Prove that the following languages are undecidable.

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

1 ACCEPTILLINI := $\{\langle M\rangle \mid M$ 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:

```
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:

$\Longrightarrow$ 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 $\langle M, w\rangle$.

$\Longleftarrow$ 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 undecidablility 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 $\textsc{AcceptThree} := \{\langle M\rangle \mid M \text{ accepts exactly three strings}\}$

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:

```
DecideHalt(⟨M, w⟩):
  Encode the following Turing machine M':
    M'(x):
      run M on input w
      if x = ε or x = 0 or x = 1
          return True
      else
          return False
  if DecideAcceptThree(⟨M'⟩)
      return True
  else
      return False
```

We prove this reduction correct as follows:

$\Longrightarrow$ Suppose M halts on input w.
Then M' accepts exactly three strings: ε, 0, and 1.
So **DecideAcceptThree** accepts the encoding $\langle M'\rangle$.
So **DecideHalt** correctly accepts the encoding $\langle M, w\rangle$.

$\Longleftarrow$ 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 **DecideAcceptThree** 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 **DecideAcceptThree** does not exist.

3 $\textsc{AcceptPalindrome} := \{\langle M\rangle \mid M \text{ accepts at least one palindrome}\}$

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

We prove this reduction correct as follows:

$\Longrightarrow$ Suppose M halts on input w.
Then M' accepts *every* input string x.
In particular, M' accepts the palindrome $RACECAR$.
So **DecideAcceptPalindrome** accepts the encoding $\langle M' \rangle$.
So **DecideHalt** correctly accepts the encoding $\langle M, w \rangle$.

$\Longleftarrow$ 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.

4 Prove that the following language is undecidable:

$$L = \{\langle M \rangle \mid M \text{ is a Turing machine, and } L(M) \text{ is decidable but not context free}\}.$$

Solution:

Lemma 0.1. *The language L is undecidable.*

Proof: We assume, for the sake of contradiction, that L is decidable. Namely, there is Turing machine N that decides it. That is, a Turing machine that always stop on any input, and accept only inputs $\langle M \rangle$ such that $\langle M \rangle \in L$.

We show a reduction from the Halting problem, which its associated language is

$$\mathrm{A_{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a TM and } M \text{ accepts } w \right\}.$$

We know that this language is undecidable (that is, there is no Turing machine that always stop, and accept this language).

So, we are given an instance $\langle M, w \rangle$ of Halting, and we want to decide if it is in A_{TM}, using the given N. To this end, we create a new program (i.e., Turing machine):

$f(\langle M, w \rangle) = \langle M' \rangle =$

```
Input: x
Code:
  r ←Run M on w
  if r = accept and x ∈ L* = {0^i 1^i 2^i 3^i | i ≥ 0} then
    return Accept
  else
    return Reject
```

Clearly, the TM M' and its encoding can be computed from $\langle M, w \rangle$ (it is essentially simple text manipulation). We now feed $\langle M' \rangle$ into the decider N for L. If M accepts w, then the language of M' is L^* which is decidable but not context-free (see pre-recorded lecture 7.8 for a proof of that, or just accept this as true). If M does not accept w, than $L(M') = \emptyset$, which is definitely a context-free language (the empty language is also regular).

Formally, now create a new decider for A_{TM} using N. Specifically, the new decider is the following.

NewHaltingDecider$(\langle m, w\rangle)$:
`Compute` $\langle M'\rangle \leftarrow f(\langle M, w\rangle)$
return $N(\langle M'\rangle)$

If N always stops, and decides L, then **NewHaltingDecider** always stops, and decides A_{TM}. Indeed, if N accepts $\langle M'\rangle$, then M accepts w. Similarly, if N rejects $\langle M'\rangle$, then M either rejects w, or M never stops in w. In either case, the new decider **NewHaltingDecider** returns the right result. But this is impossible, because by the Halting Theorem, the language A_{TM} does not have a decider. ■