HW7 : Undecidability, Co-Recognizability, and Mapping Reductions **Sample Solutions**

CSE105Sp23

Assigned questions

1. **Properties of mapping reductions** (20 points):
   In the review quizzes, we saw that mapping reductions are transitive and are not symmetric. That is, if $A \leq_m B$ and $B \leq_m C$, then $A \leq_m C$ and there are sets A, B where $A \leq_m B$ but it is not the case that $B \leq_m A$.

   In this question, we'll explore other properties of mapping reductions. We fix the alphabet Σ and all sets we consider are languages over this alphabet.

   For each of the following statements, determine if it is true or false. Clearly label your choice by starting your solution with **True** or **False** and then provide a brief (3-4 sentences or so) justification for your answer.

   (a) (*Graded for correctness*) [1] Mapping reductions are *not* related to subset inclusion. That is, there are example sets A, B, C, D where $A \subseteq B$ and $A \leq_m B$ and $C \not\subseteq D$ and $C \leq_m D$. *Note: the notation $C \not\subseteq D$ means that C is not a subset of D. That is, there is an element of C that is not an element of D.*

   Solution: True.
   Case 1: $A \subseteq B$ and $A \leq_m B$
   Let $A, B \subseteq \Sigma^*$ any languages such that $A = B$ (which implies that $A \subseteq B$). The function that witnesses the reduction is the identity function $f(x) = x$ for all $x \in \Sigma^*$. From this, we get

   $$x \in A \iff f(x) \in A \iff f(x) \in B$$

   where we have used $x = f(x)$ and then $A = B$. Since the identity function is computable (e.g., a Turing machine which just immediately halts), we get that

[1] This means your solution will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

$A \leq_m B$ and $A \subseteq B$.

Case 2: $C \not\subseteq D$ and $C \leq_m D$
Let $C = \{1^n \mid n \geq 1\}$ and $D = \{1\}$.
C is infinite and D is finite, so $C \not\subseteq D$. The function that witnesses the reduction is

$$f(x) = \begin{cases} 1 & \text{if } x \in L(1^+) \\ 0 & \text{otherwise} \end{cases}$$

We can give a simple Turing machine F to demonstrate that this function is computable:

$F =$ "On input x :
1. If x is a string of one or more has 1's, output 1.
2. Else, output 0."

If $x \in C$, then x is a string of 1's (with at least one 1), so $f(x) = 1$, which is in D. If $x \notin C$, then x is not a string of one or more 1's, so $f(x) = 0$, which is not in D. We have $x \in C \iff f(x) \in D$, and so $C \leq_m D$. The key here is that C is regular (and therefore decidable), and so the indicator function f is computable.

(b) (*Graded for correctness*) For every decidable language L, there is a regular language R such that $L \leq_m R$.

Solution: True.
Let L be any decidable language. Therefore, there exists a Turing machine M that decides L. Using this Turing machine as subroutine, we define the Turing machine F which computes the indicator function of the language L as

$F =$ "On input x
1. Run M on input x.
2. If M accepts, output 1.
3. If M rejects, output 0."

We claim that the function f computed by F witnesses a reduction from L to $\{1\}$. Since $\{1\}$ is finite, it is regular. Therefore, we need to show the reduction is correct. It is computable since M is a decider. Also, if $x \in L$, then $f(x) = 1$ which is in $\{1\}$. If $x \notin L$, then $f(x) = 0$, which is not in $\{1\}$. This completes the reduction. Thus, for every decidable language L, there is a regular language R such that $L \leq_m R$.

(c) (*Graded for correctness*) Mapping reducibility is preserved under complement. That is, for all sets A and B, if $A \leq_m B$, then $\overline{A} \leq_m \overline{B}$.

Solution: True.
Let A and B be languages. By assumption, there is function f witnessing the reduction $A \leq_m B$. In particular, we have

$$x \in A \quad \text{if and only if} \quad f(x) \in B.$$

We claim that the same function f witnesses the reduction $\overline{A} \leq_m \overline{B}$.
First we show $x \in \overline{A}$ implies $f(x) \in \overline{B}$. If $x \in \overline{A}$, then $x \notin A$ by definition of the complement. But, using the property of the reduction f, $x \notin A$ implies $f(x) \notin B$. Once again, using the complement definition, we get $f(x) \in \overline{B}$.
Using the same sequence of implications, we can show that $x \notin \overline{A}$ implies $f(x) \notin \overline{B}$:

$$x \notin \overline{A} \implies x \in A \implies f(x) \in B \implies f(x) \notin \overline{B}.$$

Thus, we have shown that f (which is computable by assumption) also reduces from $\overline{A}$ to $\overline{B}$. For all sets A and B, if $A \leq_m B$, then $\overline{A} \leq_m \overline{B}$.

(d) (*Graded for correctness*) $A \leq_m B$ for every decidable language A and every co-recognizable language B. *Note: the definition of co-recognizable is from Week 8 and is: A language L over an alphabet Σ is called* **co-recognizable** *if its complement, defined as $\Sigma^* \setminus L = \{x \in \Sigma^* \mid x \notin L\}$, is Turing-recognizable.*

Solution: False.
Let A be some non-empty decidable language (e.g. $\{0\}$), and let B be the empty language.
The complement of B is Σ^*, which is recognizable (since it is regular, it is even decidable). Therefore B is co-recognizable.
To show that the statement is false, assume there is some function f reducing A to B. In particular, it must be the case that $x \in A \implies f(x) \in B$. However, there are *no* strings that are in B. In other words, $f(x) \notin B$ for all $x \in A$. Therefore, if there is some $x \in A$ (in our example, $x = 0$), then f cannot be a function witnessing the reduction. Therefore, the statement is false.

2. **What's wrong with these reductions?** (20 points):
Suppose your friends are practicing coming up with mapping reductions $A \leq_m B$ and their witnessing functions $f : \Sigma^* \to \Sigma^*$. For each of the following attempts, determine if it is has error(s) or is correct. Do so by labelling each attempt with all and only the labels below that apply, and justifying this labelling.

- *Error Type 1:* The given function can't witness the claimed mapping reduction because there exists an $x \in A$ such that $f(x) \notin B$.
- *Error Type 2:* The given function can't witness the claimed mapping reduction because there exists an $x \notin A$ such that $f(x) \in B$.

- *Error Type 3:* The given function can't witness the claimed mapping reduction because the specified function is not computable.
- *Correct:* The claimed mapping reduction is true and is witnessed by the given function.

Clearly present your answer by first listing all the relevant labels from above and then providing a brief (3-4 sentences or so) justification for each of those labels.

(a) (*Graded for correctness*) $A_{\text{TM}} \leq_m HALT_{\text{TM}}$ and

$$f(x) = \begin{cases} \langle \text{start} \to q_{\text{acc}} \rangle & \text{if } x = \langle M, w \rangle \text{ for a Turing machine } M \text{ and string } w \\ & \quad \text{and } w \in L(M) \\ \langle \text{start} \to q_0 \circlearrowright 0,1,\sqcup \to R \rangle & \text{otherwise} \end{cases}$$

Solution: Error 1 and 3.

Error 1. There exists an $x \in A$ such that $f(x) \notin B$.

Let $x = \langle M, w \rangle$ such that M is a Turing machine, w is a string and $w \in L(M)$. Clearly such strings exist (e.g., take M to be the TM whose start state is the accept state, and w to be any string). Therefore, $x \in A_{\text{TM}}$, so by the above definition of f we have,

$$f(x) = \langle \text{start} \to q_{\text{acc}} \rangle$$

We now claim that $f(x) \notin HALT_{\text{TM}}$. This is due to the fact that the output encodes only a Turing Machine. Since elements of $HALT_{\text{TM}}$ are encodings of a Turing machine and a word, the output cannot belong to $HALT_{\text{TM}}$.

For this same reason, this shows that for all $x \notin A_{\text{TM}}$, we have that $f(x) \notin HALT_{\text{TM}}$. Therefore, we do not have have error 2.

Error 3. f is not computable.

The function f has two cases. The first case triggers on input $x = \langle M, w \rangle$ whenever w is a string, M is a Turing machine, and $w \in L(M)$. Unfortunately, in order to decide if the input x belongs to this case, we should be able to decide whether $w \in L(M)$, which is same as checking if $\langle M, w \rangle$ belongs to A_{TM}. This is an undecidable problem, as proved in class. Hence, this function is not computable.

To formalize this idea, let's start by giving the two Turing machines names:

$$M_{acc} = \text{start} \to q_{\text{acc}} \qquad \text{and} \qquad M_{rej} = \text{start} \to q_0 \circlearrowright 0,1,\sqcup \to R$$

Suppose the function f were computable, we construct the following Turing ma-

chine which decides A_{TM}:

$M =$ "On input x :
1. Compute $f(x)$
2. If $f(x) = \langle M_{acc} \rangle$, accept
3. If $f(x) = \langle M_{rej} \rangle$, rej"

First, since f is computable and we can decide if two encodings are equivalent, M is a decider. Since $f(x) = \langle M_{acc} \rangle$ exactly when $x \in A_{\text{TM}}$, we have that M accepts x exactly when $x \in A_{\text{TM}}$. Therefore, M decides A_{TM}, which is a contradiction.

(b) (*Graded for correctness*) $\{ww \mid w \in \{0,1\}^*\} \leq_m \{w \mid w \in \{0,1\}^*\}$ and

$$f(x) = \begin{cases} w & \text{if } x = ww \text{ for a string } w \text{ over } \{0,1\} \\ \varepsilon & \text{otherwise} \end{cases}$$

Solution: **Error 2.** There exists an $x \notin A$ such that $f(x) \in B$.

Notice first that f is computable as shown by the TM F below:

$F =$ "On input $x = x_1 x_2 \cdots x_n$:
1. If x has odd length, output ε.
2. If $x_1 \cdots x_{n/2} = x_{n/2+1} \cdots x_n$, output $x_1 \cdots x_{n/2}$
3. Otherwise, output ε"

For this computable function to witness a reduction $A \leq_m B$, we must have that for every $x \in \Sigma^*$

$$x \in A \quad \text{if and only if} \quad f(x) \in B.$$

To show Error 2, we start with some string x which is not in $\{ww \mid w \in \{0,1\}^*\}$, lets say $x = 000$. By the definition of the function, $f(x) = \varepsilon$ which *is* in the language $B = \{w \mid w \in \{0,1\}^*\}$.
For completeness, we must show that Error 1 does not occur. Notice that $x \in A$ does imply that $f(x) \in B$ since $f(x)$ is always a string over the alphabet $\{0,1\}^*$ and $B = \{0,1\}^*$.

(c) (*Graded for correctness*) $EQ_{\text{TM}} \leq_m A_{\text{TM}}$ with

$$f(x) = \begin{cases} \langle \text{start} \rightarrow \boxed{q_{\text{acc}}}, M_w \rangle & \text{if } x = \langle M, w \rangle \text{ for a Turing machine } M \text{ and string } w \\ \varepsilon & \text{otherwise.} \end{cases}$$

Where for each Turing machine M, we define

$$M_w = \text{“On input y}$$
1. Simulate M on w.
2. If it accepts, accept.
3. If it rejects, reject.”

You may assume that ε is never a valid encoding and that encodings of pairs of Turing machines are never the same as encodings of a Turing machine and an input string (i.e., $\langle M_1, M_2\rangle \neq \langle M_3, w\rangle$).

Solution: **Error 1.** There exists an $x \in A$ such that $f(x) \notin B$. This is due to the fact that the reduction is the incorrect direction.

Notice first that the given function f is computable since checking that a string x is an encoding $\langle M, w\rangle$ of a TM M and a string w is decidable and creating the encoding for the Turing machine M_w is computable. (Crucial point: the function f is not actually simulating M_w. This would be problematic since M_w simulates M which might not be a decider.)

We claim that f always maps to strings outside of A_{TM}. To see this, let's consider two cases:
Case 1: $x = \langle M, w\rangle$. That is, x is a valid encoding of a Turing machine M and a string w. In this case, f maps x to the encoding of two different Turing machines $\langle M', M_w\rangle$ where M' is the Turing machine which accepts all strings. However, by assumption $\langle M_1, M_2\rangle \neq \langle M_3, w\rangle$ for any Turing machines M_1, M_2, M_3 and word w. Therefore, $f(x) \notin A_{\text{TM}}$.
Case 2: x not a valid encoding of a Turing machine M and string w. In this case, $f(x) = \varepsilon$. Once again, by assumption, we have that ε is never a valid encoding, and therefore $f(x) \notin A_{\text{TM}}$.

Therefore, take any string $x \in EQ_{\text{TM}}$ (this language is clearly not empty since the encodings of two identical TM's will be in the language). We get that $x \in EQ_{\text{TM}}$, but $f(x) \notin A_{\text{TM}}$, so f does not reduce from A to B.

On the other hand (to show we don't get Error 2), notice once again that $f(x) \notin A_{\text{TM}}$ for all x. In particular, this implies that for all $x \notin EQ_{\text{TM}}$, we have that $f(x) \notin A_{\text{TM}}$, so this aspect of the witnessing function is correct.

3. **Computational histories** (10 points):
At any point in the computation of a Turing machine, we can record what is going on by (metaphorically) taking a “snapshot”. We want this snapshot to contain all the information needed to simulate the rest of the computation. In particular, the snapshot encodes the

- Tape contents: Although the tape is infinite, at any specific point in a computation, only finitely many cells have been used. These are the only relevant tape contents to be encoded.
- Head position: An index to which position on the tape the head is currently pointing.
- State: An index to which state in the finite control the computation is currently at.

Notice that much like the encoding $\langle M \rangle$ of a Turing machine M, we can encode all of this snapshot information in a single string called a *configuration* (usually denoted by the letter C). In the same spirit of getter functions for components of encodings, all of the relevant information can effectively be extracted from the configuration. More formally, there is a computable function which computes each of the tape contents, head position, and state given a configuration as input. See Sipser Figure 3.4 (and surrounding discussion) for an explicit example of a configuration.

A computational history for Turing machine M on input w is sequence of configurations $C_1, C_2, \ldots, C_k$ such that configuration C_{i+1} results from taking one step in the Turing machine computation corresponding to C_i (in other words, one application of the transition function). Additionally, C_1 is the starting configuration, corresponding to the tape that has the characters w on the leftmost $|w|$-many cells of the tape, the tape head at the leftmost position, and the current state being the starting state of the Turing machine. We say that a computational history is *accepting* if the final configuration in the sequence C_k has the current state being the accept state of the Turing machine.

Let's suppose we can describe both the encodings of Turing machines and configurations using the alphabet $\Sigma = \{0, 1\}$. That is, $\langle M \rangle \in \Sigma^*$ and $C \in \Sigma^*$ for any Turing machine M and configuration of the Turing machine C. We define the language of accepting computational histories over the alphabet $\Gamma = \{0, 1, 2\}$:

$$H := \{\langle M \rangle 2 \langle w \rangle 2 C_1 2 \cdots 2 C_k \mid M \text{ is a Turing machine, } w \text{ is a string, } C_1, \ldots, C_k \text{ is the computational history of } M \text{ on } w \text{ and is accepting}\}$$

That is, strings in the language H start with an encoding of some Turing machine M, followed by an encoding of some string w, followed by an accepting computational history of M on input w. There is a 2 symbol between each of these components to serve as a delimiter. To be clear, each of these encodings is over the alphabet $\{0, 1\}$, but you may also assume that it's possible to decide whether or not a particular bit string is an encoding of a Turing machine, a configuration, or neither.

(a) (*Graded for correctness*) Give a high-level description for a Turing machine that decides H and justify why it works. Namely, prove that the Turing machine you define halts for each input and that it accepts an arbitrary string if and only if that string is in H.

Solution: The key to the entire construction is that checking that one configuration C_{i+1} follows (in one step) from the previous C_i is a decidable problem. For concreteness, let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ be the formal specification of Turing machine M. Suppose configuration C_i has tape contents $t_1, \ldots, t_m \in \Gamma$, head pointer at index $k \in \{1, \ldots, m\}$, and is at state $q \in Q$. Suppose configuration C_{i+1} has tape contents $t'_1, \ldots, t'_{m'}$, head pointer at index $k' \in \{1, \ldots, m'\}$, and is at state $q' \in Q$. Using the transition function of the Turing machine δ, let $\delta(q, t_k) = (r, a, D)$ where $r \in Q$, $a \in \Gamma$, $D \in \{L, R\}$. We have that C_{i+1} follows from C_i if the following hold (for simplicity, let's choose our configurations so that that $m' \geq m$; i.e., taking more steps in our Turing machine does not decrease the number of tape cells we use):

i. Tape contents match except at k: $t_1 = t'_1$, $t_2 = t'_2$, ..., except at position k where $t'_k = a$. For all t'_ℓ for $\ell > m$, $t'_\ell = \sqcup$ (this captures the fact that the next Turing machine configuration might want to use more space). If $k = m$ and $D = R$, we must have that $m' > m$.

ii. Head pointer has been updated: $k' = \max\{1, k-1\}$ if $D = L$, otherwise, $k' = k + 1$ since $D = R$.

iii. State has been updated: $q' = r$.

Since there is a computable function which gives all of this information (tape content, head pointer, state) from a given configuration. This entire check is decidable.

We also need subroutines to check that C_1 is a starting configuration for w and that C_k is an accepting configuration. To check that C_1 is starting, we simply must check that w is left-aligned on the tape, that the state of computation is q_0, and that the head is pointed to the first tape symbol. To check that C_k is accepting, we simply need to check that the state is q_{acc}. Once again, both tasks are decidable.

We can now define a Turing machine D which decides H:

$D =$ "On input x :

1. Check that x is of the form $\langle M \rangle 2 \langle w \rangle 2 C_1 2 \cdots 2 C_k$. If not, reject.
2. Check that C_1 is the starting configuration for M on w. If not, reject.
3. For $i = 1$ to $k - 1$:
4. Check that C_{i+1} follows from C_i.
5. Check that C_k is an accepting configuration. If yes, accept. Otherwise, reject."

By construction, H halts and accepts exactly those sequences of configurations which lead to M accepting w. Since each step of D is decidable, D itself is a decider.

(b) (*Graded for completeness*) [2] Prove that SUBSTRING(H) is undecidable by showing a mapping reduction from A_{TM}. That is, you will prove that $A_{\text{TM}} \leq_m$ SUBSTRING(H) by giving a witnessing function. Recall that for any language $K \subseteq \Gamma^*$, we define

$$\text{SUBSTRING}(K) := \{w \in \Gamma^* \mid \text{there exist } a, b \in \Gamma^* \text{ such that } awb \in K\}.$$

Combining parts (a) and (b), notice that this implies that the class of decidable languages is not closed under the SUBSTRING operation.

Solution: Let $C_{M,w} \in \Sigma^*$ be the starting configuration of Turing machine M on input w. Define $f\colon \Gamma^* \to \Gamma^*$ as

$$f(x) = \begin{cases} \langle M \rangle 2 \langle w \rangle 2 C_{M,w} 2 & \text{if } x = \langle M, w \rangle \\ \langle M_{none} \rangle 2 \langle \varepsilon \rangle 2 C_{M_{none},\varepsilon} 2 & \text{otherwise} \end{cases}$$

where M_{none} is a Turing machine which rejects all inputs (e.g., the starting state is the reject state).

First, since f is simply taking encodings of M and w and creating the initial configuration, f is decidable. Let's show the two directions:

$x \in A_{TM} \implies f(x) \in$ SUBSTRING(H):
For any $\langle M, w \rangle$ in A_{TM}, we know there is a sequence of configurations which will lead to accept state. Therefore, $\langle M \rangle 2 \langle w \rangle 2 C_{M,w} 2$ will be a valid member of SUBSTRING(H). That is, we can append other configurations to it to the get the complete computational history of the string, hence this is a valid substring. In this way, we can map every input from A_{TM} to a value in SUBSTRING(H).

$x \notin A_{TM} \implies f(x) \notin$ SUBSTRING(H):
For the values which are not in A_{TM}, the function f outputs the concatenated encoding (separated by 2's) of a machine M, a string w, and an initial configuration C, for which the machine M does not accept the string w. If x is not a valid encoding of a machine and a string, this happens by design. If x is a valid encoding, it happens by assumption that $x \notin A_{TM}$.

Notice that since we already have $\langle M \rangle$ and the beginning of our string, we cannot prepend any more symbols to get a string in H. (Here we are implicitly assuming that it's not possible to extend an encoding of a machine M into an encoding of a different machine M' by prepending symbols to it). Therefore, we must append characters to our string if we want it to be in SUBSTRING(H). However, M does not accept w, so there cannot be a valid computational history for M on w ending in an accepting configuration. Hence, this string will not exist in SUBSTRING(H)

[2] This means you will get full credit so long as your submission demonstrates honest effort to answer the question. You will not be penalized for incorrect answers. To demonstrate your honest effort in answering the question, we ask that you include your attempt to answer *each* part of the question. If you get stuck with your attempt, you can still demonstrate your effort by explaining where you got stuck and what you did to try to get unstuck.

as there is no string that could be appended to the substring that would end in an accept state.