Prove that the halting problem for Turing machine is undecidable.

We can think of the Turing machine for the following pseudocode. The function halt takes two parameters as input. The first parameter is a program and the second is the argument to that program. We send breakHalt in as the argument to itself. The program therefore says that if breakHalt doesn’t halt then we are done, otherwise we loop forever. This creates a contradiction, therefore the halting problem is undecidable.

```python
function breakHalt(string s)
    if halt(s, s) = false
        return true
    else
        loop forever
```
Rice’s Theorem: Every nontrivial property of the RE languages is undecidable.

Proof:
Assume \( \emptyset \), the empty language is not in the nontrivial property \( P \).
Let \( L \) be a nonempty language in \( P \) and \( M_L \) be a Turing Machine accepting \( L \).

Explanation: The reduction from \( L_u \) to \( L_P \) will be sufficient in proving \( L_P \) is undecidable since we’ve already established that \( L_u \) is. We need to construct a Turing Machine \( M' \) that takes as input \( (M,w) \) which is just another Turing Machine and a string. If \( M \) accepts \( w \) then it starts up \( M_L \) so that \( M_L \) can begin to process \( x \). However, if \( M \) does not accept/halt, then it will loop forever and \( M' \) will not return a decision. This means the languages accepted by \( L(M') \) is \( \emptyset \).

Designing our \( M' \) in this manner gives us the following conclusions:
1) The language accepted by \( M' \), denoted \( L(M') \) is \( \emptyset \) if \( M \) does not accept \( w \).
2) \( L(M') = L \) if \( M \) accepts \( w \).

Case that \( M \) doesn’t accept \( w \): Since we assumed \( \emptyset \) is not in property \( P \), that means the code for \( M' \) is not in \( L_P \).

Case that \( M \) does accept \( w \): \( M' \) will accept exactly the language \( L \) in \( P \) which means the code for \( M' \) is in \( L_P \).

Conclusion:
This algorithm turns \( (M,w) \) into an \( M' \) in \( L_P \) \( \iff \) \( (M,w) \) is in \( L_u \)
Hence, the property \( P \) is undecidable.
3. We modify the diagonalization argument to create such a language. Suppose the r.e.
languages are all enumerated as \( \{L_i \mid i \geq 1\} \) and the strings are all enumerated as
\( \{w_i \mid i \geq 1\} \). Define the new language \( L \) as follows. For any language \( L' \) and a string \( w \),
denote by \( L'(w) \) the predicate that is true iff \( w \) is in \( L' \) and is false else. Then for every
\( i \geq 1 \), define \( L(w_{2i-1}) = L_i(w_{2i-1}) \) and \( L(w_{2i}) = \neg L_i(w_{2i}) \). By this definition, for any r.e.
language \( L_i \), \( L \neq L_i \) and \( \overline{L} \neq L_i \). So both are non-r.e.
4. **Which of the following classes of languages are closed under complement?**

Finite Sets: No, the complement of any finite set is an infinite set.

Regular Sets: Yes.

Context-free languages: No, the complement of \( \{a^n b^n c^n \mid n > 0\} \) is a CFL

Recursive Sets: Yes. For a set to be recursive, there must be a Turing Machine M that halts and either accepts or doesn’t accept for all inputs. The complement of the set would have a Turing Machine M’ that simply returns the opposite result of M.

R.E. Sets: No. Recursively enumerable sets have a Turing Machine M that will eventually halt on accepting strings, but M need not halt on unaccepted strings. The machine for the complement of a R.E. set, M’ would then have to accept all strings that M doesn’t accept. Since M need not halt, M’ would not be a valid Turing Machine since it can’t guarantee that it will eventually halt and accept.