1. Prove that the set of all real numbers is not countable.

We can prove this by diagonalization.

Let there is a set L which is a set of all real numbers in $(n, n+1)$ ($\forall n \in \mathbb{Z}$).

Assume L is countable.

We will enumerate the element of L.

$$
\begin{array}{ll}
l_1 & n.\,a_{11}\ a_{12}\ a_{13}\ \cdots \\
l_2 & n.\,a_{21}\ a_{22}\ a_{23}\ \cdots \\
l_3 & n.\,a_{31}\ a_{32}\ a_{33}\ \cdots \\
 & \vdots \\
l_i & n.\,a_{i1}\ a_{i2}\ a_{i3}\ \cdots \\
 & \vdots
\end{array}
$$

$\{a_{ij} : 0 \le a_{ij} \le 9,\ i, j \in N.\ a_{ij}$ is an integer.$\}$

We take the elements in the main diagnol, and subtract each element from 9 in this table. So, $l_{new} = n.(9-a_{11})(9-a_{22})\cdots(9-a_{ii})\cdots$.

The new number differs from any other entry in the enumeration. ($\because 9-a_{ii} \neq a_{ij}$ ($j=1,2,3,\cdots$))

So, L is uncountable by contradiction.

Since L is uncountable, and n can be any integer, the set of all real numbers is $L \cup \{n : \forall n \in \mathbb{Z}\}$ so that the set of all real numbers is not countable. ■

2. If a language is not recursively enumerable, its complement cannot be recursive.

We will prove this by contraposition:

$\bar{L}$ can be recursive $\longrightarrow$ L is recursively enumerable.

We will set that $\bar{L} = A$, $L = \bar{A}$.

So, A can be recursive $\longrightarrow$ $\bar{A}$ is recursively enumerable.

By following Theorem 11.4 in the textbook, assume A is recursive.

Then there is a membership algorithm for A.

The membership algorithm for A becomes a membership algorithm for $\bar{A}$ by complementing its conclusion.

Thus, $\bar{A}$ is recursive.

Since recursive languages are subset of recursively enumerable languages, L, which is $\bar{A}$, is recursively enumerable.

Because the contraposition of the statement is true, if L is not recursively enumerable, $\bar{L}$ cannot be recursive.

We're done by contraposition. ■

3. Given Turing Machines M_1 and M_2, we have to determine whether or not M_1 and M_2 accept L.

We will modify M_1 and M_2 to produce M_1' and M_2' which halt on w if $w \in L$, and does not halt on w if $w \notin L$.

We will construct a new machine $\hat{M}$ which halts on state q if and only if both M_1' and M_2' halt on w.

We can apply the state-entry algorithm A.
The algorithm A can be a solution of halting problem.
Since halting problem is undecidable, there is no algorithm for determining whether or not $L(M_1) = L(M_2)$.

(+ The halt of M_1' and M_2' is halting problem which is undecidable so that the halt of $\hat{M}$ is undecidable.)

Thus, the problem of determining whether any two Turing machines accept the same language is undecidable.

4.

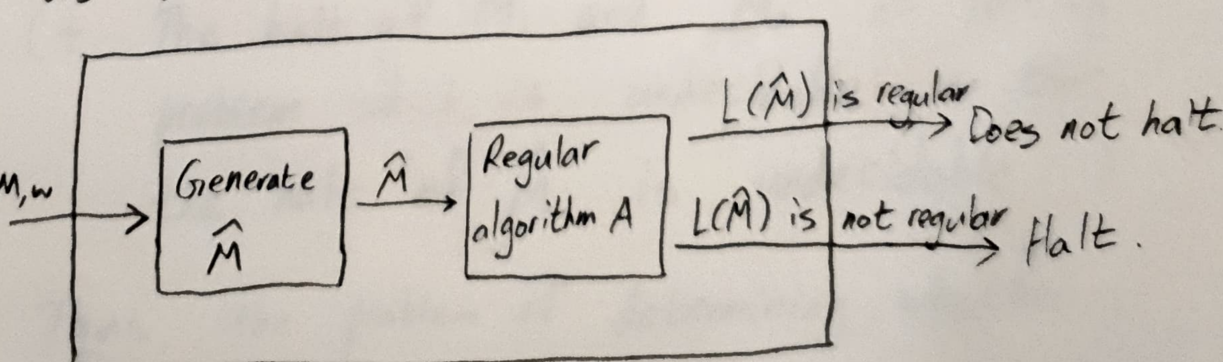
From M, we construct another Turing machine $\hat{M}$ that does the following.

1. the halting states of M are changed so that if any one is reached, all inputs is accepted by $\hat{M}$.

2. the original machine is modified so that $\hat{M}$ first generates w on its tape and performs the same computation as M, using the newly created w and some otherwise unused space.

So, if M halts on w, $\hat{M}$ will reach a final state for all input. If $\hat{M}$ does not halt on w, $\hat{M}$ will not halt either and so will accept nothing.

If we assume the existance of an algorithm A which tells us whether or not $L(\hat{M})$ is regular, we can construct the solution to the halting problem as below.



Thus, there is no algorithm for deciding whether or not $L(M)$ is regular.

$\therefore$ "$L(M)$ is regular" is undecidable.