Name: Kevin Zhang

Problem 1.

(a) M_1 decides L (and as a consequence, also recognizes L). The reason is that for any given string (including the empty string), M producesa valid output. The transition $s\ 1\ 1\ R\ q_{accept}$ covers all the cases where w is in L (and therefore begins with 1). State q eventually reaches q_{reject} when the machine reaches the end of the string, and all transitions from s eventually lead to q.

(b) M_2 neither recognizes nor decides L. The reason is that there is no transition from s to q – the transition $s\ 0\ 0\ R\ s$ loops back to s. This means that M_2 decides the language that has a 1 – the machine reaches q_{accept} when it sees 1, and q_{reject} if it reaches the end without seeing 1. This is not the same as L.

(c) M_3 recognizes L. The reason is that it accepts every word that starts with 1, and rejects every word that starts with 0. The only case that M_3 loops infinitely in is the empty string, because of the transition $s\ \sqcup\ \sqcup\ R\ s$.

(d) M_4 decides L (and as a consequence, also recognizes L). The reason is similar to M_1, where the machine eventually rejects all strings that begin with 0, and accepts all strings that start with 1. The empty string, in this case, is rejected outright, instead of being rejected when the machine reaches the end.

Problem 2.

M decides L when it has a two-way infinite tape. The reason is because when the tape is one-way, M runs into problems when the word is not greater than length 2. For example, 0 is a string that should be rejected by M, but on a one-way tape, the move-left transition gets ignored, but the state transition still happens. The transitions might proceed as follows: $(s, 0)(s, 0, R)$, $(s, \sqcup)(q_1, \sqcup, L)$, $(q_1, 0)(q_2, 0, L)$, $(q_2, 0)(q_{accept}, 0, R)$. The last transition happens because the machine stays put on the left-most cell. As a result, 0 is accepted, when it isn't in L.

Problem 3.

$$\mathrm{q} \quad \begin{pmatrix} b \\ \sqcup \end{pmatrix} \quad q_{reject} \quad \begin{pmatrix} b \\ \sqcup \end{pmatrix} \quad \begin{pmatrix} S \\ S \end{pmatrix}$$

$$\mathrm{q} \quad \begin{pmatrix} b \\ a \end{pmatrix} \quad \mathrm{q} \quad \begin{pmatrix} \sqcup \\ \sqcup \end{pmatrix} \quad \begin{pmatrix} R \\ L \end{pmatrix}$$

$$\mathrm{q} \quad \begin{pmatrix} \sqcup \\ a \end{pmatrix} \quad q_{reject} \quad \begin{pmatrix} \sqcup \\ a \end{pmatrix} \quad \begin{pmatrix} S \\ S \end{pmatrix}$$

$$\mathrm{q} \quad \begin{pmatrix} \sqcup \\ \sqcup \end{pmatrix} \quad q_{accept} \quad \begin{pmatrix} \sqcup \\ \sqcup \end{pmatrix} \quad \begin{pmatrix} S \\ S \end{pmatrix}$$

$$\mathrm{q} \quad \begin{pmatrix} a \\ \sqcup \end{pmatrix} \quad q_{reject} \quad \begin{pmatrix} a \\ \sqcup \end{pmatrix} \quad \begin{pmatrix} S \\ S \end{pmatrix}$$

$$\mathrm{q} \quad \begin{pmatrix} a \\ a \end{pmatrix} \quad q_{reject} \quad \begin{pmatrix} a \\ a \end{pmatrix} \quad \begin{pmatrix} S \\ S \end{pmatrix}$$

Problem 4.

We can design a 5-state turing machine with a tape alphabet that includes w as one of its symbols. **Parts** of an example machine for the string 00001 can be shown.

```
start: s
accept: good
reject: bad

transitions:
- [s, _, s, 00001$, S]
- [s, 00001$, p0, 0001$, R]
- [s, 0001$, p0, 001$, R]
- [s, 001$, p0, 01$, R]
- [s, 01$, p0, 1$, R]
- [s, 1$, p1, $, R]

- [s, 0, s, 0, L]
- [s, 1, s, 1, L]
- [s, $, good, _, R]

- [p0, _, s, 0, L]
- [p0, 0, p0, 0, R]
- [p0, 1, p0, 1, R]

- [p1, _, s, 1, L]
- [p1, 0, p1, 0, R]
- [p1, 1, p1, 1, R]
```

The way the machine is to use a larger tape alphabet to represent what is left to write. For example, we start with 00001, and pop off the first 0, and write 0001. The popped 0 is then encoded into the state **p0**, which will place the 0 in the first open space to the right, before returning. This will happen for each symbol, with **p1** representing when we want to move 1 instead of 0. When we are done popping, we put a $ as a placeholder (so that **s** can make its way back), and then move the head to the right, and into q_{accept}. This kind of machine will have as many transitions (and a tape alphabet to match) as the length of string, but will require only 5 states (s, p0, p1, q_{accept}, q_{reject}).

Generalized, we can represent a word w as $a_0a_1a_2a_3...a_n$. Our starting transition is then

`[s, _, s,` $a_0a_1a_2a_3...a_n$`$, S]`

For every letter $a_i | 0 \leq i \leq n$, we will also have the transition

`[s,` $a_ia_{i+1}...a_n$`$, p0 | p1,` $a_{i+1}...a_n$`$, R]`

where `p0 | p1` is dependent on what a_i is. The rest of the machine can then stay the same.

Problem 5.

(a) $L \subseteq \Sigma^*$is decidable $\Rightarrow$ There is an enumerator for L that enumerates in shortlex order
Assume L is decidable with machine M. Then, we can take $w_1, w_2, w_3, ... \in \Sigma^*$, where the words are taken in shortlex order. We run these words through M, which will either accept, or reject. We can construct an enumerator that prints out only the words that accept – this results in an enumerator for L that prints the words in shortlex order.

(b) $L \subseteq \Sigma^*$is decidable $\Leftarrow$ There is an enumerator for L that enumerates in shortlex order
Assume there is an enumerator for L that enumerates in shortlex order. Then, we can construct a decider M that on input w, runs the enumerator until it reaches the expected shortlex order. If the enumerator prints w, then we accept. If the enumerator skips over w, moving on to the next item in shortlex order, we reject. For example, if the enumerator prints aa and then aab for $\Sigma = \{a, b\}$, and $w = aaa$, we can safely reject, knowing that if aaa had been in L, the enumerator would have printed aaa before aab (because of the shortlex order constraint).

Problem 6.

Suppose we have an infinite Turing-recognizable language L. From lecture, we know that there must be an enumerator E for L. We can modify this enumerator to $\hat{E}$ such that every time it tries to prints a w, we compare w_n with w_{n-1}, the word printed previously. If w_{n-1} and w_n do not follow shortlex order, we discard w_n (which is to say, we don't let $\hat{E}$ print w_n). As L is an infinite language, we can be assured that the enumerator will eventually print some word w_n that is in shortlex order *after* w_{n-1}.

Thus, we have an enumerator $\hat{E}$ that prints a language $A \subseteq L$ in shortlex order. A is an infinite decidable subset, because of the theorem in problem 5.