Prove that the following languages are undecidable.

1. ACCEPTILLINI := \{ \langle M \rangle \mid M \text{ accepts the string } \text{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(\langle M' \rangle)
return TRUE
else
return FALSE
```

We prove this reduction correct as follows:

\[\Rightarrow\] 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 \langle M' \rangle.
So DECIDEHALT correctly accepts the encoding \langle M, w \rangle.

\[\Leftarrow\] 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 \langle M' \rangle.
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 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.
2. \textbf{AcceptThree} := \{ \langle M \rangle \mid M \text{ accepts exactly three strings} \}

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

\begin{verbatim}
DECEIDEHALT((M, w)):
Encode the following Turing machine M':
M'(x):
    run M on input w
    if x = \epsilon or x = 0 or x = 1
        return \textsc{True}
    else
        return \textsc{False}

if \textsc{DecideAcceptThree}(M')
    return \textsc{True}
else
    return \textsc{False}
\end{verbatim}

We prove this reduction correct as follows:

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

\(\impliedby\) 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 \textsc{DecideAcceptThree} rejects the encoding \(\langle M' \rangle\).
So \textsc{DecideHalt} correctly rejects the encoding \(\langle M, w \rangle\).

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

\[
\begin{array}{c}
\text{\textbf{Lab 13 Solutions}} \\
\text{Spring 2021}
\end{array}
\]
3. **Accept Palindrome** := \( \{ \langle M \rangle \mid M \text{ accepts at least one palindrome} \} \)

**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:

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

We prove this reduction correct as follows:

\[\implies\] Suppose \( M \) halts on input \( w \).
  Then \( M' \) accepts every input string \( x \).
  In particular, \( M' \) accepts the palindrome \text{RACECAR}.
  So `DecideAcceptPalindrome` accepts the encoding \( \langle M' \rangle \).
  So `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 any palindromes.
  So `DecideAcceptPalindrome` rejects the encoding \( \langle M' \rangle \).
  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.■