HW2CSE105F24: Homework assignment 2

CSE105F24

Due: October 15th at 5pm, via Gradescope

In this assignment,

You will practice designing multiple representations of regular languages and working with general constructions of automata to demonstrate the richness of the class of regular languages.

Resources: To review the topics for this assignment, see the class material from Week 2. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Section 1.1, 1.2, 1.3. Chapter 1 exercises 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.14, 1.15, 1.16, 1.17, 1.19, 1.20, 1.21, 1.22. Chapter 1 problem 1.51.

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,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.
- 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 "hw2CSE105F24".

Assigned questions

1. **Automata design** (12 points): As background to this question, recall that integers can be represented using base b expansions, for any convenient choice of base b. The precise definition is: for b an integer greater than 1 and n a positive integer, the **base b expansion of** n is defined to be

$$(a_{k-1} \cdots a_1 a_0)_b$$

where k is a positive integer, $a_0, a_1, \ldots, a_{k-1}$ are nonnegative integers less than b, $a_{k-1} \neq 0$, and

$$n = \sum_{i=0}^{k-1} a_i b^i$$

Notice: *The base b expansion of a positive integer n is a string over the alphabet $\{x \in \mathbb{Z} \mid 0 \leq x < b\}$ whose leftmost character is nonzero.*

An important property of base b expansions of integers is that, for each integer b greater than 1, each positive integer $n = (a_{k-1} \cdots a_1 a_0)_b$, and each nonnegative integer a less than b,

$$bn + a = (a_{k-1} \cdots a_1 a_0 a)_b$$

In other words, shifting the base b expansion to the left results in multiplying the integer value by the base. In this question we'll explore building deterministic finite automata that recognize languages that correspond to useful sets of integers.

(a) (*Graded for completeness*) [1] Design a DFA that recognizes the set of binary (base 2) expansions of positive integers that are powers of 2. A complete solution will include the state diagram of your DFA and a brief justification of your construction by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

Hints: (1) A power of 2 is an integer x that can be written as 2^y for some nonnegative integer y, (2) the DFA should accept the strings 100, 10 and 100000 and should reject the strings 010, 1101, and ε (can you see why?).

(b) (*Graded for completeness*) Consider arbitrary positive integer m. Design a DFA that recognizes the set of binary (base 2) expansions of positive integers that are multiples of m. A complete solution will include the formal definition of your DFA (paramterized by m) and a brief justification of your construction by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

Hints: (1) Consider having a state for each possible remainder upon division by m. (2) To determine transitions, notice that reading a new character will shift what we already read over by one slot.

(c) (*Graded for correctness*) [2] Choose a positive integer m_0 between 5 and 8 (inclusive) and draw the state diagram of a DFA recognizing the following language over $\{0, 1, 2, 3\}$

$$\{w \in \{0, 1, 2, 3\}^* \mid w \text{ is a base 4 expansion of a positive integer that is a multiple of } m_0\}$$

A complete solution will include the state diagram of your DFA and a brief justification of your construction by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

[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.

[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.

Bonus extension to think about (ungraded): Which other languages related to sets of integers can be proved to be regular using a similar strategy?

2. **Nondeterminism** (15 points): For this question, the alphabet is $\{a, b\}$.

(a) (*Graded for completeness*) Design a DFA that recognizes the language

$$\{w \in \{a, b\}^* \mid w \text{ contains at most one } a \textbf{ and } \text{at least two } b\text{s}\}$$

You can design this DFA directly or use the constructions from class (and the footnote to Theorem 1.25 in the book) to build this DFA from DFA for the simpler languages that are intersected to give this language.

A complete solution will include the state diagram of your DFA and a brief justification of your construction either by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language, or by justifying the design of the DFA for the simpler languages and then describing how the Theorem was used.

(b) (*Graded for correctness*) Design a NFA with at most 6 states that recognizes the language

$$\{w \in \{a, b\}^* \mid w \text{ contains at most one } a \textbf{ and } \text{at least two } b\text{s}\}$$

A complete solution will include the state diagram of your NFA and a brief justification of your construction by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language. Give one example string in the language and explain the computation of the NFA that witnesses that the machine accepts this string. Also, give one example string not in the language and explain why the NFA rejects this string.

(c) (*Graded for correctness*) Design a NFA with at most 6 states that recognizes the language

$$\{w \in \{a, b\}^* \mid w \text{ contains at most one } a \textbf{ or } \text{at least two } b\text{s}\}$$

A complete solution will include the state diagram of your NFA and a brief justification of your construction by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language. Give one example string in the language and explain the computation of the NFA that witnesses that the machine accepts this string. Also, give one example string not in the language and explain why the NFA rejects this string.

Bonus extension to think about (ungraded): Did you need all 6 states? Could you design DFA with 6 states that recognize each of these langauges?

3. **General constructions** (15 points): In this question, you'll practice working with formal general constructions for NFAs and translating between state diagrams and formal definitions.

(a) (*Graded for correctness*) Consider the following general construction: Let $N_1 = (Q, \Sigma, \delta_1, q_1, F_1)$ be a NFA and assume that $q_0 \notin Q$. Define the new NFA $N_2 = (Q \cup \{q_0\}, \Sigma, \delta_2, q_0, \{q_1\})$ where

$$\delta_2 : (Q \cup \{q_0\}) \times \Sigma_\varepsilon \to \mathcal{P}(Q \cup \{q_0\})$$

is defined by

$$\delta_2(q, a) = \begin{cases} \{q' \in Q \mid q \in \delta_1(q', a)\} & \text{if } q \in Q,\ a \in \Sigma_\varepsilon \\ F_1 & \text{if } q = q_0,\ a = \varepsilon \\ \emptyset & \text{if } q = q_0,\ a \in \Sigma \end{cases}$$

Illustrate this construction by defining a specific example NFA N_1 and applying the construction above to create the new NFA N_2. Your example NFA should

- Have exactly four states (all reachable from the start state),
- Accept at least one string and reject at least one string, and
- Not have any states labelled q_0.

Apply the construction above to create the new NFA. A complete submission will include the state diagram of your example NFA N_1 and the state diagram of the NFA N_2 resulting from this construction and a precise and clear description of $L(N_1)$ and $L(N_2)$, justified by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the language.

(b) In Week 2's review quiz, we saw the definition that a set X is said to be **closed under an operation** if, for any elements in X, applying to them gives an element in X. For example, the set of integers is closed under multiplication because if we take any two integers, their product is also an integer .

Recall the definitions we have: For each language L over the alphabet $\Sigma_1 = \{0, 1\}$, we have the associated set of strings

$$EXTEND(L) = \{w \in \Sigma_1^* \mid w = uv \text{ for some strings } u \in L \text{ and } v \in \Sigma_1^*\}$$

We will prove that the collection of languages over $\{0, 1\}$ that are each recognizable by some NFA is closed under the $EXTEND$ operation.

i. (*Graded for completeness*) As a helpful tool in our construction[3], prove that every NFA can be converted to an equivalent one that has a single accept state. Note: this is exercise 1.11 in the textbook.

ii. (*Graded for correctness*) Prove that the collection of languages over $\{0, 1\}$ that are each recognizable by some NFA is closed under the $EXTEND$ operation. You can assume that you are given a NFA with a single accept state $N = (Q, \{0, 1\}, \delta, q_0, \{q_{acc}\})$ and you need to define a new NFA, $N_{new} = (Q_{new}, \{0, 1\}, \delta_{new}, q_{new}, F_{new})$, so that $L(N_{new}) = EXTEND(L(N))$.
A complete solution will include precise definitions for $Q_{new}, \delta_{new}, q_{new}$, and F_{new}, as well as a a brief justification of your construction by explaining why these definitions work, referring specifically to the definition of $EXTEND$ and to acceptance of NFA.

[3] A result that is proved in order to work towards a larger theorem is called a Lemma.

4. **Multiple representations** (8 points): For any language $L \subseteq \Sigma^*$, recall that we define its *complement* as

$$\overline{L} := \Sigma^* - L = \{w \in \Sigma^* \mid w \notin L\}$$

That is, the complement of L contains all and only those strings which are not in L. Our notation for regular expressions does not include the complement symbol. However, it turns out that the complement of a language described by a regular expression is guaranteed to also be describable by a (different) regular expression.[4]

For example, over the alphabet $\Sigma = \{a, b\}$, the complement of the language described by the regular expression $\Sigma^* b$ is described by the regular expression $\varepsilon \cup \Sigma^* a$ because any string that does not end in b must either be the empty string or end in a.

For each of the regular expressions R over the alphabet $\Sigma = \{a, b\}$ below, write the regular expression for $\overline{L(R)}$. Your regular expressions may use the symbols $\varnothing$, ε, a, b, and the following operations to combine them: union, concatenation, and Kleene star.

Briefly justify why your solution for each part works by giving plain English descriptions of the language described by the regular expression and of its complement and connecting them to the regular expression via relevant definitions. An English description that is more detailed than simply negating the description in the original language will likely be helpful in the justification.

Alternatively, you can justify your solution by first designing a DFA that recognizes $L(R)$, using the construction from class and the book to modify this DFA to get a new DFA that recognizes $\overline{L(R)}$, and then applying the constructions from class and the book to convert this new DFA to a regular expression.

For each part of the question, clearly state which approach you're taking and include enough intermediate steps to illustrate your work.

(a) (*Graded for correctness*) $(a \cup b)^* a (a \cup b)^*$

(b) (*Graded for correctness*) $(a \cup b)(a \cup b)(a \cup b)$

[4] We'll see that this is connected to the result we proved in class that the complement of each language recognizable by a DFA is recognizable by a(nother) DFA.