HW4CSE105W24: Homework assignment 4

CSE105W24

Due: February 29th at 5pm (no penalty late submission until 8am next morning), via Gradescope

In this assignment, You will practice analyzing, designing, and working with Turing machines. You will use general constructions and specific machines to explore the classes of recognizable and decidable languages. You will explore various ways to encode machines as strings so that computational problems can be recognized.

Resources: To review the topics for this assignment, see the class material from Weeks 5-7. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Sections 3.1, 3.3, 4.1 Chapter 3 exercises 3.1, 3.2, 3.5, 3.8. Chapter 4 exercises 4.1, 4.2, 4.3, 4.4, 4.5.

For all HW assignments: Weekly homework may be done individually or in groups of up to 3 students. You may switch HW partners for different HW assignments. Please ensure your name(s) and PID(s) are clearly visible on the first page of your homework 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(s) to the Gradescope submission by selecting their name(s) in the "Add Group Members" dialog box. You will need to re-add your group member(s) every time you resubmit a new version of your assignment. Each homework question will be graded either for correctness (including clear and precise explanations and justifications of all answers) or fair effort completeness. For "graded for correctness" questions: collaboration is allowed only with CSE 105 students in your group; if your group has questions about a problem, you may ask in drop-in help hours or post a private post (visible only to the Instructors) on Piazza. For "graded for completeness" questions: collaboration is allowed with any CSE 105 students this quarter; if your group has questions about a problem, you may ask in drop-in help hours or post a public post on Piazza.

All submitted homework for this class must be typed. You can use a word processing editor if you like (Microsoft Word, Open Office, Notepad, Vim, Google Docs, etc.) but you might find it useful to take this opportunity to learn LaTeX. LaTeX is a markup language used widely in

computer science and mathematics. The homework assignments are typed using LaTeX and you can use the source files as templates for typesetting your solutions. To generate state diagrams of machines, we recommend using Flap.js or JFLAP. Photographs of clearly hand-drawn diagrams may also be used. We recommend that you submit early drafts to Gradescope so that in case of any technical difficulties, at least some of your work is present. You may update your submission as many times as you'd like up to the deadline.

Integrity reminders

- Problems should be solved together, not divided up between the partners. The homework is designed to give you practice with the main concepts and techniques of the course, while getting to know and learn from your classmates.
- You may not collaborate on homework questions graded for correctness with anyone other than your group members. You may ask questions about the homework in office hours (of the instructor, TAs, and/or tutors) and on Piazza (as private notes viewable only to the Instructors). You *cannot* use any online resources about the course content other than the class material from this quarter – this is primarily to ensure that we all use consistent notation and definitions (aligned with the textbook) and also to protect the learning experience you will have when the 'aha' moments of solving the problem authentically happen.
- Do not share written solutions or partial solutions for homework with other students in the class who are not in your group. Doing so would dilute their learning experience and detract from their success in the class.

You will submit this assignment via Gradescope (https://www.gradescope.com) in the assignment called "hw4CSE105W24".

Assigned questions

1. **Classifying languages** (10 points): Our first example of a more complicated Turing machine was of a Turing machine that recognized the language $\{w\#w \mid w \in \{0,1\}^*\}$, which we know is not context-free. The language

$$\{0^n 1^n 2^n \mid n \geq 0\}$$

is also not context-free.

(a) (*Graded for correctness*) [1] Give an implementation-level description of a Turing machine that recognizes this language.

[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.

(b) (*Graded for completeness*) [2] Draw a state diagram of the Turing machine you gave in part (a) and trace the computation of this Turing machine on the input 012. You may use all our usual conventions for state diagrams of Turing machines (we do not include the node for the reject state $qrej$ and any missing transitions in the state diagram have value $(qrej, \square, R)$; $b \to R$ label means $b \to b, R$).

2. **Deciders, Recognizers, Decidability, and Recognizability** (15 points): For this question, consider the alphabet $\Sigma = \{0, 1\}$.

(a) (*Graded for correctness*) Give an example of a finite, nonempty language over Σ and two different Turing machines that recognize it: one that is a decider and one that is not. A complete solution will include a precise definition for your example language, along with **both** a state diagram and an implementation-level description of each Turing machines, along with a brief explanation of why each of them recognizes the language and why one is a decider and there other is not.

(b) (*Graded for correctness*) True or false: There is a Turing machine that is not a decider that recognizes the empty set. A complete solution will include a witness Turing machine (given by state diagram or implementation-level description or high-level description) and a justification for why it's not a decider and why it does not accept any strings, or a complete and correct justification for why there is no such Turing machine.

(c) (*Graded for correctness*) True or false: There is a Turing machine that is not a decider that recognizes the set of all string Σ^*. A complete solution will include a witness Turing machine (given by state diagram or implementation-level description or high-level description) and a justification for why it's not a decider and why it accept each string over $\{0, 1\}$, or a complete and correct justification for why there is no such Turing machine.

3. **Closure** (15 points): Suppose M is a Turing machine over the alphabet $\{0, 1\}$. Let $s_1, s_2, \ldots$ be a list of all strings in $\{0, 1\}^*$ in string (shortlex) order. We define a new Turing machine by giving its high-level description as follows:

M_{new} = "On input w :

1. For $n = 1, 2, \ldots$
2. For $j = 1, 2, \ldots n$
3. For $k = 1, 2, \ldots, n$
4. Run the computation of M on $s_j w s_k$
5. If it accepts, accept.
6. If it rejects, go to the next iteration of the loop"

[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 expect you to 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.

Recall the definitions we have: For languages L_1, L_2 over the alphabet $\Sigma = \{0, 1\}$, we have the associated sets of strings

$$SUBSTRING(L_1) = \{w \in \Sigma^* \mid \text{there exist } a, b \in \Sigma^* \text{ such that } awb \in L_1\}$$

and

$$L_1 \circ L_2 = \{w \in \Sigma^* \mid w = uv \text{ for some strings } u \in L_1 \text{ and } v \in L_2\}$$

We say that self-set-wise concatenation of the set L_1 is $L_1 \circ L_1$.

Note: there was a bug in the version of this assignment that was first released.

(a) (*Graded for completeness*) Prove that this Turing machine construction **cannot** be used to prove that the class of decidable languages over $\{0, 1\}$ is closed under **either** of the above operations ($SUBSTRING$ or self-set-wise concatenation). A complete answer will give a counterexample or general description why the construction doesn't work for both operations.

(b) (*Graded for correctness*) Prove that this Turing machine construction cannot be used to prove that the class of recognizable languages over $\{0, 1\}$ is closed under the $SUBSTRING$ set operation. In particular, give a counterexample of a specific language L_1 and Turing machine M_1 recognizing it where M_{new} does not recognize $SUBSTRING(L_1)$.

(c) (*Graded for completeness*) Define a new construction by slightly modifiying this one that can be used to prove that the class of recognizable languages over $\{0, 1\}$ is closed under $SUBSTRING$. Justify that your construction works. The proof of correctness for the closure claim can be structured like: "Let L_1 be a recognizable language over $\{0, 1\}$ and assume we are given a Turing machine M_1 so that $L(M_1) = L_1$. Consider the new Turing machine M_{new} defined above. We will show that $L(M_{new}) = SUBSTRING(L_1)$... *complete the proof by proving subset inclusion in two directions, by tracing the relevant Turing machine computations*"

4. **Computational problems** (10 points): Recall the definitions of some example computational problems from class

Acceptance problem		
... for DFA	A_{DFA}	$\{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$
... for NFA	A_{NFA}	$\{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$
... for regular expressions	A_{REX}	$\{\langle R, w\rangle \mid R \text{ is a regular expression that generates input string } w\}$
... for CFG	A_{CFG}	$\{\langle G, w\rangle \mid G \text{ is a context-free grammar that generates input string } w\}$
... for PDA	A_{PDA}	$\{\langle B, w\rangle \mid B \text{ is a PDA that accepts input string } w\}$
Language emptiness testing		
... for DFA	E_{DFA}	$\{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$
... for NFA	E_{NFA}	$\{\langle A\rangle \mid A \text{ is a NFA and } L(A) = \emptyset\}$
... for regular expressions	E_{REX}	$\{\langle R\rangle \mid R \text{ is a regular expression and } L(R) = \emptyset\}$
... for CFG	E_{CFG}	$\{\langle G\rangle \mid G \text{ is a context-free grammar and } L(G) = \emptyset\}$
... for PDA	E_{PDA}	$\{\langle A\rangle \mid A \text{ is a PDA and } L(A) = \emptyset\}$
Language equality testing		
... for DFA	EQ_{DFA}	$\{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$
... for NFA	EQ_{NFA}	$\{\langle A, B\rangle \mid A \text{ and } B \text{ are NFAs and } L(A) = L(B)\}$
... for regular expressions	EQ_{REX}	$\{\langle R, R'\rangle \mid R \text{ and } R' \text{ are regular expressions and } L(R) = L(R')\}$
... for CFG	EQ_{CFG}	$\{\langle G, G'\rangle \mid G \text{ and } G' \text{ are CFGs and } L(G) = L(G')\}$
... for PDA	EQ_{PDA}	$\{\langle A, B\rangle \mid A \text{ and } B \text{ are PDAs and } L(A) = L(B)\}$

(a) (*Graded for completeness*) Pick five of the computational problems above and give examples (preferably different from the ones we talked about in class) of strings that are in each of the corresponding languages. Remember to use the notation $\langle \cdots \rangle$ to denote the string encoding of relevant objects. *Extension, not for credit:* Explain why it's hard to write a specific string of 0s and 1s and make a claim about membership in one of these sets.

(b) (*Graded for completeness*) Computational problems can also be defined about Turing machines. Consider the two high-level descriptions of Turing machines below. Reverse-engineer them to define the computational problem that is being recognized, where $L(M_{DFA})$ is the language corresponding to this computational problem about DFA and $L(M_{TM})$ is the language corresponding to this computational problem about Turing machines. *Hint*: the computational problem is not acceptance, language emptiness, or language equality (but is related to one of them).

Let $s_1, s_2, \ldots$ be a list of all strings in $\{0,1\}^*$ in string (shortlex) order. Consider the

following Turing machines

M_{DFA} = "On input $\langle D \rangle$ where D is a DFA :

1. for $i = 1, 2, 3, \ldots$
2. Run D on s_i
3. If it accepts, accept.
4. If it rejects, go to the next iteration of the loop"

and

M_{TM} = "On input $\langle T \rangle$ where T is a Turing machine :

1. for $i = 1, 2, 3, \ldots$
2. Run T for i steps on each input $s_1, s_2, \ldots, s_i$ in turn
3. If T has accepted any of these, accept.
4. Otherwise, go to the next iteration of the loop"