)\,(2^S)$ is not countable.

Assume 2^S is countable.

We encode each element $t \in 2^S$ as an infinite sequence of 0's and 1's, with 1 at the position if and only if $s_i \in t$.

$t_1, t_2, t_3, \ldots, \in 2^S$

$t_1 = 1\,0\,0\,1\,0\,1 \ldots \quad \{s_1, s_4, s_6, \ldots$

$t_2 = 1\,0\,0\,0\,0\,0 \ldots \quad \{s_1, \ldots$

$t_3 = 1\,0\,1\,1\,0\,0 \ldots \quad \{s_1, s_3, s_4$

Since we assumed that 2^S is countable, I can map the set of natural numbers to 2^S using a bijection

i	$f(i)$
1	$\underline{1}\,0\,0\,1\,1\,0\,0\,-\,-\,-\,-$
2	$0\,\underline{1}\,0\,1\,1\,0\,1\,-\,-\,-\,-$
3	$1\,1\,\underline{1}\,0\,0\,1\,0\,-\,-\,-\,-$

I will create element t_k where each entry of t_k is constructed by taking the complement of the diagonal in this table.

$t_k = 0\,0\,0\,-\,-\,-\,-\,-$

$\therefore$ t_k does not have a mapping in this bijection

$\therefore$ 2^S is not countable.

We can use this idea to show that there are some languages not Turing-Recognizable

Theorem: Some languages are not Turing-Recognizable.

Proof Idea:

Set of all possible turing machines are countable.

- TM M can be encoded into a string $\langle M \rangle$.
- This string can be further encoded into a binary string.
- The set of all binary strings is countable
  - $\Sigma = \{0, 1\}$
  - Σ^* is countable
    - Simply list the strings in Σ^* in lexicographical order
    - $\{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\ldots\}$

- you can assign a unique natural number to each string in this set.

- Observe that each TM recognizes a language $L \subseteq \Sigma^*$
- A language is a set of strings.
- $L \in P(\Sigma^*)$
- Note that Σ^* is contably infinite.
- We showed that $P(\Sigma^*)$ is not countable.
- Therefore, we do not have enough turing machines to recognize all the possible languages.

Undecidability

We will consider several languages.

$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

$E_{TM} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

we will look at A_{TM}

let us create an algorithm for A_{TM}

1. Simulate M on w
2. If M stops at the accept state, accept;
   If M stops at a reject state, reject

//otherwise loop, looping is hard to distinguish from taking a long time to run.

This is a recognizer for A_{TM}

- A_{TM} is not the problem of asking you to write a program that accepts w.

- Suppose you have a program that someone else wrote, which you do not know how it works. Now you need to check whether this program works as intended. For example you need to check whether specific input is accepted by the program. what you can do feed the program and the input to A_{TM}, and it will say whether that specific input is accepted by the program or not

Theorem 4.11 A_{TM} is undecidable.

Proof: We prove this theorem by contradiction.

Assume A_{TM} is decidable.

Let TM H is a decider for the A_{TM} language.

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

Then I am going to construct TM D, where it will take <K>; K is a TM, as the input.

D = "on <K>, where K is a TM

1. Run H on <K, <K>>
2. output the opposite of what H outputs on <K, <K>>

Now, I am going to feed <D> to D.

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

This is a contradiction.

$\therefore A_{TM}$ is not decidable.