Solutions to Homework 3

This homework has 8 questions, for a total of 100 points and 0 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up*.

A few note about this homework:

- This homework is due on **Tuesday, May 28 at 8pm**. The solution will be posted at 10pm on the same day.
- While some parts in this homework are optional and ungraded, **they are within the scope of the midterm**.
- After each question (or in some cases question part), we've indicated which lecture number we expect to cover the relevant material. So "(L8)" indicates that we expect to cover the material in lecture 8.

(0 pts) 0. **Before you start; before you submit.**

If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

Solution:

2. **Decision Problems and Languages.** (L8)

Recall that the goal of a *decision problem* is to determine whether a given input "object" has a certain "property", e.g., determine whether a given integer is prime, or whether a given string is a palindrome. In class, we said that any decision problem is equivalent to a *membership problem* for a corresponding *language*. That is, determining whether a given object has the property is equivalent to determining whether its encoding (as a string) is a member of the language.

For each of the following decision problems, (i) define a reasonable (finite) alphabet Σ, (ii) give the encoding (over the alphabet) of a representative input object, (iii) analyze the length of the encoding in terms of the value of the input, and (iv) define a language L for the corresponding decision problem. Use the notation $\langle X \rangle$ for the encoding of object X as a string over the alphabet. As examples, we have provided solutions to the first two parts.

(a) **Example:** "Does a given non-negative integer k have 3 as its last digit, when written in base 10?"

Solution:

(i) A reasonable alphabet consists of the decimal digits, $\Sigma = \{\texttt{0}, \dots, \texttt{9}\}$. Alternatively, we could use the binary digits $\{0, 1\}$, or hexadecimal digits (0 through 9 and A through F), or the digits from any other base (including unary), or some entirely different-looking alphabet like $\{\star, \otimes, \perp\}$ (though the latter would not be very human-friendly). No matter the choice of (nonempty, finite) alphabet, there is a way to unambiguously encode (represent) non-negative integers as strings of characters from that alphabet.

Note that it would *not* be valid to include all of the non-negative integers in the alphabet, because an alphabet must be a *finite* set, and there are infinitely many non-negative integers.

(ii) Encoding: For an integer k, write it in base 10 in the usual way, as a string of digits. For example, if k is the integer forty-seven, then $\langle k \rangle = \texttt{47}$. We stress that this is a *string* of the characters 4 and 7, which *represents* the number forty-seven.

(iii) The length of this encoding would be the number of digits in the value, which is $\Theta(\log k)$.

(iv) A corresponding language would be $L_{EndsWith3} = \{\langle k \rangle : k \bmod 10 = 3\}$. It would also be acceptable to copy the phrasing of the decision problem, i.e., $L_{EndsWith3} = \{\langle k \rangle : k \text{ written in base 10 has 3 as the last digit}\}$

Alternatively, we could also encode integers in binary, using the alphabet $\Sigma = \{0, 1\}$. For example, if $k = 5$, then $\langle k \rangle = \texttt{101}$. The length of this encoding is still $\Theta(\log k)$. The above two definitions of the language hold without modification, because they both describe membership only in terms of the *value* of k, not its encoding.

(b) **Example:** "Is a given array of non-negative integers sorted?"

Solution:

(i) A reasonable alphabet Σ is the set of ASCII characters, or more selectively, the decimal digits along with some separator characters: $\Sigma = \{\texttt{0}, \dots, \texttt{9}, \texttt{[}, \texttt{,}, \texttt{]}\}$. (Notice that a comma is one of these characters.)

Note that we can't just use the decimal digits alone if we also want some special symbols to indicate the start/end of the array and to separate the elements.

(ii) Example encoding: For an array A, encode its elements as in the previous part, and list those encodings with appropriate separators. For example, the array A with entries one, two, three, and four would have $\langle A\rangle =$ `[1,2,3,4]`.

(iii) The length of this encoding is $\Theta(n+e)$, where n is the number of elements in the array, and e is the total length of the encodings of the elements.

(iv) The corresponding language is

$$L_{\text{sorted}} = \{\langle A\rangle : A \text{ is a sorted array of non-negative integers.}\}.$$

(4 pts) (c) "Given two strings compost of English alphabets, do they have a common subsequence of length k?"

Solution:

(i) A reasonable alphabet Σ is the set of ASCII characters, or more selectively, the English alphabets $\{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}, \texttt{a}, \texttt{b}, \ldots, \texttt{z}\}$, union with the set of decimal digits to for k, and some special character, say '`,`' to separate the two strings.

(ii) Encoding: For the strings, encode its elements using the same characters as in the string with the separators between them, followed by k in its decimal representation. For example, $\langle$"Waquackquack", "Quack", $5\rangle =$ `Waquackquack,quack,5`.

(iii) The length of this encoding is $\Theta(n+\log k)$, where n is the length of the longer string.

(iv) The corresponding language is

$$L_{k\text{CS}} = \{\langle (s_1, s_2, k)\rangle : s_1 \text{ and } s_2 \text{ have a common subsequence of length } k.\}.$$

where s_1 and s_2 are strings compost of English alphabets.

(4 pts) (d) "Does a given undirected graph G has a Hamiltonian cycle?" *Hint: Consider adjacency matrix.*

Solution: Without loss of generality we can name the vertices $1, 2, \ldots, |V|$, and we can represent the edges by an adjacency matrix. To encode this matrix as a string, we can represent each row of the matrix as a 0/1 string, and separate the rows by a special character.

(i) We define $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{,}\}$.

(ii) We encode each row of the graph's adjacency matrix as a bit string, and separate the rows by commas. For example, consider an undirected graph $G = (V, E)$

with $V = \{1, 2, 3\}$ and a single edge $(1, 2)$. Then, the string encoding would be $\langle G\rangle =$ `010,100,000`.

(iii) The length of this encoding is $\Theta(|V|^2)$.

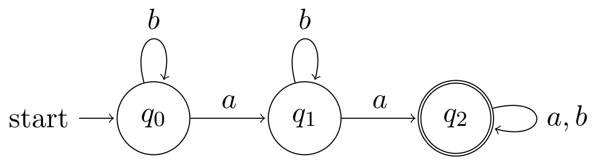
(iv) We could express the language as

$$L_{\text{HamCycle}} = \{\langle G\rangle : G \text{ is an undirected graph with a Hamiltonian cycle.}\}\ .$$

3. **DFAs.** (L8)

For this problem, check out this tool to generate `tikzpicture` script for your DFA.

(4 pts) (a) Consider the following DFA:



Determine the (i) alphabet, (ii) state-transition function and (iii) language decided by the DFA. You may describe the language in English or regular expression, whichever you prefer.

Solution:

(i) $\Sigma = \{a, b\}$

(ii) The transition function is

Current state	Input	Next state
q_0	a	q_1
q_0	b	q_0
q_1	a	q_2
q_1	b	q_1
q_2	a	q_2
q_2	b	q_2

(iii) Observe that the DFA will only accept if it contains at least two a's. Hence we have the language

$$L = \{s \in \{a, b\}^* : s \text{ contains at least two } a\text{'s.}\}.$$

(5 pts) (b) Give a DFA that decides language L over the alphabet $\{a, b\}$ corresponding to the regular expression $a^*ba^*(ba^*ba^*)^*$. You may represent the DFA as a state-transition function or as a diagram, whichever you prefer.

Solution: Observe that we essentially accept when the string contains an odd number of b's.

$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[loop above, "a"] \arrow[r, bend left, "b"] & q_1 \arrow[loop above, "a"] \arrow[l, bend left, "b"]
\end{tikzcd}$$

(6 pts) (c) Give a DFA that decides the following language over the alphabet $\Sigma = \{a, b\}$

$$L = \{s \in \Sigma^* : s \text{ contains at least two } a\text{'s } \textbf{and} \text{ odd number of } b\text{'s.}\}.$$

You may represent the DFA as a state-transition function or as a diagram, whichever you prefer.

Solution: Notice that this is essentially the combination of (a) and (b). Hence we have the DFA

$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[r, "a"] \arrow[d, bend left, "b"] & q_1 \arrow[r, "a"] \arrow[d, bend left, "b"] & q_2 \arrow[loop above, "a"] \arrow[d, bend left, "b"] \\
& q_3 \arrow[r, "a"] \arrow[u, bend left, "b"] \arrow[loop below, "b"] & q_4 \arrow[r, "a"] \arrow[u, bend left, "b"] & q_5 \arrow[u, bend left, "b"] \arrow[loop below, "a"]
\end{tikzcd}$$

Observe that we combine two DFAs from (a) horizontally and three DFAs from (b) vertically.

(15 pts) 4. **Turing Machines Potpourri.** (L9, L11)

Determine whether the following statements about Turing Machines are always/ sometimes/ never true. While justification is **not required** to received for full credit, providing it is highly encouraged.

(a) A language that is decidable by a DFA is (always/ sometimes/ never) decidable by a Turing machine.

(b) A Turing machine will (always/ sometimes/ never) halt before reading in its full input.

(c) If a Turing machine does not halt on a given input, then the number of cells written to on its tape for that input is (always/ sometimes/ never) unbounded.

(d) If a given Turing machine is guaranteed to only write to a finite number of cells on a tape, then it would (always/ sometimes/ never) halt.

(e) A Turing machine can (always/ sometimes/ never) decide its own halting problem.

Solution:

(a) **Always.** A DFA can be thought of as a special kind of Turing machine that does not use its ability to write on the tape or move the tape head left. Therefore, any language that a DFA can decide, a Turing machine can also decide.

(b) **Sometimes.** Consider the following two cases:

A decider for $L_{\text{begin-1}} = \{x : x \text{ begins with a } 1\}$ could simply check the first character on the input tape, and immediately accept or reject from there, without needing to read in the rest of the tape.

A decider for $L_{\text{all-1}} = \{x : x \text{ contains only 1s}\}$ would have to iterate through the whole input tape to check every single character in the input to verify that the input is indeed all 1s.

(c) **Sometimes.** A Turing Machine can rewrite over a cell. A Turing Machine with an unbounded number of steps can write over the same finite number of cells repeatedly. A Turing Machine with an unbounded number of steps could *also* write to a new cell with ever step, thus writing to an unbounded number of cells on its tape.

(d) **Sometimes.** If a Turing Machine only writes to a finite number of cells, then it is possible that the Turing Machine halted. Nevertheless, even if only finitely many cells are written to, the Turing Machine could continue looping over the finite set of cells indefinitely since the head of the Turing Machine can move left or right at each transition.

(e) **Never.** We have shown that L_{HALT} is undecidable, which means no Turing machine can decide its own halting problem.

5. **A Powerful Diagonalization.** (L10)

(7 pts) (a) The *power set* of a set A, denoted $\mathcal{P}(A)$, is defined as the *set of all subsets* of A. For example, $\mathcal{P}(\{0,1\}) = \{\emptyset, \{0\}, \{1\}, \{0,1\}\}$.

Use diagonalization to prove that for any *countably infinite* set A, the power set $\mathcal{P}(A)$ is *uncountably* infinite.

Hint: View each subset as an infinite binary sequence representing whether each element of A is included in the subset or not.

Solution: Let A be a countably infinite set. By definition, there is an enumeration of its elements, as $A = \{a_0, a_1, a_2, a_3, \ldots\}$.

Now suppose for the sake of contradiction that $\mathcal{P}(A)$ is *countably* infinite. (Note that $\mathcal{P}(A)$ is infinite because there are infinitely many 'singleton' subsets $\{a_i\} \in \mathcal{P}(A)$, since A is infinite.) Then by definition, there is an enumeration of all the subsets of A, as

$$A_0,\ A_1,\ A_2,\ A_3,\ \ldots$$

where each $A_i \subseteq A$. Define the "flipped diagonal" set $D = \{a_i : a_i \notin A_i \text{ for } i \geq 0\}$. Notice that $D \subseteq A$, because all its elements are in A. However, by construction, $D \neq A_i$ for every $i \geq 0$ because $a_i \in D$ if and only if $a_i \notin A_i$. Therefore, D does *not* appear in the above enumeration of $\mathcal{P}(A)$, which contradicts our assumption that $\mathcal{P}(A)$ is countable. Therefore, $\mathcal{P}(A)$ is uncountably infinite.

As an alternative (but closely related) approach, note that each subset $A' \subseteq A$ can be equivalently represented by an infinite binary sequence s, whose ith digit is

$$s_i = \begin{cases} 0 & \text{if } a_i \notin A' \\ 1 & \text{if } a_i \in A' \text{ .} \end{cases}$$

That is, each subset corresponds to its own infinite binary sequence, and vice versa. In discussion, we proved by diagonalization that there are *uncountably many* infinite binary sequences, i.e., the *set* of infinite binary sequences is uncountable. Since the subsets of A are in exact (bijective) correspondence with the infinite binary sequences, we conclude that there are uncountably many subsets of A.

(7 pts) (b) Prove that any infinite language $L \subseteq \Sigma^*$ has an undecidable subset $L' \subseteq L$.
Hint: Part (a) will be useful.

Solution: We first show that L is *countably* infinite, which we will need in order to use part (a). This is because $L \subseteq \Sigma^*$: since there is an enumeration of Σ^* (in which every string $x \in \Sigma^*$ appears), we can modify it to get an enumeration of L by just removing the strings that are not in L. (Alternatively: the identity function from L to Σ^* is injective, and there is an injective function from Σ^* to $\mathbb{N}$ because Σ^* is countable, so composing these functions yields an injective function from L to $\mathbb{N}$, which shows that L is countable.)

Now suppose for the sake of contradiction that every $L' \in \mathcal{P}(L)$ (i.e., every $L' \subseteq L$) is decidable. This means that each $L' \in \mathcal{P}(L)$ is decided by some *distinct* Turing machine $M_{L'}$, because a particular Turing machine can decide at most one language. (It's even true that L' is decided by *many* different Turing machines, but for this argument it suffices to take just one.)

We have seen in class that the set of all Turing machines is countable, i.e., they can be enumerated as $M_0, M_1, M_2, \ldots$. We can 'filter' this list to include only those machines $M_{L'}$ identified above, removing all others. Then because every $L' \in \mathcal{P}(L)$ is represented in the filtered list by the distinct machine $M_{L'}$ that decides it, it follows that $\mathcal{P}(L)$ is countable. However, this contradicts the result from part (a), which says that $\mathcal{P}(L)$ is uncountable (here is where we use the fact that L is countably infinite). So our initial assumption was false, i.e., some $L' \in \mathcal{P}(L)$ is undecidable, as claimed.

(Alternatively, we can reach the contradictory conclusion that $\mathcal{P}(L)$ is countable via injective functions. Because a Turing machine can decide at most one language, the function from $\mathcal{P}(L)$ to the set of Turing machines that maps L' to $M_{L'}$ is injective.

Moreover, there is an injective function from the (countable) set of Turing machines to $\mathbb{N}$. So, composing these two functions yields an injective function from $\mathcal{P}(L)$ to $\mathbb{N}$, as needed.)

Alternatively, we can prove the claim directly via diagonalization, similarly to the proofs we have seen in class, discussion, or part (a). Since we proved above that L is countably infinite, we can enumerate it as $L = \{x_0, x_1, x_2, \ldots\}$, and we have previously seen that we can enumerate all Turing machines as $M_0, M_1, M_2, \ldots$. Then, we consider the "flipped diagonal" language $L' = \{x_i \in L : x_i \notin L(M_i) \text{ for } i \geq 0\}$, so $L' \subseteq L$ by definition. By construction, $L' \neq L(M_i)$ for every $i \geq 0$ because $x_i \in L'$ if and only if M_i does *not* accept x_i. Thus, L' is not decided by (or even recognized by!) any Turing machine, because our enumeration has *every* Turing machine in it. So, we have shown that there is an undecidable subset $L' \subseteq L$.

6. **Decidability and Set Operations.** (L11)

Consider four languages where L_{D1} and L_{D2} are decidable, and L_{U1} and L_{U2} are undecidable. Determine whether the following languages are always/ sometimes/ never decidable. The case of $L_{D1} \cap L_{D2}$ is provided as an example. While justification is **not required** to received for full credit, you are highly encouraged to do so.

(a) **Example:** $L_A = L_{D1} \cap L_{D2}$

Solution: Always. Let M_1 be a TM that decides L_{D1}, and M_2 be a TM that decides L_{D2}. We use these to construct the following Turing Machine that decides $L_{D1} \cap L_{D2}$:

```
$M'$ = on input ($x$):
  1: Run $M_1$ on $x$ and run $M_2$ on $x$
  2: if both $M_1$ and $M_2$ accepted $x$ then accept
  3: else reject
```

Analysis:

- $x \in (L_{D1} \cap L_{D2}) \implies x \in L_{D1} \wedge x \in L_{D2} \implies M_1 \text{ accepts } \wedge M_2 \text{ accepts } \implies M \text{ accepts}$
- $x \notin (L_{D1} \cap L_{D2}) \implies x \notin L_{D1} \vee x \notin L_{D2} \implies M_1 \text{ rejects } \vee M_2 \text{ rejects } \implies M \text{ rejects}$

Since M_1 and M_2 are deciders, they necessarily halt on all inputs. Thus, M' must also halt on all inputs. In addition, since M' accepts all $x \in (L_{D1} \cap L_{D2})$ and rejects all $x \notin (L_{D1} \cap L_{D2})$, it follows that M' decides $L_{D1} \cap L_{D2}$. As we have constructed a decider for $(L_{D1} \cap L_{D2})$, we can conclude that $(L_{D1} \cap L_{D2})$ is necessarily decidable.

(4 pts) (b) $L_B = L_{D1} \cup L_{D2}$

Solution: Always. Let M_1 be a TM that decides L_{D1}, and M_2 be a TM that decides L_{D2}. We use these to construct the following Turing Machine that decides $L_{D1} \cup L_{D2}$:

```
M' = on input (x):
  1: Run M_1 on x and run M_2 on x
  2: if either M_1 or M_2 accepted x then accept
  3: else reject
```

Analysis:

- $x \in (L_{D1} \cup L_{D2}) \implies x \in L_{D1} \vee x \in L_{D2} \implies M_1$ accepts $\vee\, M_2$ accepts $\implies M$ accepts
- $x \notin (L_{D1} \cup L_{D2}) \implies x \notin L_{D1} \wedge x \notin L_{D2} \implies M_1$ rejects $\wedge\, M_2$ rejects $\implies M$ rejects

Since M_1 and M_2 are deciders, they necessarily halt on all inputs. Thus, M' must also halt on all inputs. In addition, since M' accepts all $x \in (L_{D1} \cup L_{D2})$ and rejects all $x \notin (L_{D1} \cup L_{D2})$, it follows that M' decides $L_{D1} \cup L_{D2}$. As we have constructed a decider for $(L_{D1} \cup L_{D2})$, we can conclude that $(L_{D1} \cup L_{D2})$ is necessarily decidable.

Alternatively, we can prove using result from part (a) and logic. We know that the complement of a decidable language is always decidable (run the decider and return opposite). Thus, $\overline{L_{D1}}$ and $\overline{L_{D2}}$ are both decidable. Thus, $\overline{L_{D1}} \cap \overline{L_{D2}}$ is decidable by part (a). Then by DeMorgan's law, we now know that $\overline{L_{D1}} \cap \overline{L_{D2}} = \overline{(L_{D1} \cup L_{D2})}$ is decidable. Again since the complement of a decidable language is decidable, $\overline{\overline{(L_{D1} \cup L_{D2})}} = L_{D1} \cup L_{D2}$ is necessarily decidable.

(4 pts) (c) $L_C = L_{D1} \cap L_{U1}$

Solution: Sometimes. Consider the case where $L_{D1} = \Sigma^*$. Since $L_{D1} \cap L_{U2} = L_{U1}$, which is undecidable by hypothesis, $L_{D1} \cap L_{D2}$ is undecidable.

Now consider the case where $L_1 = \emptyset$. Since $L_{D1} \cap L_{U1} = L_{D1} = \emptyset$, which is trivially decidable.

(4 pts) (d) $L_D = L_{D1} \cup L_{U1}$

Solution: Sometimes. Consider the case where $L_{D1} = \Sigma^*$. Since $L_{D1} \cup L_{D2} = L_{D1} = \Sigma^*$, which is trivially decidable.

Now consider the case where $L_{D1} = \emptyset$. Since $L_{D1} \cup L_{D2} = L_{D2}$, which is undecidable by hypothesis, $L_{D1} \cup L_{D2}$ is undecidable.

(4 pts) (e) $L_E = L_{U1} \cap L_{U2}$

Solution: Sometimes. Consider the case where $L_{U1} = L_{U2}$. In this case $L_{U1} \cap L_{U2} = L_{U1}$, which is undecidable by hypothesis.

Now let L_{U1} be some undecidable language and let $L_{U2} = \overline{L_{U1}}$ be its complement. So, both L_{U1} and L_{U2} are undecidable. Then $L_{U1} \cap L_{U2} = \emptyset$, which is decidable (by a decider which just ignores its input and rejects).

(4 pts) (f) $L_F = L_{U1} \cup L_{U2}$

Solution: Sometimes. Consider the case where $L_{U1} = L_{U2}$. In this case $L_{U1} \cup L_{U2} = L_{U1}$, which is undecidable by hypothesis.
Now, let $L_{U2} = \overline{L_{U1}}$. We know that this is undecidable from part (a). Then, $L_{U1} \cup L_{U2} = \Sigma^*$, which is trivially decidable (by a decider which just ignores its input and accepts).

7. **Can You Decide?**

For each of the following language, determine with proof whether the language is decidable or not. If decidable, describe and analyze a Turing machine that decides it. If undecidable, prove that reduce it from some language L of your choice that we have previously proved is undecidable.

(6 pts) (a) $L_{\text{ODD-5}} = \{x \in \{0, 1, 2, 3, 4\}^* : x \text{ represents an odd number in base } 5\}$. (L11)

Solution: The language $L_{\text{ODD-5}}$ is decidable. The key insight to see whether a number in odd base is odd, is making sure the count of odd digits is actually odd. We can construct a decider D as follows:

```
function D(x)
    count ← 0
    for digit in $x$ do
        if digit is odd then
            count ← count + 1
    if count is odd then
        accept
    else
        reject
```

If $x \in L_{\text{ODD-5}} \implies x$ has odd number of odd digits $\implies$ *count* is odd $\implies D$ accepts x.
If $x \notin L_{\text{ODD-5}} \implies x$ has even number of odd digits $\implies$ *count* is even $\implies D$ rejects x.
The decider always halts. Hence D is a decider for $L_{\text{ODD-5}}$ and $L_{\text{ODD-5}}$ is a decidable language.

(6 pts) (b) $L_{\text{AND-LOOP}} = \{(\langle M \rangle, x, y) : M \text{ is a TM that loops on input } x \text{ and loops on input } y\}$ (L11)

Solution: We will show $L_{\text{HALT}} \leq_{\text{T}} L_{\text{AND-LOOP}}$ here. Given an oracle $M_{\text{AND-LOOP}}$ for $L_{\text{AND-LOOP}}$, construct a machine M_{HALT} that decides L_{HALT} as follows: $M_{\text{HALT}}(\langle M \rangle, x)$

simply calls $M_{\text{AND-LOOP}}(\langle M\rangle, x, x)$ and outputs the opposite answer.

Suppose $(\langle M\rangle, x) \in L_{\text{HALT}}$. Then $M(x)$ does not loop. So, $M_{\text{AND-LOOP}}(\langle M\rangle, x, x)$ rejects. Then because $M_{\text{HALT}}(\langle M\rangle, x)$ returns the opposite answer, it accepts, as needed.

Now suppose $(\langle M\rangle, x) \notin L_{\text{HALT}}$. Then $M(x)$ loops. So, $M_{\text{AND-LOOP}}(\langle M\rangle, x, x)$ accepts. Then because $M_{\text{HALT}}(\langle M\rangle, x)$ returns the opposite answer, it rejects, as needed.

(6 pts) (c) Consider a language $L_{\text{I'MFINITE}}$ that contains a finite number of strings. Is $L_{\text{I'MFINITE}}$ (always/ sometimes/ never) decidable? Justify your answer. (L11)

Solution: $L_{\text{I'MFINITE}}$ is **always** decidable. Let $L_{\text{I'MFINITE}} = \{\ell_1, \ldots, \ell_n\}$ be any finite language. We can write a decider $D_{L_{\text{I'MFINITE}}}$ for $L_{\text{I'MFINITE}}$ as follows.

```
$L_{\text{I'MFINITE}}$ = on input $(x)$:
1: if $x = \ell_1$ or $x = \ell_2$ or $\ldots x = \ell_n$ then
2:     accept
3: else
4:     reject
```

We claim that this pseudocode can be implemented as a Turing machine, where each comparison of x against some $\ell_i \in L_{\text{I'MFINITE}}$ is performed using a finite number of states, and thus there are a finite number of states overall. More specifically, for each $\ell_i \in L_{\text{I'MFINITE}}$ we have a finite sequence of states that compare each cell of the tape to the corresponding character of ℓ_i, and finally compare the next cell to the blank symbol (to check that the end of the input string has been reached). This can be done with one state per character to be compared, where a match transitions to the next state (or to the accept state if all characters match), and a non-match transitions to a state that moves the head back to the start of the input, then transitions to the next comparison (or to the reject state if there are no more strings to be compared against). Essentially, all this "hard-codes" the strings of L into the states and transition function of the machine.

That M is a decider for L is clear by inspection: if $x \in L_{\text{I'MFINITE}}$ then $x = \ell_i$ for some $\ell_i \in L_{\text{I'MFINITE}}$, so one of the comparisons in the first line will cause the machine to accept. If $x \notin L$, then none of the comparisons in the first line will cause the machine to accept, and hence it will reject on the next line.

It is important to realize that the above strategy would *not* work if L was an *infinite* language. (Indeed, if it did, then every language would be decidable!) One reason is that the "code" (states and transitions) for a Turing machine needs to be finite, so we can't hard-code infinitely many strings into it. Furthermore, even if there was a way for a Turing machine to *generate* the elements of L in some sequence (and it turns out that this is possible for any *recognizable* language), we can't check x against an infinitely long sequence of strings in finite time. Specifically, if $x \notin L$, then the machine would not halt because it would never stop comparing x against the sequence of strings in L.

(d) *This part is optional and ungraded for this homework, but it's* **within the scope of the midterm**. *Therefore, we highly encourage you to attempt the problem.*

Define the language

$$L_{\text{FINITE}} = \{\langle M \rangle : M \text{ is a Turing machine and } L(M) \text{ is finite}\}.$$

Determine whether L_{FINITE} is decidable or undecidable. If decidable, describe and analyze a Turing machine that decides it. If undecidable, prove that $L \leq_T L_{\text{FINITE}}$ for some language L of your choice that we have previously proved is undecidable. (L12)

Solution:

L_{FINITE} is undecidable. Here, we show that $L_{\text{HALT}} \leq_{\text{T}} L_{\text{FINITE}}$. Let $D_{L_{\text{FINITE}}}$ be a "black box" (oracle) that decides L_{FINITE}, we construct a Turing Machine $D_{L_{\text{HALT}}}$ that uses $D_{L_{\text{FINITE}}}$ to decide L_{HALT}

```
$D_{L_{\text{HALT}}}$ = on input $(\langle M \rangle, x)$):
  1: Construct a machine $M'$ as follows:
       $M'$ = on input $\langle w \rangle$:
         1: Run $M(x)$
         2: accept
  2: Run $D_{L_{\text{FINITE}}}$ on input $\langle M' \rangle$
  3: if $D_{L_{\text{FINITE}}}(\langle M' \rangle)$ accepted then reject
  4: else accept"
```

First, we verify that $D_{L_{\text{HALT}}}$ accepts all strings in the language L_{HALT}:

$$\begin{aligned}(\langle M \rangle, x) \in L_{\text{HALT}} &\implies M \text{ halts on } x \text{ in a finite number of steps} \\ &\implies \text{for all } w \in \Sigma^*,\ M' \text{ accepts } w \\ &\implies L(M') = \Sigma^*, \text{ which is \textit{not} finite} \\ &\implies D_{L_{\text{FINITE}}} \text{ rejects } \langle M' \rangle \\ &\implies D_{L_{\text{HALT}}} \text{ accepts } (\langle M \rangle, x).\end{aligned}$$

Next, we verify that D rejects all strings not in the language L_{HALT}:

$$\begin{aligned}(\langle M \rangle, x) \notin L_{\text{HALT}} &\implies M \text{ loops on } x \\ &\implies \text{for all } w \in \Sigma^*,\ M' \text{ loops on } w \\ &\implies L(M') = \emptyset, \text{ which \textit{is} finite} \\ &\implies D_{L_{\text{FINITE}}} \text{ accepts } \langle M' \rangle \\ &\implies D_{L_{\text{HALT}}} \text{ rejects } (\langle M \rangle, x).\end{aligned}$$

Therefore, $D_{L_{\text{HALT}}}$ is a decider for L_{HALT}, which implies that $L_{\text{HALT}} \leq_{\text{T}} L_{\text{FINITE}}$. But L_{HALT} is undecidable, so we conclude that L_{FINITE} is also undecidable.