1. Consider the following random access Turing machine:

1. load =1 // Store 1 in R0
2. store 1 // Store contents of R0 in R1
3. load =0 // Store 0 in R0
4. store 2 // Store contents of R0 in R2
5. read 1 // Read T[R1] into R0
6. jzero 13 // If contents of R0 == 0, go to 13
7. add 2 // Add contents of R2 to R0
8. store 2 // Store contents of R0 in R2
9. load 1 // Load contents of R1 into R0
10. add =1 // Add 1 to contents of R0
11. store 1 // Store contents of R0 in R1
12. jump 5 // Go to 5
13. load =1 // Set R0 to 1
14. store 1 // Put the 1 into R1
15. load 2 // Load contents of R2 into R0
16. write 1 // Write R0 contents to T[R1] = T[1]
17. halt

(a) If the input tape contains the following pairs of decimal integers, find the output of the machine.

(1,1)(2,3)(3,2)(4,1)(5,0)

The contents of the tape are the same except that the second component of the first pair is 7 instead of 1.

(b) More generally, describe what the machine does if its input tape has the “proper” format?
The tape has “proper” format if the first components of the pairs are numbered 1, 2, . . . , k, for some positive integer k, and the kth pair is (k, 0). The program adds the second components of the tape pairs until it encounters a 0 for the second component. When this happens it writes the sum into the second component of the first tape pair.

2. First part of problem 4.5.2 on page 227.

From q_0, the machine will transition to q_1 and overwrite the first square with a blank or an a:

\[(q_0, \triangleright \sqcup) \vdash (q_1, \triangleright \sqcup),\]

or

\[(q_0, \triangleright \sqcup) \vdash (q_1, \triangleright a).\]

From q_1, if the input is blank, the machine will loop forever in q_1 writing blanks:

\[(q_0, \triangleright \sqcup) \vdash^* (q_1, \triangleright \sqcup).\]

From q_1, if the input is a, the machine will either halt and move 1 square to the right:

\[(q_0, \triangleright \sqcup) \vdash^* (h, \triangleright a \sqcup),\]

or it will return to q_0 and move 1 square to the right:

\[(q_0, \triangleright \sqcup) \vdash^* (q_0, \triangleright a \sqcup).\]

So the machine will write some indeterminate number of a’s, moving to the right after each is written, and then either halt on the blank to the right of the a’s, or it will loop forever on the blank to the right of the a’s.

A tree diagram with nodes showing the current machine configurations, and edges indicating the transitions, contains 6 leaves with depth (i.e., distance to the root) less than or equal to 5. Of these leave, two are halted, three are infinite loops, and one has neither halted nor entered an infinite loop.

3. Use the grammar in example 4.6.2 on page 229 to derive the string aabbcc.

\[S \Rightarrow ABCS \Rightarrow ABCABCS \Rightarrow ABCABCT_c \Rightarrow ABACBCT_c \Rightarrow AABCBCT_c \Rightarrow AABBCCT_c \Rightarrow AABBBCT_c, \]

or

\[AABBT_i c \Rightarrow AABBT_i c \Rightarrow AABBT_i b c \Rightarrow AABT_i b c c \Rightarrow AAT_i b c c \Rightarrow AT_i a b b c c \Rightarrow T_i a a b b c c \Rightarrow eaabbcc.\]

4. Problem 5.2.1, parts (a), (b), and (c) on page 250.

For the TM in Example 4.1.1, \(|K| = 3\), and \(|\Sigma| + 2 = 5\). So we need \(i = 2\) and \(j = 3\). This gives the following representation of states and symbols.
(a) The representation of \( M \) is

\[
(q_{00}, a_{000}, q_{10}, a_{000}), \quad (q_{00}, a_{001}, q_{00}, a_{011}), \quad (q_{00}, a_{100}, q_{01}, a_{000}), \\
(q_{01}, a_{000}, q_{01}, a_{011}), \quad (q_{01}, a_{001}, q_{01}, a_{011}), \quad (q_{01}, a_{100}, q_{00}, a_{100}).
\]

(b) The representation of \( aaa \) is \( a_{100}a_{100}a_{100} \).

(c) At the beginning of the simulation, after the input "\( M \)“ \( w \) has been distributed among the three tapes, the first tape of \( U' \) contains

\[
\geq a_{001}a_{000}a_{100}a_{100}a_{100}.
\]

Depending on how you read the text, the first head is either under the \( a \) in \( a_{000} \) or under the \( a \) in the first \( a_{100} \). The second tape contains

\[
\geq q_{00}a_{000}q_{00}a_{000}q_{00}a_{001}q_{00}a_{011}q_{00}a_{100}q_{01}a_{000} \\
q_{01}a_{000}q_{01}a_{011}q_{01}a_{001}q_{01}a_{011}q_{01}a_{100}q_{00}a_{100}.
\]

The third tape contains

\[
\geq q_{00}.
\]

The answer to the second part of the question depends on where you place the head of the first tape at the start of the simulation. If it’s placed under the \( a \) in \( a_{000} \) (i.e., “under” the encoded blank), then the machine halts immediately, and there is no third step. If, the head of the first tape is started under the \( a \) in the first \( a_{100} \), (i.e., “under” the first encoded \( a \)), then the configurations of the (unsimulated) machine are

\[
(q_0, \geq \sqcup a_{aa}) \vdash (q_1, \geq \sqcup \sqcup a_{aa}) \\
\vdash (q_0, \geq \sqcup a_{aa})
\]

So at the start of the third simulated step, the contents of the first tape will be

\[
\geq a_{001}a_{000}a_{000}a_{100}a_{100}.
\]

The contents of the second and third tapes will be the same as they were at the start of the simulation.
5. Consider the language

\[ H = \{ "M" "w" : \text{Turing machine } M \text{ halts on input } w \} . \]

In class we saw that \( H \) is recursively enumerable. We also saw that if \( H \) is recursive, then every recursively enumerable language is also recursive.

This problem looks at the remainder of the proof that \( H \) is not recursive. (See page 252, paragraphs 2, 3, and 4.)

(a) Consider the language

\[ H_1 = \{ "M" : \text{Turing machine } M \text{ halts on input } "M" \} . \]

Show that if \( H \) is recursive, then \( H_1 \) is recursive.

So in order to show that \( H \) is not recursive, it suffices to show that \( H_1 \) is not recursive.

Suppose that \( H \) is recursive. Then there is a TM \( M_0 \) that decides \( H \). That is, if its input belongs to \( H \), then it halts in \( y \), and if its input doesn’t belong to \( H \), it halts in \( n \).

We can use \( M_0 \) to build a TM \( M_1 \) that decides \( H_1 \). \( M_1 \) first transforms its input string

\[ \triangledown "M" \]

into the input string

\[ \triangledown "M" "M" . \]

\( M_1 \) now runs \( M_0 \) on this input. So if \( "M" \in H_1 \), then \( M \) halts on input \( "M" \), and hence \( "M" "M" \) belongs to \( H \). So \( M_0 \) halts in \( y \), on input \( "M" "M" \). If \( "M" \notin H_1 \), then \( M \) runs forever on input \( "M" \), and hence \( M_0 \) halts in \( n \) on input \( "M" "M" \). That is, \( M_1 \) decides \( H_1 \).

(b) If \( H_1 \) were recursive, then its complement

\[ \overline{H_1} = \{ w : w \text{ is not the encoding of a Turing machine} \} \]

would also be recursive. Why?

By theorem 4.2.2 on page 199, the complement of a recursive language is also recursive.

(c) If \( \overline{H_1} \) isn’t recursively enumerable, explain why it can’t be recursive.

By theorem 4.2.1 on page 199, the recursive languages are contained in the recursively enumerable languages. So if \( \overline{H_1} \) isn’t recursively enumerable, it can’t be recursive.

(d) Show that \( \overline{H_1} \) isn’t recursively enumerable.

Suppose to the contrary that \( \overline{H_1} \) is recursively enumerable. Then there is a TM \( M^* \) that semidecides \( \overline{H_1} \). That is, if \( w \) is not the encoding of TM then \( M^* \) halts on input \( w \). Furthermore, if \( M \) is a TM that does not halt on input \( "M" \), then \( M^* \) halts on input
“$M$”. Finally, $M$ is a TM that halts on input “$M$” (i.e., “$M$” $\in H_1$), then $M^*$ runs forever on input “$M$”.

Now we want to look at whether “$M^*$” belongs to $\overline{H_1}$.

Suppose first that “$M^*$” belongs to $\overline{H_1}$. Since “$M^*$” is an encoding of a TM, “$M^*$” must belong to the second set making up $\overline{H_1}$. That is, $M^*$ does not halt on input “$M^*$”. But $M^*$ semidecides $\overline{H_1}$. So if “$M^*$” belongs to $\overline{H_1}$, we must have that $M^*$ halts on input “$M^*$”. So “$M^*$” $\in \overline{H_1}$ can’t be the case.

So suppose that “$M^*$” $\not\in \overline{H_1}$. That is, $M^* \in H_1$. So $M^*$ halts on input “$M^*$”. But by the definition of $M^*$, it runs forever when its input belongs to $H_1$. So this also cannot be the case.

That is, there can be no Turing machine that semidecides $\overline{H_1}$, and $\overline{H_1}$ is not recursively enumerable.

Now working our way back through the arguments, we have

1. $\overline{H_1}$ is not recursively enumerable.
2. $\overline{H_1}$ is not recursive.
3. The complement of $\overline{H_1}$, $H_1$ cannot be recursive.
4. The language $H$ cannot be recursive.

So $H$ provides us with an example of a language that is recursively enumerable, but not recursive. $\overline{H_1}$ provides us with an example of a language that isn’t even recursively enumerable.