Answer the questions below, paying particular attention to the logic of your arguments. Short descriptions for how a function might computed (on any machine we’ve talked about) are sufficient to show that it is computable. Pseudocode is often appropriate for reductions.

1. Show that if is recognizable and reduces to , then is decidable.

By definition, a language L is decidable if and only if L and its complement are recognizable. We are given that A is recognizable, so all that is left is to show that is recognizable. From the definition of reduction, if A is a reduction of B and B is recognizable, then A is recognizable as well. In this case, is a reduction of A and A is recognizable, so is recognizable as well. So we have shown that A and are both recognizable, so A is decidable.

1. Let accepts exactly one of the strings 00 and 11, i.e. .
   1. What does Rice's theorem say about B?

This is a nontrivial property of the Turing machine. As such, the language B is undecidable.

* 1. Show that halting problem ( halts ) reduces to B.

Def R(<M>):

Def N(x):

M(‘’)

Return x == ‘00’

Return <N>

* 1. Show that halting problem reduces to the complement of B.

Def R(<M>):

Def N(x):

M(‘’)

Return x != ‘00’ AND x != ‘11’

Return <N>

* 1. Are B or its complement recognizable?

The halting problem might loop, but that is acceptable for recognizability. In this case, since B and its complement are reductions of the halting problem, they too are recognizable by the properties of reduction.

1. A language is *co-recognizable* if its complement is recognizable. Argue why each of the following languages is or is not recognizable and why it is or is not co-recognizable.
   1. enters state for the input string .

L1 is recognizable. By definition of decidable, the machine must accept <M> if it enters state q27 but can reject or loop on other input. This language is recognizable because the machine can accept if state q27 is reached. If it halts or loops before reaching state q27, then the machine neither accepts nor rejects, fulfilling the requirements of recognizability.

The complement of L1 (the string representations of machines such that the machine does *not* enter state q27 for input 001101) is not recognizable. The machine would always have to halt for this language to be recognizable. If the machine enters a loop, there is no way for the recognizer to know if it is stuck in a loop or is simply taking a long time to compute and could potentially enter state q27. This is the only way to know for certain that the machine has completed its computation and did not enter state q27 for the input 001101.

Since the complement of L1 is not recognizable, L1 is not co-recognizable.

* 1. contains at most two strings}.

Proving that L2 is recognizable is hard. You have to go through the entire set of strings (infinite!) and show that the machine accepts at most two strings.

Showing that the complement of L2 (the string representations of machines that accept three or more strings) is recognizable is much easier. You have to go through the set of strings in a dovetail fashion and you can halt whenever you accept three or more strings.

L3 accepts anything (including the empty string). This is different from everything, You can simulate and as soon as you get an accept, then you know that <M> is in the language.

The complement of L3 rejects everything, including the empty string. To test this, you would have to simulate every sting in sigma\* and see whether it is rejected by the Turing machine.

1. A computable verifier is a deterministic Turing machine V that takes two arguments: x (the input) and y (the proof). A computable verifier always halts. Show that a language L is recognizable if and only if there exists a computable verifier V such that
   1. if , then there is a string y such that V(x,y) accepts, and
   2. if then V(x,y) rejects for every string y.

First, we need to show that if L is recognizable, a computable verifier exists. From the definition of recognizability, we know that for all strings *x* that are elements of language L, there exists a Turing machine M such that M accepts x. For strings that are not elements of L, M may loop or reject, but not accept. In this case with verifier V(x,y), we let V simulate the machine M that accepts L and define y to be the number of steps that V is allowed to execute. We can see that a y exists such that it prevents V from looping infinitely but provides sufficient steps that all the strings that are elements of L will be accepted. If V does not accept before y steps, we assume that it is looping and reject x. So we have shown that if L is recognizable, computable verifier V exists with the specified properties.

Second, we need to show that if a computable verifier V with the above properties exists for language L, than L is recognizable. Since we are assuming we have a verifier, we can create a machine M that uses V to recognize L. V has to accept all strings within language L with at least one proof y, but has to reject all strings not in L no matter the proof y that is used. For this reason, we cannot define y to be the number of steps V is allowed to run as we did above. Instead, we have to define y to be a proof represented in a (potentially infinite) binary string, i.e., . If x is in language L, recognizer M passes new proofs y to verifier V until V accepts at which point M accepts. If string x is not in language L, recognizer M will pass new proofs to V, but V will never accept, so M will loop forever, thus not accepting input x. This meets the requirements for recognizability and thus L is recognizable.

1. Consider the following property: is the language of some Turing machine that has an odd number of states. Show that is a trivial property. (Yes, this problem is trivial).

Another way of thinking about this is to apply Rice’s theorem. In this case, it would be possible to build two different machines that both recognize language L but are not both elements of P. In other words, they could both recognize the same language, but one uses an even number of states to do so and the other uses an odd number of states to do so.

[Bonus] Consider the property is accepted by some Turing machine that never halts after an even number of steps. Show that is a trivial property.

We can apply Rice’s theorem again here. If the property does not meet the conditions of Rice’s theorem, then it is a trivial property.

1. [Bonus] Read about The Recursion Theorem in the Sipser text. One implication of the recursion theorem is that in any general purpose programming language, one can write code that outputs the code itself. Write a python program that prints its own code. Do not use any file operations.

https://www.udacity.com/course/viewer#!/c-ud557/l-1209378918/m-2986218594