HW5CSE105W25: Homework assignment 5

CSE105W25

Due: February 27th at 5pm, 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 and solved.

Resources: To review the topics for this assignment, see the class material from Weeks 6, 7, and 8. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Chapters 3 and 4. 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. On the "graded for correctness" questions, you may only collaborate 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. On the "graded for completeness" questions, you may collaborate with all other CSE 105 students this quarter, and you may make public posts about these questions 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, you can (1) use the LaTex tikzpicture environment (see templates in the class notes), or (2) use the software tools Flap.js or JFLAP described in the class syllabus (and include a screenshot in your PDF), or (3) you can carefully and clearly hand-draw the diagram and take a picture and include it in your PDF. 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.
- On the "graded for correctness" questions, you may only collaborate with CSE 105 students in your group. 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 "hw5CSE105W25".

Assigned questions

1. **Equally expressive models** (10 points): The **Church-Turing Thesis** (Sipser p. 183) says that the informal notion of algorithm is formalized completely and correctly by the formal definition of a Turing machine. In other words: all reasonably expressive models of computation are equally expressive with the standard Turing machine. In this question, we will give support for this thesis by showing that some adaptations of the standard (Chapter 3) Turing machine model still gives us a new model that is equally expressive.

(a) (*Graded for completeness*) [1] Let's define a new machine model, and call it the **May-stay** machine. The May-stay machine model is the same as the usual Turing machine model, except that on each transition, the tape head may move L, move R, or Stay.

[1]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.

Formally: a May-stay machine is given by the 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where Q is a finite set with $q_0 \in Q$ and $q_{accept} \in Q$ and $q_{reject} \in Q$ and $q_{accept} \neq q_{reject}$, Σ and Γ are alphabets and $\Sigma \subseteq \Gamma$ and $\square \in \Gamma$ and $\square \notin \Sigma$, and the transition function has signature

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

The notions of computation and acceptance are analogous to that from Turing machines.

Prove that Turing machines and May-stay machines are equally expressive. A complete proof will use the formal definitions of the machines.

Hint: Include two directions of implications. First, let M be an arbitrary Turing machine and prove that there's a May-stay machine that recognizes the language recognized by M. Next, let M_S be an arbitrary May-stay machine and prove that there's a Turing machine that recognizes the language recognized by M_S.

(b) (*Graded for correctness*) [2] Let's define a new machine model, and call it the **Double-move** machine. The Double-move machine model is the same as the usual Turing machine model, except that on each transition, the tape head may move L, move R one cell, or move R two cells. Formally: a Double-move machine is given by the 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where Q is a finite set with $q_0 \in Q$ and $q_{accept} \in Q$ and $q_{reject} \in Q$ and $q_{accept} \neq q_{reject}$, Σ and Γ are alphabets and $\Sigma \subseteq \Gamma$ and $\square \in \Gamma$ and $\square \notin \Sigma$, and the transition function has signature

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, T\}$$

where L means that the read-write head moves to the left one cell (or stays put if it's at the leftmost cell already), R means that the read-write head moves one cell to the right , and T means that the read-write head moves two cells to the right. The notion of computation and acceptance are analogous to that from Turing machines.

Prove that Turing machines and Double-move machines are equally expressive. A complete proof will use the formal definitions of the machines.

Hint: Include two directions of implications. First, let M be an arbitrary Turing machine and prove that there's a Double-move machine that recognizes the language recognized by M. Next, let M_D be an arbitrary Double-move machine and prove that there's a Turing machine that recognizes the language recognized by M_D.

(c) (*Graded for completeness*) In your proofs of equal expressivity in the previous parts of this question, you proved that a language is recognizable by some Turing machine if and only if it is recognizable by some May-stay machine or by some Double-move machine. Do your proofs also prove that a language is decidable by some Turing machine if and only if it is decidable by some May-stay machine or by some Double-move machine? Justify your answer.

[2] 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.

2. **Modifying machines** (12 points)

(a) (*Graded for correctness*) Suppose a friend suggests that the following construction can be used to prove that the class of decidable languages is closed under intersection.

Construction: given deciders M_1 and M_2 build the following machine M

```
$M$ = "On input $w$ :
  1. Run $M_1$ on input $w$.
  2. If $M_1$ accepts $w$, accept.
  3. Run $M_2$ on input $w$.
  4. If $M_2$ accepts $w$, accept.
  5. If $M_2$ rejects $w$, reject."
```

Build a counterexample that could be used to convince your friend that this construction doesn't work. A complete counterexample will include (1) a high-level description of M_1, (2) a high-level description of M_2, (3) a justification for why they provide a counterexample (that references the definitions of M , decidable languages, and intersection).

Ungraded bonus: Is it possible to change one line of the construction to make it work?

(b) (*Graded for correctness*) Suppose a friend suggests that the following construction can be used to prove that the class of recognizable languages is closed under intersection.

Construction: given Turing machines M_1 and M_2 build the following machine M'

```
$M'$ = "On input $w$ :
  1. Run $M_1$ on input $w$.
  2. If $M_1$ rejects $w$, reject.
  3. Run $M_2$ on input $w$.
  4. If $M_2$ rejects $w$, reject."
```

Build a counterexample that could be used to convince your friend that this construction doesn't work. A complete counterexample will include (1) a high-level description of M_1, (2) a high-level description of M_2, (3) a justification for why they provide a counterexample (that references the definition of M', recognizable languages, and intersection).

Ungraded bonus: Is it possible to change one line of the construction to make it work?

3. **Closure** (12 points):

For each language L over an alphabet Σ, we have the associated sets of strings (also over Σ)

$$L^* = \{w_1 \cdots w_k \mid k \geq 0 \text{ and each } w_i \in L\}$$

and

$$SUBSTRING(L) = \{w \in \Sigma^* \mid \text{there exist } x, y \in \Sigma^* \text{ such that } xwy \in L\}$$

and

$$EXTEND(L) = \{w \in \Sigma^* \mid w = uv \text{ for some strings } u \in L \text{ and } v \in \Sigma^*\}$$

(a) (*Graded for correctness*) Prove whether this Turing machine construction below **can** or **cannot** be used to prove that the class of recognizable languages over Σ is closed under the Kleene star operation or the $SUBSTRING$ operation or the $EXTEND$ operation.

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

```
$M_a$ = "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 $s_j w s_k$ for at most $n$ steps
5.       If that computation halts and accepts within $n$ steps, accept.
6.       Otherwise, continue with the next iteration of this inner loop"
```

A complete and correct answer will either identify which operation works and give the proof of correctness why, for any Turing machine M, $L(M_a)$ is equal to the result of applying this operation to $L(M)$; **or** give a counterexample (a recognizable set A and a Turing machine M recognizing A and a description of why $L(M_a)$ where M_a is the result of the construction applied to M doesn't equal A^* and doesn't equal $SUBSTRING(A)$ and doesn't equal $EXTEND(A)$.

(b) (*Graded for correctness*) Prove whether this Turing machine construction below **can** or **cannot** be used to prove that the class of recognizable languages over Σ is closed under the Kleene star operation or the $SUBSTRING$ operation or the $EXTEND$ operation.

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

```
$M_b$ = "On input $w$ :
1. For $n = 1, 2, \ldots$
2.   For $j = 0, \ldots, |w|$
3.     Let $u$ be the string consisting of the first $j$ characters of $w$
4.     Run the computation of $M$ on $u$ for at most $n$ steps
5.     If that computation halts and accepts within $n$ steps, accept.
6.     Otherwise, continue with the next iteration of this inner loop"
```

A complete and correct answer will either identify which operation works and give the proof of correctness why, for any Turing machine M, $L(M_b)$ is equal to the result of applying this operation to $L(M)$; **or** give a counterexample (a recognizable set B and a Turing machine M recognizing B and a description of why $L(M_b)$ where M_b is the result of the construction applied to M doesn't equal equal B^* and doesn't equal $SUBSTRING(B)$ and doesn't equal $EXTEND(B)$.

4. **Computational problems** (8 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 three 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 for 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 ⟨D⟩ where D is a DFA :
    1. for i = 1, 2, 3, ...
    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 ⟨T⟩ where T is a Turing machine :
    1. for i = 1, 2, 3, ...
    2.     Run T for i steps on each input s_1, s_2, ..., s_i in turn
    3.     If T has accepted any of these, accept.
    4.     Otherwise, go to the next iteration of the loop"
```

5. **Computational problems** (8 points):

(a) (*Graded for completeness*) Prove that the language

$$\{\langle D\rangle \mid D \text{ is an NFA over } \{0,1\} \text{ and } D \text{ accepts at least 3 strings of length less than 5 }\}$$

is decidable.

(b) (*Graded for correctness*) Prove that the language

$$\{\langle R\rangle \mid R \text{ is a regular expression over } \{0,1\} \text{ and } L(R) \text{ has infinitely many strings starting with } 0\}$$

is decidable.