Prove that the following languages are undecidable.

See outline of how to solve such problems in the original problem set.

1. **AcceptIllini** := \{⟨M⟩ | M accepts the string **ILLINI**\}

**Solution:**

For the sake of argument, suppose there is an algorithm **DecideAcceptIllini** that correctly decides the language **AcceptIllini**. Then we can solve the halting problem as follows:

```plaintext
DecideHalt((M, w)):
   Encode the following Turing machine M':
      M'(x):
         run M on input w
         return True
   if DecideAcceptIllini(⟨M'⟩)
      return True
   else
      return False
```

We prove this reduction correct as follows:

\[\iff\]

- Suppose M halts on input w.
  - Then M' accepts *every* input string x.
  - In particular, M' accepts the string **ILLINI**.
  - So **DecideAcceptIllini** accepts the encoding ⟨M'⟩.
  - So **DecideHalt** correctly accepts the encoding ⟨M, w⟩.

- Suppose M does not halt on input w.
  - Then M' diverges on *every* input string x.
  - In particular, M' does not accept the string **ILLINI**.
  - So **DecideAcceptIllini** rejects the encoding ⟨M'⟩.
  - So **DecideHalt** correctly rejects the encoding ⟨M, w⟩.

In both cases, **DecideHalt** is correct. But that’s impossible, because **Halt** is undecidable. We conclude that the algorithm **DecideAcceptIllini** does not exist.

As usual for undecidability proofs, this proof invokes *four* distinct Turing machines:

- The hypothetical algorithm **DecideAcceptIllini**.
- The new algorithm **DecideHalt** that we construct in the solution.
- The arbitrary machine M whose encoding is part of the input to **DecideHalt**.
- The special machine M' whose encoding **DecideHalt** constructs (from the encoding of M and w) and then passes to **DecideAcceptIllini**.
AcceptThree := \{ (M) \mid M \text{ accepts exactly three strings} \}

Solution:
For the sake of argument, suppose there is an algorithm DecideAcceptThree that correctly decides the language AcceptThree. Then we can solve the halting problem as follows:

\[
\begin{align*}
\text{DecideHalt(} &\langle M, w \rangle\rangle: \\
\text{Encode the following Turing machine } &M': \\
M'(x): & \begin{cases} 
\text{run } M \text{ on input } w \\
\text{if } x = \varepsilon \text{ or } x = 0 \text{ or } x = 1 \\
\quad \text{return } \text{True} \\
\text{else} \\
\quad \text{return } \text{False} 
\end{cases} \\
\text{if } \text{DecideAcceptThree}(\langle M' \rangle) & \text{return } \text{True} \\
& \text{else} \\
& \text{return } \text{False}
\end{align*}
\]

We prove this reduction correct as follows:

\[\implies\] Suppose \( M \) halts on input \( w \).
Then \( M' \) accepts exactly three strings: \( \varepsilon, 0, \) and \( 1 \).
So \text{DecideAcceptThree} accepts the encoding \( \langle M' \rangle \).
So \text{DecideHalt} correctly accepts the encoding \( \langle M, w \rangle \).

\[\iff\] Suppose \( M \) does not halt on input \( w \).
Then \( M' \) diverges on every input string \( x \).
In particular, \( M' \) does not accept exactly three strings (because \( 0 \neq 3 \)).
So \text{DecideAcceptThree} rejects the encoding \( \langle M' \rangle \).
So \text{DecideHalt} correctly rejects the encoding \( \langle M, w \rangle \).

In both cases, \text{DecideHalt} is correct. But that’s impossible, because \text{HALT} is undecidable. We conclude that the algorithm \text{DecideAcceptThree} does not exist.

AcceptPalindrome := \{ (M) \mid M \text{ accepts at least one palindrome} \}

Solution:
For the sake of argument, suppose there is an algorithm DecideAcceptPalindrome that correctly decides the language AcceptPalindrome. Then we can solve the halting problem as follows:

\[
\begin{align*}
\text{DecideHalt(} &\langle M, w \rangle\rangle: \\
\text{Encode the following Turing machine } &M': \\
M'(x): & \begin{cases} 
\text{run } M \text{ on input } w \\
\text{return } \text{True} 
\end{cases} \\
\text{if } \text{DecideAcceptPalindrome}(\langle M' \rangle) & \text{return } \text{True} \\
& \text{else} \\
& \text{return } \text{False}
\end{align*}
\]
We prove this reduction correct as follows:

\[ \implies \text{ Suppose } M \text{ halts on input } w. \]
\[ \quad \text{Then } M' \text{ accepts every input string } x. \]
\[ \quad \text{In particular, } M' \text{ accepts the palindrome } RACECAR. \]
\[ \quad \text{So DecideAcceptPalindrome accepts the encoding } \langle M' \rangle. \]
\[ \quad \text{So DecideHalt correctly accepts the encoding } \langle M, w \rangle. \]

\[ \iff \text{ Suppose } M \text{ does not halt on input } w. \]
\[ \quad \text{Then } M' \text{ diverges on every input string } x. \]
\[ \quad \text{In particular, } M' \text{ does not accept any palindromes.} \]
\[ \quad \text{So DecideAcceptPalindrome rejects the encoding } \langle M' \rangle. \]
\[ \quad \text{So DecideHalt correctly rejects the encoding } \langle M, w \rangle. \]

In both cases, DecideHalt is correct. But that’s impossible, because HALT is undecidable. We conclude that the algorithm DecideAcceptPalindrome does not exist.

Yes, this is exactly the same proof as for problem 1.