CSE 105 - Theory of Computation (Winter 2026)
Homework 7: Undecidability, Co-Recognizability, and Mapping Reductions
Due Wednesday, March 4, 11:59pm

Instructions: There are two parts to this homework:

- Concept check (Question 1): Every student must complete this individually on Gradescope.
- Written Homework (Question 2-4): You may work individually or in a team of up to 3 people. Please ensure your name(s) and PID(s) are clearly visible on the first page of your submission, and then upload the PDF to Gradescope. If working in a group, submit only one submission per group: one partner uploads the submission through their Gradescope account and then adds the other group member to the Gradescope submission by selecting their name in the "Add Group Members" dialog box. You will need to re-add your group member every time you resubmit a new version of your assignment.

It is highly recommended (though not required) that you type your answers. It is your responsibility to make any handwriting clear and legible for grading. A LaTeX template for the homework is provided on Canvas. For ease of grading, please start each new problem on a separate page.

We will only be grading some of the problems below for correctness. However, because all of the concepts are important, we will not reveal which problems are being graded for correctness until after the assignment has been submitted. The remaining problems will be graded for completeness (i.e., does it look like there was a good-faith effort to solve the problem?).

Reading and extra practice problems: Chapter 5 exercises 5.4, 5.5, 5.6, 5.7. Chapter 5 problems 5.10, 5.11, 5.16.

Problems:

1. **Concept check**

   Complete the assignment "Homework 7 - Concept Check" on Gradescope.

2. **Properties of mapping reductions**

   Fix some alphabet Σ for all languages in this problem. 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) 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.

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

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

(d) $A \leq_m B$ for every decidable language A and every co-recognizable language B.

Note: A language L over an alphabet Σ is co-recognizable *if its complement, defined as $\Sigma^* \setminus L = \{x \in \Sigma^* \mid x \notin L\}$, is Turing-recognizable.*

3. **What's wrong with these reductions?**

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 labeling each attempt with all the labels below that apply, and justifying this labeling.

- *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) $A_{\text{TM}} \leq_m HALT_{\text{TM}}$ and

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

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

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

(c) $EQ_{\text{TM}} \leq_m A_{\text{TM}}$ with

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

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., $\langle M_1, M_2 \rangle \neq \langle M_3, w \rangle$).

4. **Computational histories**

At any point in the computation of a Turing machine, there is some finite size “snapshot” that records all of the relevant information about the computation of the Turing machine. 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 into the tape where the head is currently pointing.
- State: An encoding of the current state of the Turing machine.

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). Similarly, all of the relevant information can effectively be extracted from the configuration. More formally, there is a computable function that can output 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) 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.

(b) 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.

5. **Course evaluation**

Please fill out Student Evaluations of Teaching (SET) at this website (starting March 2):

academicaffairs.ucsd.edu/Modules/Evals.

This question is not graded since we cannot know or enforce participation. That said, we do really appreciate your feedback!