1 Exercises

Determine which of the following decision problems are undecidable and which are decidable. Remember the key to proving that "Problem P is undecidable" is to provide an argument that "If we had a black box that solved P, then we could solve X" where X is some problem already known to be undecidable (most frequently the halting problem). That is: if problem X is reducible to problem P, and X is undecidable, then P must be undecidable.

(Note that a *transition* is any single "read-write-move" event, even if it does not involve a change of state or tape)

1. SOMETIMES HALT
   Instance: A Turing machine M.
   Problem: Does M halt on some input?

   The reduction we've used previously works to show this is undecidable. Convert an instance $R(M)w$ of HALT to one of SOMETIMES HALT by converting it to a Turing machine N (which depends on M and w) that works as follows:

   - Erase any contents of the input tape
   - Write w on the input tape
   - Run M

   This machine N is positive for SOMETIMES HALT if and only if $R(M)w$ was positive for HALT. Since we can construct N from $R(M)w$, we have a reduction of HALT to SOMETIMES HALT and hence SOMETIMES HALT is undecidable.

2. EVEN HALT
   Instance: A Turing machine M and an input word w.
   Problem: Does M halt on input w after an even number of transitions?

   It's clearly feasible to add some sort of dummy transitions so that given a machine M we can construct another one that takes 2 transitions for every one of M's. That machine halts after an even number of transitions on input word w if and only if

M halts on w. So, if we could decide EVEN HALT we could also decide HALT and therefore EVEN HALT must be undecidable.

3. BOUNDED HALT
   Instance: A Turing machine M, an input word w and an integer n.
   Problem: Does M halt on input w without making more than n transitions?

   This is decidable. We simply add a "clock tape" that we initialise to n and subtract one from on every transition. We halt and accept if we don't run out of "time" and halt and reject otherwise when we do.

4. (*) NO WRITE
   Instance: A Turing machine M and an input word w.
   Problem: Does M halt on input w without making any change to the tape (i.e. all transitions only move the tape head)?

   This is a bit tricky. If the read-write head stays within a bounded region of the tape (e.g., the region of the actual input) then we can decide this by recording each combination of tape head position and state. There are only a finite number of such pairs so either we will halt, or we will see a repeated pair at some point in which case we're in an infinite loop (and can announce that).

   So let's do that anyway – let's record, for each position on the original input tape the state we're in when we visit that cell (each time we do so). If we see a duplicate then we decide that M is not going to halt.

   If M is not going to halt but we don't observe that using the technique above then it must be the case that at some point we move beyond the original input area and never return. So, whenever the head is in that region we just record the current state and the location where we saw it. Whenever we return to the original input we can wipe that complete record. If we do get stuck there will be a point at which we are in some state, and later we are in the same state but further along the tape (and we didn't return to the input area in between). Now we're caught in a loop - we're going to repeat all those transitions, not return to the input (because we started further along) and we'll be back in the same state again, still further along the tape. So, when that happens we can stop and announce "not halt"

In other words - we'll recognise that we do halt when we do, and if we're not going to halt, we'll be able to diagnose that in finite time – i.e., we can decide NO WRITE.

5. *(Review) Show that the language* HALT *consisting of all strings $R(M)w$ such that M halts on input w* is *recursively enumerable. What can be said about its complement?*

   It's recursively enumerable because it's precisely the language accepted by a Universal Turing Machine. We know that it's not recursive, and therefore its complement must not be recursively enumerable (since, if both a language and its complement are recursively enumerable, then the language is recursive.)