Reducibility
There are two main techniques for showing that problems are undecidable: diagonalization and reduction
We say that a problem A is reduced to a problem B if the decidability of A follows from the decidability of B.

Linz, 6th, page 315
Problem $A$ is reduced to problem $B$

If we can solve problem $B$ then we can solve problem $A$
A is reducible to B if we can use B as a subroutine to solve A.

Use a halting TM for B in the construction of a halting TM which decides problem A.
Problem \( A \) is reduced to problem \( B \)

If \( B \) is decidable then \( A \) is decidable

If \( A \) is undecidable then \( B \) is undecidable
Example: the halting problem is reduced to the state-entry problem.

So, if the halting problem is undecidable, then the state-entry problem is undecidable.
The state-entry problem

Inputs:  
• Turing Machine $M$
• State $q$
• String $w$

Question: Does $M$ enter state $q$ on input $w$?
Theorem: The state-entry problem is undecidable

Proof: Reduce the halting problem to the state-entry problem.
Suppose we have an algorithm (Turing Machine) that solves the state-entry problem.

We will construct an algorithm that solves the halting problem.
Assume we have the state-entry algorithm:

Algorithm for state-entry problem

M \rightarrow \text{Algorithm for state-entry problem} \rightarrow \begin{cases} \text{YES} & M \text{ enters } q \\ \text{NO} & M \text{ doesn't enter } q \end{cases}

w \rightarrow \text{Algorithm for state-entry problem} \rightarrow \begin{cases} M \text{ enters } q \\ M \text{ doesn't enter } q \end{cases}

q \rightarrow \text{Algorithm for state-entry problem} \rightarrow \begin{cases} \text{YES} & M \text{ enters } q \\ \text{NO} & M \text{ doesn't enter } q \end{cases}
We want to design the halting algorithm:
Modify any machine $M$ to construct $M'$:

- Add new state $q$
- From any halting state add transitions to $q$
$M$ halts if and only if

$M'$ halts on state $q$
Algorithm for halting problem:

Inputs: machine $M$ and string $w$

1. Construct machine $M'$ with state $q$

2. Run algorithm for state-entry problem with inputs: $M'$, $q$, $w$
Halting problem algorithm

Generate

$M'$

$M$

$w$

$q$

State-entry algorithm

YES

YES

NO

NO
We reduced the halting problem to the state-entry problem.

Since the halting problem is undecidable, it must be that the state-entry problem is also undecidable.

END OF PROOF
Another example:

the halting problem

is reduced to

the blank-tape halting problem
The blank-tape halting problem

Input: Turing Machine $M$

Question: Does $M$ halt when started with a blank tape?
Theorem:
The blank-tape halting problem is undecidable

Proof: Reduce the halting problem to the blank-tape halting problem
Suppose we have an algorithm for the blank-tape halting problem.

We will construct an algorithm for the halting problem.
Assume we have the blank-tape halting algorithm/machine. It looks like this:

\[ M \xrightarrow{\text{Algorithm for blank-tape halting problem}} \]

- YES: \( M \) halts on blank tape
- NO: \( M \) doesn’t halt on blank tape
We want to design a machine that decides the halting problem. The machine looks like this:

Algorithm for halting problem

YES

M halts on w

NO

M doesn’t halt on w
Construct a new machine $M_w$

- On blank tape writes $w$
- Then continues execution like $M$

\[
\begin{array}{c}
\text{step 1} \\
\text{if blank tape} \\
\text{then write } w
\end{array}
\quad
\begin{array}{c}
\text{step 2} \\
\text{execute } M \\
\text{with input } w
\end{array}
\]
$M$ halts on input string $w$

if and only if

$M_w$ halts when started with blank tape
Algorithm for halting problem:

Inputs: machine $M$ and string $w$

1. Construct $M_w$

2. Run algorithm for blank-tape halting problem with input $M_w$
Halting problem algorithm

\[ M \xrightarrow{\mbox{Generate}} M_w \xrightarrow{M_w} \text{blank-tape halting algorithm} \]

Output:
- YES
- NO
We reduced the halting problem to the blank-tape halting problem.

Since the halting problem is undecidable, the blank-tape halting problem is also undecidable.

END OF PROOF
Summary of Undecidable Problems

Halting Problem:

Does machine $M$ halt on input $w$?

Membership problem:

Does machine $M$ accept string $w$?

In other words: Is a string $w$ member of a recursively enumerable language $L$?
Blank-tape halting problem:
Does machine $M$ halt when starting on blank tape?

State-entry Problem:
Does machine $M$ enter state $q$ on input $w$?
Uncomputable Functions
A function is uncomputable if it cannot be computed for all of its domain.
An uncomputable function:

$$f(n) = \begin{cases} \text{maximum number of moves until} \\ \text{any Turing machine with } n \text{ states} \\ \text{halts when started with the blank tape} \end{cases}$$
Theorem: Function $f(n)$ is uncomputable

Proof: Assume for contradiction that $f(n)$ is computable.

Then the blank-tape halting problem is decidable.
Algorithm for blank-tape halting problem:

Input: machine $M$

1. Count states of $M : m$

2. Compute $f(m)$

3. Simulate $M$ for $f(m)$ steps starting with empty tape

If $M$ halts then return YES
otherwise return NO
Therefore, the blank-tape halting problem is decidable

However, the blank-tape halting problem is undecidable

Contradiction!!!
Therefore, function $f(n)$ is uncomputable.

END OF PROOF
Rice’s Theorem
Definition:

Non-trivial properties of recursively enumerable languages:

any property possessed by some (not all) recursively enumerable languages
Some non-trivial properties of recursively enumerable languages:

- $L$ is empty
- $L$ is finite
- $L$ contains two different strings of the same length
Rice’s Theorem:

Any non-trivial property of a recursively enumerable language is undecidable
In exactly the same manner, we can substitute other questions such as “Does $L(M)$ contain any string of length five?” or “Is $L(M)$ regular?” without affecting the argument essentially. These questions, as well as similar questions, are all undecidable. A general result fromalizing this is known as Rice’s theorem.
Rice's Theorem

This theorem states that any nontrivial property of a recursively enumerable language is undecidable. The adjective "nontrivial" refers to a property possessed by some but not all recursively enumerable languages. A precise statement and a proof of Rice's theory can be found in Hopcroft and Ullman (1979).

Linz, 6th, pages 321-322
A property of the RE languages is simply a set of RE languages. Thus, the property of being context-free is formally the set of all CFL’s. The property of being empty is the set \{\emptyset\} consisting of only the empty language.

Theorem 9.11: (Rice’s Theorem) Every nontrivial property of the RE languages is undecidable.

Hopcroft, Motwani, Ullman, 3rd, pages 397-398
We will prove some non-trivial properties without using Rice’s theorem
Theorem:

For any recursively enumerable language $L$, it is undecidable to determine whether $L$ is empty.

Proof:

We will reduce the membership problem to this problem.
Let $M$ be the machine that accepts $L$

$L(M) = L$

Assume we have the empty language algorithm:

Algorithm for empty language problem

YES $\rightarrow L(M)$ empty

NO $\rightarrow L(M)$ not empty
We will design the membership algorithm:

Algorithm for membership problem

- If $M$ accepts $w$, then $w$ is in the language of $M$.
- If $M$ rejects $w$, then $w$ is not in the language of $M$. 
First construct machine $M_w$: 

When $M$ enters a final state, compare original input string with $w$. 

Accept if original input is the same with $w$. 


\( w \in L \)

if and only if

\( L(M_w) \) is not empty

\[ L(M_w) = \{ w \} \]
Algorithm for membership problem:

Inputs: machine $M$ and string $w$

1. Construct $M_w$

2. Determine if $L(M_w)$ is empty

YES: then $w \notin L(M)$

NO: then $w \in L(M)$
Membership algorithm

$M \rightarrow$ construct $M_w$

$w \rightarrow$ construct $M_w$

$L(M_w) \rightarrow$ Check if is empty

YES $\rightarrow$ NO

NO $\rightarrow$ YES

END OF PROOF