Solutions to Homework 6

This homework has 10 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*.

(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.)

If you didn't do last week's homework, choose a problem from it that looks challenging to you, and in a few sentences, explain the key ideas behind its solution in your own words.

Solution: My answer for 3.b. lost points due to an incorrect explanation of the validity of OPT', and incorrect justification of why item (ii) implies that my algorithm is correct. My solution for 3.b. is as follows:

(i) If ALG = OPT, then the algorithm is optimal. Otherwise, let i_{diff} be the first index in ALG that is not in OPT.
Let j be the first refill station in OPT occurring after the last common refill with ALG, ($j = \text{OPT}\{i_{diff}\}$).

$$\text{OPT}' = (\text{OPT} \setminus \{j\}) \cup \{i_{\text{diff}}\}.$$

OPT$'$ contains all indices $i_1, \ldots, i_{\text{diff}}$.

(ii) Upon arriving at i_{diff}, the remaining water is insufficient to cover the next segment. Thus, refilling at i_{diff} is necessary. Since any remaining water at j would be without use in the event of a refill, replacing j with i_{diff} still provides a full bottle at the right time. Since this exchange involves only one station and does not change the total number of refills, $|\text{OPT}'| = |\text{OPT}|$, so OPT$'$ is optimal.

(iii) Since any optimal solution OPT can be transformed via ALG exchanges, it must be the case that ALG uses no more refills than OPT. Therefore, the greedy algorithm ALG is correct.

A correct solution for this problem might look like showing that because i_{diff} is the vertex with the largest difference to $i_{\text{diff - 1}}$ such that the distance is less than R, $j < i_{\text{diff}}$, and then that the difference from i_{diff} to the next vertex in OPT' (really, the next vertex in OPT) is at most R, and thus OPT' is valid. Also, it would employ an exchange argument to show that the algorithm is correct.

2. **Turing machines warmup.**

Consider this Turing machine at turingmachine.io (follow the link) over the input alphabet $\Sigma = \{0, 1, \#\}$ and tape alphabet $\Gamma = \Sigma \cup \{\text{x}, \perp\}$. Note that the site uses $\llcorner$ for the 'blank' character, whereas we use $\perp$.

(2 pts) (a) Run the machine on the following input strings, and list all the ones it accepts. Fell free to simulate it on https://turingmachine.io/. You do not have to justify your answer.

- ε
- 010
- 10#10
- 100#1
- 101#101
- 110#001

Solution:

- 10#10
- 101#101

(4 pts) (b) State, with brief justification, what language the Turing machine decides, or that it does not decide any language.

Solution: The Turing machine decides the language $\{w\#w : w \in \{0,1\}^*\}$. For index i in the string to the left of the #, the machine removes $\textsc{Left}(i)$, and writes an x in the ith position in the string to the right of the # if the characters at those positions are the same. If they are not the same, the machine rejects. If the result is a # followed by a string of x's, the machine accepts. Otherwise, it rejects.

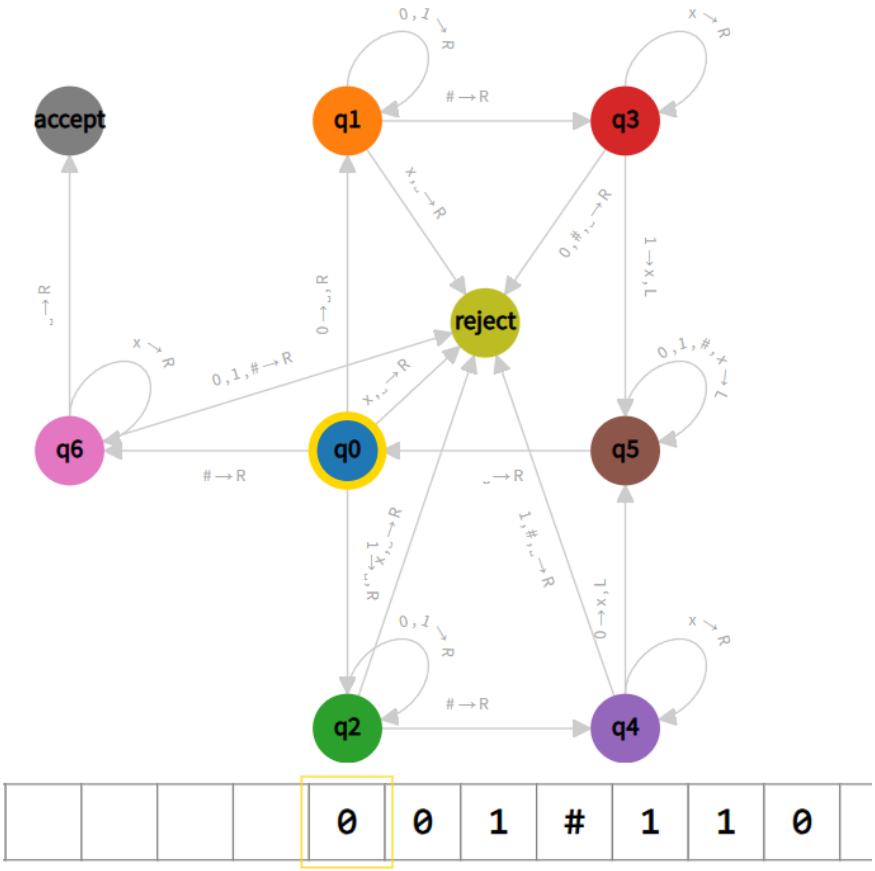
(2 pts) (c) A *bitwise complement* of a binary string is the result of flipping each bit, i.e., every 0 is replaced with 1 and vice versa. For example, the bitwise complement of 101 is 010.

Consider the following language:

$$L_{\text{BIT-COMP}} = \{b\#\bar{b} : b \in \{0,1\}^* \text{ and } \bar{b} \text{ is the bitwise complement of } b\}.$$

Modify the transitions of the Turing machine above to decide $L_{\text{BIT-COMP}}$. Include the screenshot of the modified Turing machine in your solution. You do not need to justify your answer.

Solution: The modifications to the Turing Machine are making it so that if q3 sees a 0, it rejects, and if q4 sees a 1, it rejects. Also, if q3 sees a 1, it writes an x, and if q4 sees a 0, it writes an x. The modified Turing Machine is shown below:



3. **Decidability and set operations.**

Let L be an *undecidable* language over the alphabet Σ. Recall that this means the complement language $\overline{L}$ is also undecidable, because we have shown that a language is decidable if and only if its complement is decidable. (In brief, the proof works by swapping the accept and reject states of any Turing machine that decides the language.)

(2 pts) (a) For each of following statements, give a *decidable* language L_D that makes the statement true.

(No justification is needed; the language L_D can vary from one statement to the next.)

1. $L \cup L_D$ is decidable.
2. $L \cup L_D$ is undecidable.
3. $L \cap L_D$ is decidable.
4. $L \cap L_D$ is undecidable.

Solution:

1. Let $L_D = \Sigma^*$. Then, $L \cup L_D = \Sigma^*$ is decidable.
2. Let $L_D = \varnothing$. Then, $L \cup L_D = L$ is undecidable.
3. Let $L_D = \varnothing$. Then, $L \cap L_D = \varnothing$ is decidable.
4. Let $L_D = \Sigma^*$. Then, $L \cap L_D = L$ is undecidable.

(2 pts) (b) For each of following statements, give an *undecidable* language L_U that makes the statement true.

(No justification is needed; the language L_U can vary from one statement to the next.)

1. $L \cup L_U$ is decidable.
2. $L \cup L_U$ is undecidable.
3. $L \cap L_U$ is decidable.
4. $L \cap L_U$ is undecidable.

Solution:

1. Let $L_U = \overline{L}$. Then, $L \cup L_U = \Sigma^*$ is decidable.
2. Let $L_U = L$. Then, $L \cup L_U = L$ is undecidable.
3. Let $L_U = \overline{L}$. Then, $L \cap L_U = \varnothing$ is decidable.
4. Let $L_U = L$. Then, $L \cap L_U = L$ is undecidable.

(6 pts) 4. **Undecidable within the infinite.**

Prove, using diagonalization, that any *infinite* language $L \subseteq \Sigma^*$ has an *undecidable* subset.

Hint: You may use the fact that any subset T of a countable set S is itself countable.

Solution: Since L is infinite, enumerate its elements as $w_1, w_2, w_3, \ldots$.
Let $\{M_1, M_2, M_3, \ldots\}$ be an enumeration of all deciders for decidable languages.
Define:

$$L' = \{\, w_i \in L \mid M_i \text{ does not accept } w_i \,\}.$$

Assume, for contradiction, that L' is decidable by some decider M_k. Then

$$w_k \in L' \iff M_k \text{ does not accept } w_k$$
$$w_k \notin L' \iff M_k \text{ accepts } w_k.$$

Based on the definition of M_k being a decider for L', if w_k is in L', then M_k should accept w_k, and if w_k is not in L', then M_k should not accept w_k. This does not happen, and thus is a contradiction.
Therefore, L' is undecidable.

Question 5 through Question 8 contain content covered in Lecture 11 on 2/17.

Reminder: Turing machines are computationally equivalent to computer programs, so treat "give a Turing machine" as "give an algorithm," except possibly providing a halting analysis instead of a runtime analysis when necessary.

(8 pts) 5. **Alternative L_{HALT} undecidability.**

Recall the language

$$L_{\text{HALT}} = \{(\langle M \rangle, x) : M \text{ is a TM and } M \text{ halts on } x\}.$$

In this problem, you will derive a proof that L_{HALT} is undecidable without relying on the undecidability of any other language. This was the first existence proof for an undecidable language, and it is due to Alan Turing.

Suppose for the sake of contradiction that some Turing machine H decides L_{HALT}. Give and analyze a Turing machine B which takes (the description of) a Turing machine as input, and has access to H as an oracle, that shows that the original hypothesis (that L_{HALT} is decidable) must be false.

Hint: A Turing machine may be designed to loop indefinitely.

Solution: Turing machine B, on input $\langle M \rangle$, operates as follows:

1. Run H on the input $(\langle M \rangle, \langle M \rangle)$.
2. If H accepts (M halts on $\langle M \rangle$), then loop indefinitely.
3. If H rejects (M does not halt on $\langle M \rangle$), then halt.

Consider the behavior of B on its own description $\langle B \rangle$:

- If $B(\langle B \rangle)$ halts, then H must have rejected on $(\langle B \rangle, \langle B \rangle)$, implying that $B(\langle B \rangle)$ does not halt.
- If $B(\langle B \rangle)$ does not halt, then H must have accepted on $(\langle B \rangle, \langle B \rangle)$, implying that $B(\langle B \rangle)$ halts.

In both cases, a contradiction is reached, and therefore, the assumption that L_{HALT} is decidable must be false.

6. **Turing reduction warmup.**

(4 pts) (a) Prove or disprove: Every language is Turing-reducible to its complement, i.e., $L \leq_T \overline{L}$ for any language L.

> **Solution:** Consider a Turing machine $M^{\overline{L}}$ that, on input x, queries the oracle for $\overline{L}$ on x:
>
> if $x \in \overline{L}$ then reject; otherwise, accept.
>
> Since, for this machine, $x \in L$ if and only if $x \notin \overline{L}$, $L \leq_T \overline{L}$.

(4 pts) (b) Prove or disprove: A decidable language is Turing reducible to any language, i.e., $L \leq_T L'$ for any decidable language L and any language L'.

(A disproof could consist of specific languages L, L' and a proof that the statement does not hold for them.)

> **Solution:** Let L be decidable and D be a decider for L. Consider a Turing machine that on input x uses the oracle to simulate $D(x)$, but ignores any queries to the oracle for L'. With this Turing machine, $L \leq_T L'$.

(4 pts) (c) Prove or disprove: let L be a language; if there exists a language L' such that $L \leq_T L'$, then L is decidable.

(A disproof could consist of specific languages L and L' where $L \leq_T L'$ but L is undecidable.)

> **Solution:** Let $L = L_{\text{HALT}}$, which is undecidable. Since every language is trivially Turing-reducible to itself ($L_{\text{HALT}} \leq_T L_{\text{HALT}}$), there exists a language L' such that $L \leq_T L'$ while L is undecidable. Thus, $L \leq_T L'$ does not imply that L is decidable.

If you haven't already, please read Handout 5: Turing Reductions and apply it in your solution for the next few problems.

7. **Professor ϒ's reductions.**

Professor ϒ has given you several statements and claimed proofs concerning Turing reductions, but every claimed proof has one major logical error! Identify and briefly explain each of these errors. You do not need to repair the proofs when that's possible, nor determine whether the statements are actually true, though these are good exercises too!

(5 pts) (a) **Statement:** Let $A = \{0^k 1^k : k \geq 0\}$ over $\Sigma = \{0, 1\}$, and let $B = a^*b^*$ over $\Sigma = \{a, b\}$. Then A Turing-reduces to B.

Claimed Proof: Let D_B be an oracle that decides B. Consider the following Turing machine D_A that has access to D_B.

```
1: function D_A(x) // x = x_1 x_2 ··· x_n
2:     for i = 1, ..., n do
3:         if x_i = 0 then w_i ← a
4:         else w_i ← b
5:     return D_B(w)
```

Clearly, D_A halts on any input because x has finite length, and D_B halts on any input by assumption. By the definitions of A and B, the code of D_A, and the hypothesis on D_B,

$$\begin{aligned} x \in A &\iff w \in \{a^k b^k : k \geq 0\} \\ &\iff w \in B \text{ (since 'star' means "zero or more")} \\ &\iff D_B(w) \text{ accepts} \\ &\iff D_A(x) \text{ accepts.} \end{aligned}$$

So, we have shown that D_A decides A, and hence $A \leq_T B$.

Solution: The proof equates $\{a^k b^k : k \geq 0\}$ with a^*b^*, which is incorrect. While every string of the form $a^k b^k$ is in a^*b^*, the converse is false, for example: aab is in a^*b^* but not of the form $a^k b^k$. Thus, the reduction does not correctly preserve inclusion in language A.

(5 pts) (b) **Statement:** L_{ACC} Turing-reduces to *any* language L where $L \neq \emptyset$ and $L \neq \Sigma^*$.

Claimed Proof: Let L be any such language and let D_L be an oracle that decides L. Consider the following Turing machine D_A that has access to D_L.

```
1: function D_A(⟨M⟩, x)
2:     if M(x) accepts then
3:         let w be an arbitrary string in L
4:     else
5:         let w be an arbitrary string not in L
6:     return D_L(w)
```

By the definition of L_{ACC}, the code of D_A, and the fact that D_L decides L, we have that

$$\begin{aligned}(\langle M\rangle, x) \in L_{\text{ACC}} &\iff M(x) \text{ accepts} \\ &\iff w \in L \\ &\iff D_L(w) \text{ accepts} \\ &\iff D_A(\langle M\rangle, x) \text{ accepts.}\end{aligned}$$

Thus, D_A decides L_{ACC}, hence we have shown that $L_{\text{ACC}} \leq_T L$.

Solution: The reduction requires knowledge on whether $M(x)$ accepts ("if $M(x)$ accepts then") to choose w. But, deciding whether $M(x)$ accepts is the undecidable problem L_{ACC}. This proof relies on solving an undecidable problem to construct its reduction. Thus, the given proof is incorrect.

8. **Loop and loop.**

(6 pts) (a) Define

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

Prove that $L_{\text{HALT}} \leq_T L_{\text{AND-LOOP}}$. Note that since L_{HALT} is undecidable, this means that $L_{\text{AND-LOOP}}$ is undecidable.

Solution: Given an instance $(\langle M\rangle, x)$ of L_{HALT}, define a Turing machine M' that on every input w does:

1. Simulate M on x.
2. If the simulation halts, then halt; otherwise, loop forever.

Let $y_1 = y_2 = 0$ (a fixed string). Notice that:

(M' loops on both y_1 and y_2) if and only if (M does not halt on x).

By running $L_{\text{AND-LOOP}}$ with $(\langle M'\rangle, y_1, y_2)$, we decide whether M halts on x. Therefore, $L_{\text{HALT}} \leq_T L_{\text{AND-LOOP}}$.

(6 pts) (b) Define

$$L_{\text{SELF-LOOP}} = \{\langle M\rangle : M \text{ is a TM that loops on input } \langle M\rangle\}.$$

Prove that $L_{\text{BARBER}} \leq_T L_{\text{SELF-LOOP}}$. Note that because L_{BARBER} is undecidable, we can conclude $L_{\text{SELF-LOOP}}$ is undecidable.

Solution: Recall that:

$$L_{\text{BARBER}} = \{\langle M\rangle : M \text{ does not accept } \langle M\rangle\}.$$

By the Recursion Theorem, given $\langle M\rangle$ we can construct a Turing machine M' with description $\langle M'\rangle$ that behaves as follows on input w:

$$M'(w) = \begin{cases} \text{Simulate } M \text{ on } \langle M\rangle; & \\ \quad \text{if } M \text{ accepts } \langle M\rangle, \text{ then halt;} & \\ \quad \text{otherwise, loop forever.} & \text{if } w = \langle M'\rangle; \\ \text{Halt immediately.} & \text{if } w \neq \langle M'\rangle \end{cases}$$

Then, M' loops on input $\langle M'\rangle$ if and only if M does not accept $\langle M\rangle$; that is,

$$\langle M\rangle \in L_{\text{BARBER}} \iff \langle M'\rangle \in L_{\text{SELF-LOOP}}.$$

Thus, $L_{\text{BARBER}} \leq_T L_{\text{SELF-LOOP}}$.

Question 9 and Question 10 contain content covered in Lecture 12 on 2/19.

9. **Finite and finite.**

For each of the following languages, determine whether it is decidable. If it is decidable, describe and analyze a Turing machine that describes it. Otherwise, prove its undecidability by reducing a known undecidable language to it.

(6 pts) (a) Let $L_{\text{I'M-FINITE}} = \{\ell_1, \ldots, \ell_n\}$ be a finite language, i.e., a language containing only a finite number of strings.

Determine, with proof, if $L_{\text{I'M-FINITE}}$ is decidable.

Solution: $L_{\text{I'M-FINITE}}$ is decidable. Consider a decider for $L_{\text{I'M-FINITE}}$, which operates as follows:

1. On input w, compare w with each string $\ell_1, \ell_2, \ldots, \ell_n$.
2. If $w = \ell_i$ for some i, accept; otherwise, reject.

Since there are only finitely many strings to compare, the machine halts on every input. Thus, $L_{\text{I'M-FINITE}}$ is decidable.

(8 pts) (b) Define the language

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

Determine, with proof, if L_{FINITE} is decidable.

Solution: Note that 0^n is a string of n zeros i.e, $0^3 = 000$.
L_{FINITE} is undecidable, as it is Turing-reducible to L_{ACC}. Given an instance $(\langle M \rangle, x)$ of L_{ACC}, construct a Turing machine M' as follows:

1. On input w, check if w is of the form 0^n; if not, reject.
2. If $w = 0^n$, simulate M on x for n steps.
3. If the simulation accepts within n steps, accept; otherwise, reject.

If $M(x)$ does not accept, then for every n the simulation fails and $L(M')$ is empty (finite). If $M(x)$ accepts, then there exists n_0 such that for all $n \geq n_0$, M accepts x within n steps, so M' accepts all strings of the form 0^n with $n \geq n_0$, making $L(M')$ infinite. Thus, deciding whether $L(M')$ is finite would decide L_{ACC}, which is undecidable. Hence, L_{FINITE} is undecidable.

10. **Turing reductions.**

For each of the following languages, show that it is undecidable via a Turing reduction from a language we have already shown is undecidable.

(8 pts) (a) $L_{\text{SUBSET}} = \{(\langle M_1 \rangle, \langle M_2 \rangle) : M_1, M_2 \text{ are TMs and } L(M_1) \subseteq L(M_2)\}$.
Hint: You may call an oracle multiple times.

Solution: Given an instance $(\langle M \rangle, x)$ of $\overline{L_{\text{ACC}}}$, construct TMs M_1 and M_2 as follows.

- Define M_2 to be the TM that rejects every input ($L(M_2) = \emptyset$).
- Define M_1 to operate on every input w by simulating M on x. If M ever accepts x, then M_1 accepts w if $w = 0$ (fixed string) and rejects otherwise; if M does not accept x, then M_1 never accepts any input.

Thus, if M does not accept x, then $L(M_1) = \emptyset \subseteq \emptyset = L(M_2)$; but if M accepts x, then $L(M_1) = \{0\} \not\subseteq \emptyset$. Therefore,

$$(\langle M \rangle, x) \in \overline{L_{\text{ACC}}} \iff (\langle M_1 \rangle, \langle M_2 \rangle) \in L_{\text{SUBSET}}.$$

Since $\overline{L_{\text{ACC}}}$ is undecidable, L_{SUBSET} is undecidable.

(8 pts) (b) $L_{\text{FASTER}} = \{(\langle M_1 \rangle, \langle M_2 \rangle, x) : M_1(x) \text{ runs for strictly fewer steps than } M_2(x) \text{ does}\}$.
Hint: A computation that halts takes strictly fewer steps than one that loops. If both computations loop, then neither takes strictly fewer steps than the other.

Solution: Given an instance $(\langle M \rangle, x)$ of L_{HALT}, construct TMs M_1 and M_2 and arbitrary string y as follows:

- Define M_1 on input y to simulate M on x. If $M(x)$ halts then M_1 halts; otherwise, M_1 loops forever.
- Define M_2 on input y to immediately loop forever.

Thus, if $M(x)$ halts, then $M_1(y)$ halts in a finite number of steps while $M_2(y)$ never halts, and hence $M_1(y)$ must run for strictly fewer steps than $M_2(y)$. If $M(x)$ does not halt, both $M_1(y)$ and $M_2(y)$ loop, and no one runs strictly fewer steps than the other. Thus,

$$(\langle M \rangle, x) \in L_{\text{HALT}} \iff (\langle M_1 \rangle, \langle M_2 \rangle, y) \in L_{\text{FASTER}}.$$

Since L_{HALT} is undecidable, L_{FASTER} is undecidable.