HW1 : Regular Expressions and Finite Automata

CSE105W24

Due: April 11th at 5pm (no penalty late submission until 8am next morning), via Gradescope

In this assignment,

You will practice reading and applying the definitions of alphabets, strings, languages, Kleene star, and regular expressions. You will use regular expressions and relate them to languages and finite automata. You will use precise notation to formally define the state diagram of finite automata, and you will use clear English to describe computations of finite automata informally.

Resources: To review the topics for this assignment, see the class material from Week 1. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Section 0, 1.3, 1.1. Chapter 1 exercises 1.1, 1.2, 1.3, 1.18, 1.23.

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. You may only collaborate on HW with CSE 105 students in your group; if your group has questions about a HW problem, you may ask in drop-in help hours or post a private post (visible only to the Instructors) 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 "hw1CSE105Sp23".

Assigned questions

1. **Functions over sets of strings** (17 points):
For this question, fix the alphabets $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, 2\}$.

Whenever K is a set of strings over Γ and L is a set of strings over Σ, we can use the following rules to define associated sets of strings:

$$\begin{aligned} \textsc{Substring}(K) &:= \{w \in \Gamma^* \mid \text{there exist } a, b \in \Gamma^* \text{ such that } awb \in K\} \\ \textsc{Rep}(L) &:= \{w \in \Gamma^* \mid \text{between every pair of successive 2s in } w \text{ is a string in } L\} \\ &= \{w \in \Gamma^* \mid \text{for all } v \in \Sigma^* \text{ if } 2v2 \in \textsc{Substring}(\{w\}), \text{ then } v \in L\} \end{aligned}$$

Note: Formally, Substring and Rep are functions whose domains and codomains are specified as

$$\textsc{Substring} : \mathcal{P}(\Gamma^*) \to \mathcal{P}(\Gamma^*)$$

and

$$\textsc{Rep} : \mathcal{P}(\Sigma^*) \to \mathcal{P}(\Gamma^*)$$

In other words, SUBSTRING maps sets of strings with characters $\{0, 1, 2\}$ to associated sets of strings with characters $\{0, 1, 2\}$; and REP maps sets of strings with characters in $\{0, 1\}$ to associated sets of strings with characters in $\{0, 1, 2\}$.

(a) (*Graded for correctness*) [1] Consider $w = 0120$ (which is a string in Γ^*). List every element of the set SUBSTRING($\{w\}$). In other words, fill in the blank

$$\text{SUBSTRING}(\{w\}) = \{\_\_\_\_\_\}$$

Briefly justify your answer by referring back to the relevant definitions.

Not graded, but good to think about: Why do we need the curly braces—"{" and "}"—around w for the input to SUBSTRING*?*

(b) (*Graded for correctness*) Specify an example language A over Γ such that $A \neq \Gamma^*$ and yet SUBSTRING(A) $= \Gamma^*$, or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your example language A and a precise and clear description of the result of computing SUBSTRING(A) using relevant definitions to justify this description and to justify the set equality with Γ^*, or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

(c) (*Graded for completeness*) [2] Define the language B to be the language over Σ described by the regular expression

$$\Sigma^* 1 \Sigma^*$$

In plain English, we might explain that B is the set of all strings of 0s and 1s that contain a 1. Give a plain English explanation for the set of strings REP(B).

(d) (*Graded for correctness*) Prove/disprove: For every finite language L over Σ, REP(L) is also a finite set of strings. A complete answer will either give a general argument starting with an arbitrary finite language and proving that the result of applying REP is also finite, or will give a counterexample (which is a specific example of a finite language L for which applying REP gives an infinite language, with justification referring back to the relevant definitions).

Note: A finite language is a set of finitely many strings. This includes the possibility that L is the empty set!

(e) (*Graded for completeness*) Write a template for a regular expression that describes REP(L) when L is described by a regular expression R. You may use union, concatenation, Kleene star, and Σ, Γ, and R. (We're using the shorthand for regular expressions describing alphabets from page 64.)

[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.

[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.

2. **Deciphering regular expressions** (22 points):
For this question, let's fix the regular expression over the alphabet $\{0, 1\}$

$$R = 0^*(1 \cup 10)^*$$

For each choice of strings of length 3, $a, b, c \in \{0, 1\}^3$ we can define the regular expression:

$$X_{a,b,c} = 0(a \cup b \cup c)^*$$

(a) (*Graded for completeness*) Give a plain English explanation for the language described by the regular expression R. This continues a theme from Problem 1—before trying to prove formal statements about a specific regular expression, it's often good to try to translate it into a form that is more easy to reason about. Typically speaking, the shorter and more concise your plain English description is, the more useful it will be in reasoning about the language.

(b) (*Graded for correctness*) Suppose $a = 000$, $b = 001$, $c = 011$ so

$$X_{a,b,c} = 0(000 \cup 001 \cup 011)^*$$

Show that $L(R) \not\subseteq L(X_{a,b,c})$ by giving some string in $L(R)$ which is not in $L(X_{a,b,c})$, and justifying this choice referring back to relevant definitions.

(c) (*Graded for correctness*) More generally, prove that

$$L(R) \not\subseteq L(X_{a,b,c})$$

for *all* possible strings $a, b, c \in \{0, 1\}^3$. Hint: What are the possible lengths of strings in $L(R)$ (and why does this help)?

(d) (*Graded for correctness*) Give a specific example of three distinct strings $a, b, c \in \{0, 1, 2\}^3$ such that

$$L(X_{a,b,c}) \subseteq L(R)$$

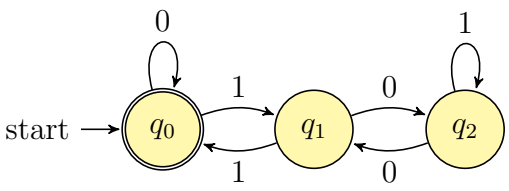
Briefly justify your answer by explaining how an arbitrary element of $L(X_{a,b,c})$ is guaranteed to be an element of $L(R)$.

(e) (*Graded for correctness*) Give a specific example of three distinct strings $a, b, c \in \{0, 1, 2\}^3$ such that

$$L(X_{a,b,c}) \not\subseteq L(R)$$

Briefly justify your answer by giving a counterexample string that is in $L(X_{a,b,c})$ and is not in $L(R)$ (and explaining why using relevant definitions).

3. **The right transition function can make or break a DFA** (6 points):
Consider the finite automaton $(Q, \Sigma, \delta, q_0, F)$ depicted below



where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, and $F = \{q_0\}$.

(a) (*Graded for completeness*) Find and fix the mistake in the following symbolic description of the transition function $\delta\colon Q \times \Sigma \to Q$: for each $j \in \{0, 1\}$

$$\delta(q_0, j) = q_j \qquad \delta(q_1, j) = q_{1-j} \qquad \delta(q_2, j) = q_{1+j}$$

(b) (*Graded for correctness*) Keeping the same set of states $Q = \{q_0, q_1, q_2\}$, alphabet $\Sigma = \{0, 1\}$, starting state q_0, and set of accepting states $F = \{q_0\}$, change the transition function δ so that the resulting finite automaton recognizes the language described by the regular expression

$$0^* \cup \Sigma^* 1000^*$$

Briefly justify why the resulting finite automaton works by describing the role of each state with your new transition function and relating it to a plain English description of the language described by the regular expression.

Note: with regular expressions $*$ binds more tightly than concatenation so $1000^* = (100)(0^*)$.

(Challenge question, not graded) There is a beautiful plain English description of the language recognized by the finite automaton with the state diagram depicted at the start of Problem 3. What is it?

4. **Being precise with terminology** (5 points):
For each of the following statements, determine if it is true, false, or if the question doesn't even make sense (because the statement isn't well formed or doesn't use terms in ways consistent with definitions from class).

(a) (*Graded for completeness*) The empty string is in every language.

(b) (*Graded for completeness*) Σ^* is a language.

(c) (*Graded for completeness*) Every language is a regular expression.

(d) (*Graded for completeness*) Alphabets are infinite.

(e) (*Graded for completeness*) There is a (finite) number $k \in \mathcal{N}$ such that every DFA has fewer than k states.

Project - CSE 105 Spring 2022 Due 6/1/23 at 5pm (no penalty late submission until 8am next day)

The project component is designed for you to go deeper and extend your work on assignments and to explore an application of your choosing. The project is an individual assignment and has two tasks:

- Task 1: A meaningful language (written) and
- Task 2: A helpful function (some programming, presented as a screencast video).

What resources can you use? This project must be completed individually, without any help from other people, including the course staff (other than logistics support if you get stuck with screencast).

You can use any of this quarter's CSE 105 offering (notes, readings, class videos, homework feedback). These resources should be more than enough.

If you are struggling to get started and want to look elsewhere online, you must acknowledge this by listing and citing any resources you consult (even if you do not explicitly quote them), including any large-language model style resources. Link directly to them and include the name of the author / video creator, any search strings or prompts you used, and the reason you consulted this reference.

The work you submit for the project needs to be your own. Again, you shouldn't need to look anywhere other than this quarter's material and doing so may result in definitions or notations that conflict with our norms in this class so think carefully before you go down this path.

Submitting the project You will submit a PDF for Task 1 and a video for Task 2 to an online assignment on Gradescope.

Task 1: A meaningful language

Automaton models are useful for concisely representing patterns. In this question, you'll choose a pattern in an application you care about, define it precisely, and then build a DFA, NFA, or PDA that recognizes it.

First, pick **one** application for your example. Here are some ideas to get you started - but you can choose to go in a different direction all together.

- Data validation for input in text files (e.g. emails with specific domains, dates in specific formats, PIDs in a class list, etc.)
- Finding ASCII codes for punctuation in a binary file.
- The CDC recommended procedure for hand washing (Refer to the guidelines from the CDC here https://www.cdc.gov/handwashing/index.html in your explanation. You might find the first example in chapter 1 about automatic door controllers helpful when starting your design.)
- See more ideas here:

  `https://theory-cs.github.io/files/practical-applications-of-theory-of-computation.pdf`

Then:

1. In a paragraph or so, give the context for your chosen application and why you chose it.
2. Specify the alphabet for your example.
3. Write a precise (mathematical and/or English) description of a set of strings over this alphabet that is important, and include a sentence or so justifying why this set is important.
4. Give one example of a string in this set and a string not in this set, and explain why you chose these example strings.
5. Classify your language as regular or context-free, and prove this classification by giving an appropriate machine that recognizes your language. Justify your construction.

Grading criteria and checklists Solution is typed out in detail step-by-step, with clear and correct logic and justification.

Each of the five items are included, with precise language and notation for all terms and complete, correct, and clear justification.

Task 2: A helpful function

To relate the difficulty level of one language to another we use mapping reduction, which relies on the notion of computable function. In this part of the project, you will define, program, and trace a specific computable function from $\{a, b\}^*$ to $\{a, b\}^*$.

First, choose a function you will be working with. You can pick any function you like so long as:

- Its domain is $\{a, b\}^*$ and its codomain is $\{a, b\}^*$
- There is at least one domain element that is mapped to a string that is **longer** than it and there is at least one domain element that is mapped to a string that is **shorter** than it.

Then:

1. Give a high-level description of a Turing machine that computes this function.
2. Draw the state diagram of this Turing machine.
3. Write a program in Java, Python, JavaScript, or C++ (or another programming language of your choosing) that simulates the computations of this Turing machine. Your program should display a snapshot of each step of the computation, including the state of the machine, the current (non-blank) contents of the tape, and the location of the read/write head of the Turing machine. If you would like, you may use aids such as co-pilot or ChatGPT to help you write this program. However, you should test the code that is produced and be able to explain what it is doing. As a header in your code file, include a comment block describing any resources that were used to help generate your code.
4. Run your program on a string over $\{a, b\}$ which the function maps to a string that is longer than it, and
5. Run your program on a string over $\{a, b\}$ which the function maps to a string that is shorter than it.

Presenting your reasoning and demonstrating it via screenshare are important skills that also show us a lot of your learning. Getting practice with this style of presentation is a good thing for you to learn in general and a rich way for us to assess your skills. To demonstrate your work, you will create a 3-5 minute screencast video with the following components:

- Start with your face and your student ID for a few seconds at the beginning, and introduce yourself audibly while on screen. You don't have to be on camera for the rest of the video, though it's fine if you are. We are looking for a brief confirmation that it's you creating the video and doing the work you submitted.

- Present the function you will be working with. Your video should include a clear and precise definition of the function (before you introduce any Turing machines).
- Show on the screen and explain the high-level description of your Turing machine witnessing that your function is computable.
- Show on the screen and explain the state diagram of your Turing machine.
- Show on the screen and present your code, including the software design choices you made (e.g. which data structures are you using, etc.) and any resources you used.
- Demonstrate running your code on the two example input strings specified above. Explain why the output of your program is what you would expect.

Grading criteria and checklists Logistics: video loads correctly, is between 3 and 5 minutes, shows the student's face and ID, and they introduce themself audibly while on screen.

The video clearly states which function was chosen for study, the function which is well-defined and computable, the video presents the two different descriptions of the Turing machine clearly, and the Turing machine correctly computes the function.

The video clearly describes which programming language was chosen for the implementation and gives the reasons why.

The video discusses the connections between the state diagram of the Turing machine and its implementation in the code.

The video clearly demonstrates all test cases, including both expected and actual output. The video should include screencasts of running the code live to demonstrate these test cases.

Your video: You may produce screencasts with any software you choose. One option is to record yourself with Zoom; a tutorial on how to use Zoom to record a screencast (courtesy of Prof. Joe Politz) is here:

`https://drive.google.com/open?id=1KROMAQuTCk4OzwrEFotlYSJJQdcG_GUU`.

The video that was produced from that recording session in Zoom is here:

`https://drive.google.com/open?id=1MxJN6CQcXqIbOekDYMxjh7mTt1TyRVMl`

Please send an email to the instructors (minnes@ucsd.edu and dgrier@ucsd.edu) if you have concerns about the video / screencast components of this project or cannot complete projects in this style for some reason.