HW1CSE105F24: Homework assignment 1

CSE105F24

Due: October 8th at 5pm, 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 Weeks 0 and 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. 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 "hw1CSE105F24".

Assigned questions

1. **Finding examples and edge cases** (12 points):

With $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $\Gamma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

(a) (*Graded for completeness*) [1] Give an example of a string over Σ that is meaningful to you in some way and whose length is between 5 and 20, and explain why this string is meaningful to you.

[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.

(b) (*Graded for completeness*) Calculate the number of distinct strings of length 3 over Σ and then explain your calculation.

(c) (*Graded for completeness*) With the ordering $0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < A < B < C < D < E < F$, list the first 50 strings over Γ in string order. Explain how you constructed this list. *Note: you can write a program to generate this list if you'd like, and you may use any external tools to help you write this program. If you do use a program to generate the list, include it (and documentation for how it works) as part of your submission.*

(d) (*Graded for correctness*) [2] Give an example of a finite set that is a language over Σ and over Γ, or explain why there is no such set. A complete and correct answer will use clear and precise notation (consistent with the textbook and class notes) and will include a description of why the given example is a language over Σ and over Γ and is finite, or an explanation why there is no such example.

(e) (*Graded for correctness*) Give an example of an infinite set that is a language over Σ and not over Γ, or explain why there is no such set. A complete and correct answer will use clear and precise notation (consistent with the textbook and class notes) and will include a description of why the given example is a language over Σ and not over Γ and is infinite, or an explanation why there is no such example.

2. **Regular expressions** (10 points):

(a) (*Graded for completeness*) Give three regular expressions that all describe the set of all strings over $\{a, b\}$ that have odd length. Ungraded bonus challenge: Make the expressions as different as possible!

(b) (*Graded for completeness*) A friend tells you that each regular expression that has a Kleene star (*) describes an infinite language. Are they right? Either help them justify their claim or give a counterexample to disprove it and explain your counterexample.

3. **Functions over languages** (15 points):

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

$$SUBSTRING(L) = \{w \in \Sigma_1^* \mid \text{there exist } x, y \in \Sigma_1^* \text{ such that } xwy \in L\}$$

and

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

(a) (*Graded for completeness*) Specify an example language A over Σ_1 such that $SUBSTRING(A) = EXTEND(A)$, or explain why there is no such example. A complete solution will include

[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.

either (1) a precise and clear description of your example language A and a precise and clear description of the result of computing $SUBSTRING(A)$, $EXTEND(A)$ (using the given definitions) to justify this description and to justify the set equality, or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

(b) (*Graded for correctness*) Specify an example language B over Σ_1 such that

$$SUBSTRING(B) = \{\varepsilon\}$$

and

$$EXTEND(B) = \Sigma_1^*$$

or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your example language B and a precise and clear description of the result of computing $SUBSTRING(B)$, $EXTEND(B)$ (using the given definitions) to justify this description and to justify the set equality with $\{\varepsilon\}$ and Σ_1^* (respectively), or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

(c) (*Graded for correctness*) Specify an example **infinite** language C over Σ_1 such that

$$SUBSTRING(C) \neq \Sigma_1^*$$

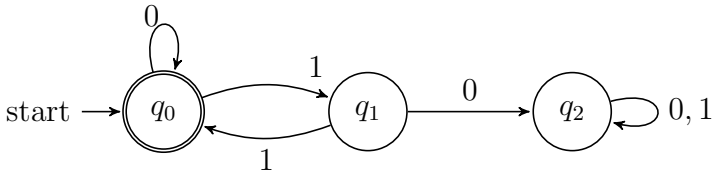
and

$$EXTEND(C) \neq \Sigma_1^*$$

, or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your example language C and a precise and clear description of the result of computing $SUBSTRING(B)$, $EXTEND(B)$ (using the given definitions) to justify this description and to justify the set nonequality claims, or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

4. **Finite automata** (13 points):

Consider the finite automaton $(Q, \Sigma, \delta, q_0, F)$ whose state diagram is depicted below



where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, and $F = \{q_0\}$, and $\delta : Q \times \Sigma \to Q$ is specified by the look-up table

	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_2	q_2

(a) (*Graded for completeness*) A friend tries to summarize the transition function with the formula

$$\delta(q_i, x) = \begin{cases} q_0 & \text{when } i = 0 \text{ and } x = 0 \\ q_2 & \text{when } x < i \\ q_j & \text{when } j = (i+1) \mod 2 \text{ and } x = 1 \end{cases}$$

for $x \in \{0, 1\}$ and $i \in \{0, 1, 2\}$. Are they right? Either help them justify their claim or give a counterexample to disprove it and then fix their formula.

(b) (*Graded for correctness*) Give a regular expression R so that $L(R)$ is the language recognized by this finite automaton. Justify your answer by referring to the definition of the semantics of regular expressions and computations of finite automata. Include an explanation for why each string in $L(R)$ is accepted by the finite automaton *and* for why each string not in $L(R)$ is rejected by the finite automaton.

(c) (*Graded for correctness*) Keeping the same set of states $Q = \{q_0, q_1, q_2\}$, alphabet $\Sigma = \{0, 1\}$, same start state q_0, and same transition function δ, choose a new set of accepting states F_{new} so that the new finite automaton that results accepts at least one string that the original one rejected **and** rejects at least one string that the original one accepted, or explain why there is no such choice of F_{new}. A complete solution will include either (1) a precise and clear description of your choice of F_{new} and a precise and clear the two example strings using relevant definitions to justify them, or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

HW2CSE105F24: Homework assignment 2 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*) [3] 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*) [4] 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.

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

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

[3] 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.

[4] 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.

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[5], 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.

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.[6]

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

[6] 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.

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*) $(a \cup b)(a \cup b)(a \cup b)$

HW3CSE105F24: Homework assignment 3 Due: October 22nd at 5pm, via Gradescope

In this assignment,

You will demonstrate the richness of the class of regular languages, as well as its boundaries.

Resources: To review the topics for this assignment, see the class material from Week 3. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Chapter 1. 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

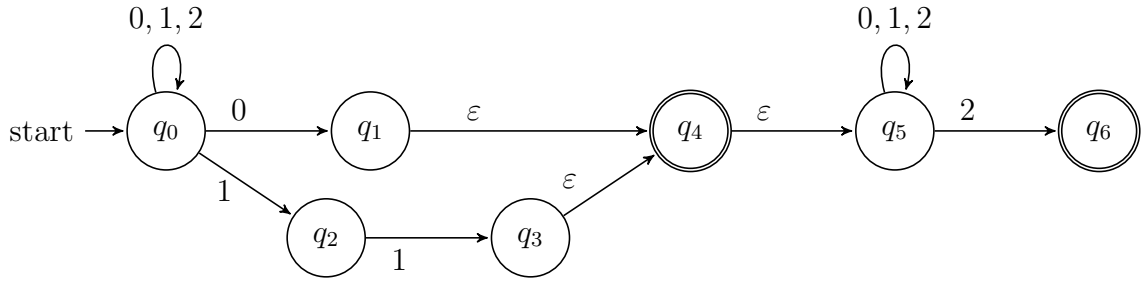
getting to know and learn from your classmates.

- 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 "hw3CSE105F24".

Assigned questions

1. **Using general constructions** (16 points): Consider the NFA N over $\{0, 1, 2\}$ with state diagram



(a) (*Graded for completeness*) [7] Give two examples of strings of length greater than 2 that are accepted by N and two examples of strings of length greater than 2 that are rejected by N. For each example string, list at least one of the computations of N on this string and label whether this computation witnesses that the string is accepted by N.

(b) (*Graded for correctness*) [8] Use the "macro-state" construction from Theorem 1.39 and class to create the DFA M recognizing the same language as N. You only need to include states

[7] 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.

[8] 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.

that are reachable from the start state. For full credit, submit (1) a state diagram that is deterministic (there should be arrows labelled 0, 1, and 2 coming out of each state) and where each state is labelled by a subset of the states in N; and (2) for one of your example strings that is accepted by N, give the computation of M on this string as a seqeuence of states visited; and (3) for one of your example strings that is rejected by N, give the computation of M on this string as a seqeuence of states visited.

(c) (*Graded for completeness*) Give a mathematical description either using set builder notation or a regular expression for $L(N)$ and for $L(M)$.

2. **Multiple representations** (12 points):

(a) Consider the language $A_1 = \{uw \mid u \text{ and } w \text{ are strings over } \{0,1\} \text{ and have the same length}\}$ and the following argument.

> "Proof" that A_1 is not regular using the Pumping Lemma: Let p be an arbitrary positive integer. We will show that p is not a pumping length for A_1.
> Choose s to be the string 1^p0^p, which is in A_1 because we can choose $u = 1^p$ and $w = 0^p$ which each have length p. Since s is in A_1 and has length greater than or equal to p, if p were to be a pumping length for A_1, s ought to be pump'able. That is, there should be a way of dividing s into parts x, y, z where $s = xyz$, $|y| > 0$, $|xy| \leq p$, and for each $i \geq 0$, $xy^iz \in A_1$. Suppose x, y, z are such that $s = xyz$, $|y| > 0$ and $|xy| \leq p$. Since the first p letters of s are all 1 and $|xy| \leq p$, we know that x and y are made up of all 1s. If we let $i = 2$, we get a string xy^iz that is not in A_1 because repeating y twice adds 1s to u but not to w, and strings in A_1 are required to have u and w be the same length. Thus, s is not pumpable (even though it should have been if p were to be a pumping length) and so p is not a pumping length for A_1. Since p was arbitrary, we have demonstrated that A_1 has no pumping length. By the Pumping Lemma, this implies that A_1 is nonregular.

i. (*Graded for completeness*) Find the (first and/or most significant) logical error in the "proof" above and describe why it's wrong.

ii. (*Graded for completeness*) Prove that the set A_1 is actually regular (by finding a regular expression that describes it or a DFA/NFA that recognizes it, and justifying why) **or** fix the proof so that it is logically sound.

(b) Consider the language $A_2 = \{u1w \mid u \text{ and } w \text{ are strings over } \{0,1\} \text{ and have the same length}\}$ and the following argument.

> "Proof" that A_2 is not regular using the Pumping Lemma: Let p be an arbitrary positive integer. We will show that p is not a pumping length for A_2.
> Choose s to be the string $1^{p+1}0^p$, which is in A_2 because we can choose $u = 1^p$ and $w = 0^p$ which each have length p. Since s is in A_2 and has length greater than or equal to p, if p were to be a pumping length for A_2, s ought to be pump'able. That is, there should be a way of dividing s into parts x, y, z where $s = xyz$, $|y| > 0$, $|xy| \leq p$, and for each $i \geq 0$, $xy^iz \in A_2$. When $x = \varepsilon$ and $y = 1^{p+1}$ and $z = 0^p$, we have satisfied that $s = xyz$, $|y| > 0$ (because p is positive) and $|xy| \leq p$. If we let

$i = 0$, we get the string $xy^iz = 0^p$ that is not in A_2 because its middle symbol is a 0, not a 1. Thus, s is not pumpable (even though it should have been if p were to be a pumping length) and so p is not a pumping length for A_2. Since p was arbitrary, we have demonstrated that A_2 has no pumping length. By the Pumping Lemma, this implies that A_2 is nonregular.

i. (*Graded for completeness*) Find the (first and/or most significant) logical error in the "proof" above and describe why it's wrong.

ii. (*Graded for completeness*) Prove that the set A_2 is actually regular (by finding a regular expression that describes it or a DFA/NFA that recognizes it, and justifying why) **or** fix the proof so that it is logically sound.

3. **Pumping** (10 points):

(a) (*Graded for correctness*) Give an example of a language over the alphabet $\{a, b\}$ that has cardinality 5 and for which 4 is a pumping length and 3 is not a pumping length. Is this language regular? A complete solution will give (1) a clear and precise description of the language, (2) a justification for why 4 is a pumping length, (3) a justification for why 3 is not a pumping length, (4) a correct and justified answer to whether the language is regular.

(b) (*Graded for completeness*) In class and in the reading so far, we've seen the following examples of nonregular sets:

$\{0^n1^n \mid n \geq 0\}$	$\{0^n1^m \mid 0 \leq m \leq n\}$	$\{0^n1^m0^n \mid n, m \geq 0\}$
$\{0^n1^n \mid n \geq 2\}$	$\{0^i1^{2i} \mid 0 \leq i\}$	$\{w \in \{0,1\}^* \mid w = w^R\}$
$\{0^n1^m \mid 0 \leq n \leq m\}$	$\{0^i1^{i+1} \mid 0 \leq i\}$	$\{ww^R \mid w \in \{0,1\}^*\}$

Modify one of these sets in some way and use the Pumping Lemma to prove that the resulting set is still nonregular.

4. **Regular and nonregular languages** (12 points): 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 .

Prove or disprove each closure claim statement below about the class of regular languages and the class of nonregular languages. Your arguments may refer to theorems proved in the textbook and class, and if they do, should include specific page numbers and references (i.e. write out the claim that was proved in the book and/or class).

Recall the definitions we have:

For language L over the alphabet $\Sigma_1 = \{0, 1\}$, we have the associated sets of strings

$$SUBSTRING(L) = \{w \in \Sigma_1^* \mid \text{there exist } a, b \in \Sigma_1^* \text{ such that } awb \in L\}$$

and

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

(a) (*Graded for completeness*) The set of regular languages over $\{0, 1\}$ is closed under the $SUBSTRING$ operation.

(b) (*Graded for completeness*) The set of nonregular languages over $\{0, 1\}$ is closed under the $SUBSTRING$ operation.

(c) (*Graded for correctness*) The set of regular languages over $\{0, 1\}$ is closed under the $EXTEND$ operation.

(d) (*Graded for correctness*) The set of nonregular languages over $\{0, 1\}$ is closed under the $EXTEND$ operation.

HW4CSE105F24: Homework assignment 4 Due: November 12, 2024 at 5pm, via Gradescope

In this assignment,

You will work with context-free languages and their representations. You will also 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.

Resources: To review the topics for this assignment, see the class material from Weeks 4, 5, and 6. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Chapters 2 and 3. Chapter 2 exercises 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.9, 2.10, 2.11, 2.12, 2.13, 2.16, 2.17. Chapter 3 exercises 3.1, 3.2, 3.5, 3.8.

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 "hw4CSE105F24".

Assigned questions

1. **Push-down automata (PDA) and context-free grammars (CFG)** (8 points): On page 14 of the week 3 notes, we have the following list of languages over the alphabet $\{a, b\}$

$$\{a^n b^n \mid 0 \leq n \leq 5\} \quad \{b^n a^n \mid n \geq 2\} \quad \{a^m b^n \mid 0 \leq m \leq n\}$$
$$\{a^m b^n \mid m \geq n+3, n \geq 0\} \quad \{b^m a^n \mid m \geq 1, n \geq 3\}$$
$$\{w \in \{a,b\}^* \mid w = w^{\mathcal{R}}\} \qquad \{ww^{\mathcal{R}} \mid w \in \{a,b\}^*\}$$

(a) (*Graded for completeness*) [9] Pick one of the regular languages and design a regular expression that describes it. Briefly justify your regular expression by connecting the subexpressions of it to the intended language and referencing relevant definitions.

(b) (*Graded for completeness*) Pick another one of the regular languages and design a deterministic finite automaton (DFA) that recognizes it. Draw the state diagram of your DFA. Briefly justify your design 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.

(c) (*Graded for completeness*) Pick one of the nonregular languages and design a PDA that recognizes it. Draw the state diagram of your PDA. Briefly justify your design 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.

[9] 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.

(d) (*Graded for completeness*) Pick one of the nonregular languages and write a CFG that generates it. Briefly justify your design by demonstrating how derivations in the grammar relate to the intended language.

2. **General constructions for context-free languages** (21 points):

In class in weeks 4 and 5, we described several general constructions with PDAs and CFGs, leaving their details to homework. In this question, we'll fill in these details. The first constructions help us prove that the class of regular languages is a subset of the class of context-free languages. The other construction allows us to make simplifying assumptions about PDAs recognizing languages.

(a) (*Graded for correctness*) [10] When we first introduced PDAs we observed that any NFA can be transformed to a PDA by not using the stack of the PDA at all. Suppose a friend gives you the following construction to formalize this transformation:

> Given a NFA $N = (Q, \Sigma, \delta_N, q_0, F)$ we define a PDA M with $L(M) = L(N)$ by letting $M = (Q, \Sigma, \Sigma, \delta, q_0, F)$ where $\delta(\ (q, a, b)\ ) = \delta_N(\ (q, a)\ )$ for each $q \in Q$, $a \in \Sigma_\varepsilon$ and $b \in \Sigma_\varepsilon$.

For each of the six defining parameters for the PDA, explain whether it's defined correctly or not. If it is not defined correctly, explain why not and give a new definition for this parameter that corrects the mistake.

(b) (*Graded for correctness*) In the book on page 107, the top paragraph describes a procedure for converting DFAs to CFGs:

> You can convert any DFA into an equivalent CFG as follows. Make a variable R_i for each state q_i of the DFA. Add the rule $R_i \to aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is a transition in the DFA. Add the rule $R_i \to \varepsilon$ if q_i is an accept state of the DFA. Make R_0 the start variable ofthe grammar, where q_0 is the start state of the machine. Verify on your own that the resulting CFG generates the same language that the DFA recognizes.

Use this construction to get a context-free grammar generating the language

$$\{w \in \{0,1\}^* \mid w \text{ does not end in } 101\}$$

by (1) designing a DFA that recognizes this language and then (2) applying the construction from the book to convert the DFA to an equivalent CFG. A complete and correct submission will include the state diagram of the DFA, a brief justification of why it recognizes the language, and then the complete and precise definition of the CFG that results from applying the construction from the book to this DFA. *Ungraded bonus: take a sample string in the language and see how the computation of the DFA on this string translates to a derivation in your grammar.*

[10] 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.

(c) Let $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F_1)$ be a PDA and let $q_{new}, r_{new}, s_{new}$ be three fresh state labels (i.e. $Q_1 \cap \{q_{new}, r_{new}, s_{new}\} = \emptyset$) and let $\#$ be a fresh stack symbol (i.e. $\# \notin \Gamma_1$). We define the PDA M_2 as

$$(Q_2, \Sigma, \Gamma_2, \delta_2, q_{new}, \{s_{new}\})$$

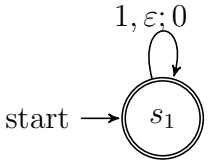
with $Q_2 = Q_1 \cup \{q_{new}, r_{new}, s_{new}\}$ and $\Gamma_2 = \Gamma_1 \cup \{\#\}$ and $\delta_2 : Q_2 \times \Sigma_\varepsilon \times \Gamma_{2\varepsilon} \to \mathcal{P}(Q_2 \times \Gamma_{2\varepsilon})$ given by

$$\delta_2(\ (q,a,b)\ ) = \begin{cases} \{(q_1, \#)\} & \text{if } q = q_{new}, a = \varepsilon, b = \varepsilon \\ \delta_1(\ (q,a,b)\ ) & \text{if } q \in Q_1 \setminus F_1, a \in \Sigma_\varepsilon, b \in \Gamma_{1\varepsilon} \\ \delta_1(\ (q,a,b)\ ) & \text{if } q \in F_1, a \in \Sigma, b \in \Gamma_{1\varepsilon} \\ \delta_1(\ (q,a,b)\ ) & \text{if } q \in F_1, a = \varepsilon, b \in \Gamma_1 \\ \delta_1(\ (q,a,b)\ ) \cup \{(r_{new}, \varepsilon)\} & \text{if } q \in F_1, a = \varepsilon, b = \varepsilon \\ \{(r_{new}, \varepsilon)\} & \text{if } q = r_{new}, a = \varepsilon, b \in \Gamma_1 \\ \{(s_{new}, \varepsilon)\} & \text{if } q = r_{new}, a = \varepsilon, b = \# \\ \emptyset & \text{otherwise} \end{cases}$$

for each $q \in Q_2$, $a \in \Sigma_\varepsilon$, and $b \in \Gamma_{2\varepsilon}$.

In this question, we'll apply this construction for a specific PDA and use this example to extrapolate the effect of this construction.

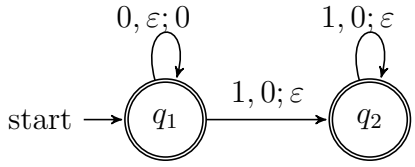
i. (*Graded for correctness*) Consider the PDA M_1 with input alphabet $\{0, 1\}$ and stack alphabet $\{0, 1\}$ whose state diagram is



Draw the state diagram for the PDA M_2 that results from applying the construction to M_1.

ii. (*Graded for completeness*) Compare $L(M_1)$ and $L(M_2)$. Are these sets equal? Does your answer depend on the specific choice of M_1? Why or why not?

iii. (*Graded for completeness*) Consider the PDA N with input alphabet $\{0, 1\}$ and stack alphabet $\{0, 1\}$ whose state diagram is



Remember that the definition of set-wise concatenation is: for languages L_1, L_2 over the alphabet Σ, we have the associated set of strings

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

In class, we discussed how extrapolating the construction that we used to prove that the class of regular languages is closed under set-wise concategation by drawing spontaneous transitions from the accepting states in the first machine to the start state of the second machine doesn't work. Use the example of M_1 and N_1 to prove this by showing that

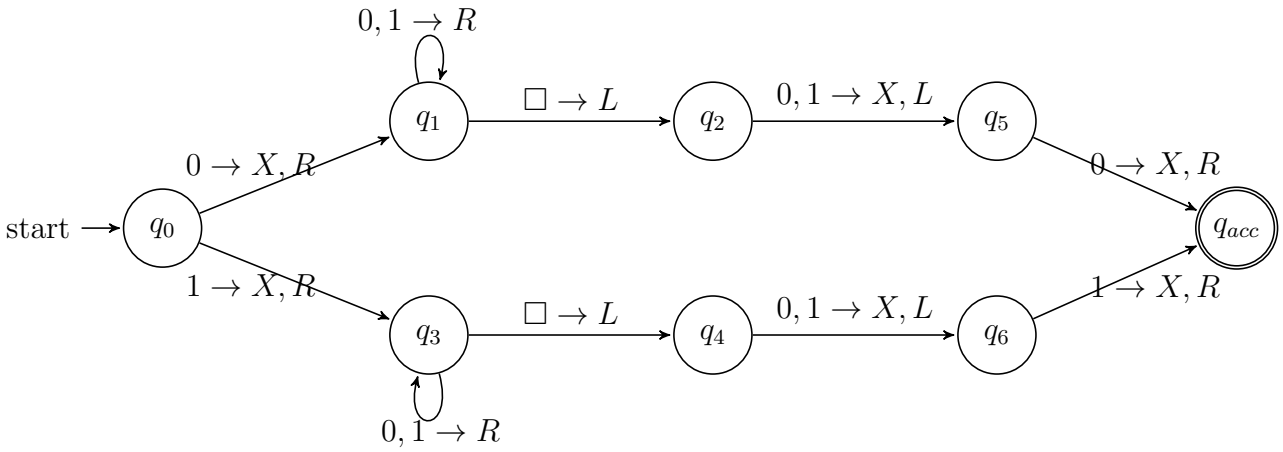
$$L(M_1) \circ L(N)$$

is **not** the language recognized by the machine results from taking the two machines M_1 and N, setting the start state of M_1 to be the start state of the new machine, setting the set of accepting states of N to be the set of accepting states of the new machine, and drawing spontaneous arrows from the accepting states of M_1 to the start state of N.

iv. (*Graded for completeness*) Describe the language recognized by the machine that results from taking the two machines M_2 and N, setting the start state of M_2 to be the start state of the new machine, setting the set of accepting states of N to be the set of accepting states of the new machine, and drawing spontaneous arrows from the accepting states of M_2 to the start state of N. Use this description to explain why we used the construction of M_2 from M_1 and how this construction could be used in a proof of the closure of the class of context-free languages under set-wise concatenation.

3. **Turing machines** (12 points):

Consider the Turing machine T over the input alphabet $\Sigma = \{0, 1\}$ with the state diagram below (the tape alphabet is $\Gamma = \{0, 1, X, \square\}$). Convention: we do not include the node for the reject state $qrej$ and any missing transitions in the state diagram have value $(qrej, \square, R)$



(a) (*Graded for correctness*) Specify an example string w_1 of length 4 over Σ that is **accepted** by this Turing machine, or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your example string and a precise and clear description of the accepting computation of the Turing machine on this string or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

To describe a computation of a Turing machine, include the contents of the tape, the state of the machine, and the location of the read/write head at each step in the computation.

Hint: In class we've drawn pictures to represent the configuration of the machine at each step in a computation. You may do so or you may choose to describe these configurations in words.

(b) (*Graded for correctness*) Specify an example string w_2 of length 3 over Σ that is **rejected** by this Turing machine or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your example string and a precise and clear description of the rejecting computation of the Turing machine on this string or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

(c) (*Graded for correctness*) Specify an example string w_3 of length 2 over Σ on which the computation of this Turing machine is **never halts** or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your example string and a precise and clear description of the looping (non-halting) computation of the Turing machine on this string or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

Note: when a Turing machine does not halt on a given input string, we say that it **loops** on that string.

4. **Implementation-level descriptions of deciders and recognizers** (9 points):

For this question, consider the alphabet $\Sigma = \{a, b, c\}$.

(a) (*Graded for correctness*) Give an example of an infinite language over Σ (that is not Σ^*) and give 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 completeness*) 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 completeness*) 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 $\{a, b, c\}$, or a complete and correct justification for why there is no such Turing machine.

HW5CSE105F24: Homework assignment 5 Due: November 19, 2024 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.

Resources: To review the topics for this assignment, see the class material from Weeks 6 and 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,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 "hw5CSE105F24".

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. Let's call that Turing machine M_0. The language

$$L = \{ww \mid w \in \{0,1\}^*\}$$

is also not context-free.

(a) (*Graded for correctness*) [11] Choose an example string of length 4 in L that is in **not** in $\{w\#w \mid w \in \{0,1\}^*\}$ and describe the computation of the Turing machine M_0 on your example string. Include the contents of the tape, the state of the machine, and the location of the read/write head at each step in the computation.

(b) (*Graded for completeness*) [12] Explain why the Turing machine from the textbook and class that recognized $\{w\#w \mid w \in \{0,1\}^*\}$ does not recognize $\{ww \mid w \in \{0,1\}^*\}$. Use your example to explain why M_0 doesn't recognize L.

[11] 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.

[12] 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.

(c) (*Graded for completeness*) Explain how you would change M_0 to get a new Turing machine that does recognize L. Describe this new Turing machine using both an implementation-level definition and a state diagram of the Turing machine. 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. **Closure** (18 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"

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

$$SUBSTRING(L) = \{w \in \Sigma_1^* \mid \text{there exist } x, y \in \Sigma_1^* \text{ such that } xwy \in L\}$$

and

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

(a) (*Graded for correctness*) Prove that this Turing machine construction **cannot** be used to prove that the class of decidable languages over $\{0,1\}$ is closed under the $EXTEND$ operation. A complete and correct answer will give a counterexample which is a set A over Σ_1 that is decidable, along with a definition of Turing machine M_A that decides A (with a justification why this Turing machine accepts all strings in A and rejects all strings not in A), and then either a description of the language of M_{new} that results when setting the Turing machine $M = M_A$ and an explanation why $L(M_{new}) \neq EXTEND(A)$ or a description why M_{new} is not a decider and therefore can't witness that $EXTEND(A)$ is decidable.

(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. A complete and correct answer will give a counterexample of a specific language B and Turing machine M_B recognizing it (with a justification why this Turing machine accepts all and only strings in B), and then a description of the language of M_{new} that results when setting the Turing machine $M = M_B$ and an explanation why $L(M_{new}) \neq SUBSTRING(B)$

(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*"

(d) (*Graded for completeness*) Prove that the class of recognizable languages over $\{0,1\}$ is closed under $EXTEND$.

3. **Computational problems** (12 points): Recall the definitions of some example computational problems from class

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

4. **Computational problems** (10 points): 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 justification for your answer.

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

$$\{\langle D \rangle \mid D \text{ is an NFA over } \{0,1\} \text{ and } L(D) = L(0^* \cup 1^*)\}$$

is decidable.

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

$$\{\langle R_1, R_2 \rangle \mid R_1, R_2 \text{ are regular expressions over } \{0,1\} \text{ and } L(R_1) \subseteq L(R_2)\}$$

is decidable.

ProjectCSE105F24: Project Due December 11, 2024 at 11am

The CSE 105 project is designed for you to go deeper and extend your work on assignments and to see how some of the abstract notions we discuss can be implemented in concrete ways. The project is an individual assignment and has two tasks:

Task 1: Illustrating the decidability of a computational problem, and

Task 2: Illustrating a mapping reduction

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). Tools for drawing state diagrams (like Flap.js and JFLAP and the PrairieLearn automata library) can be used to help draw the diagrams in the project too.

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 (ChatGPT, Bard, Co-Pilot, etc.). Link directly to them and include the name of the author / video creator, any and all 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 that conflict with our conventions in this class so think carefully before you go down this path.

If you get stuck on any part of the project, we encourage you to focus on communicating what you think the question might mean, including bringing an example from class or homework you think might be relevant, and include any submission any aspect where you're unsure. Clear communication about these theoretical ideas and their applications is one of the main goals of the project.

Submitting the project You will submit a PDF plus a video file for the first task and a PDF plus a video fiile for the second task. All file submissions will be in Gradescope.

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) if you have concerns about the video / screencast components of this project or cannot complete projects in this style for some reason.

Reference definitions for computational problems from Section 4.1:

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

Task 1: Illustrating the decidability of a computational problem Many computational problems are decidable, sometimes using beautiful algorithms. In this part of the project, you'll choose a **decidable** computational problem, and demonstrate the proof that it is decidable by building a program in a programming language of your choice (aka a high-level description of a Turing machine) that decides it. You will then demonstrate how your construction works for some test examples.

Specifically:

1. Choose a decidable computational problem from Section 4.1. *Note: if you'd like to consider a different computational problem instead, please check with Prof. Minnes first. You must do so no later than the start of Week 9.*

2. Write a program in Java, Python, JavaScript, C++ , or another programming language of your choosing that decides this computational problem. The function input must be a **string** and part of your work in this program is to design string representations for arbitrary instances of the model of computation the computational problem you picked is about (e.g. DFA, NFA, regular expressions, CFG, or NFA). The function output must be a **boolean** true (if the string is in the set representing the computational problem) or false (if the string is not in the set representing the computational problem).

    - You may use our class notes and the textbook for ideas on the algorithm that your program will implement.
    - 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. Your code needs to be well-organized and well-documented. As a header in your code file, include a comment block describing any resources that were used to help generate your code, including any and all prompts used in interactions with LLM coding tools.

3. To demonstrate your program, select one string that is in the set representing the computational problem, and one string that is not in the set representing the computational problem, explain why these strings are valid examples, and demonstrate running your program on each to get the appropriate output.

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 explaining your code design and demonstrating its functionality.

Checklist for submission For this task, you will submit a PDF plus a video file.

(PDF) Writeup includes a clear specification of computational problem being decided.

(PDF) Documentation for program deciding this computational problem: include a description of how input strings are parsed to represent instances of the computational problem.

(PDF) Clear specification of two example strings, explaining which is in the set (and why) and which is not in the set (and why not).

(PDF) Project submission includes a printout of code for program implementing algorithm to decide the computational problem, as well as screen shots demonstrating running your program on your example strings.

(PDF) Project writeup is typed or clearly hand drawn with precise language and notation for all terms.

(Video) Start with your face and your student ID visible 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.

(Video) Present the computational problem you will be working with, and example strings that you will be using, including explanations of why you chose this problem and these strings (and why one of the strings is in the set and why the other is not).

(Video) Show on the screen and explain the code for your program, including the software design choices you made (e.g. which data structures are you using, etc.) and any resources you used. The video should clearly describe which programming language was chosen for the implementation and gives the reasons why.

(Video) Demonstrate running your code on each of your example inputs. The video should include screencasts of running the code live. Explain why the output of your program is what you would expect, by connecting the output of the the definition of the computational problem and your chosen parsing of input strings.

(Video) Logistics: video needs to load correctly, be between 3 and 5 minutes, show your face and ID, and you introduce yourself audibly while on screen.

Note: Clarity and brevity are both important aspects of your video. In previous years, we've seen students speed up their videos to get below the 5 minute upper bound. This is ok so long as it doesn't compromise clarity. If the graders need to slow your video down to understand it, it may not earn full credit.

Task 2: Illustrating a mapping reduction We can use mapping reductions to prove that interesting computational problems are undecidable, building on the undecidability of other computational problems. In this part of the project, you'll choose a specific **mapping reduction** and implement a computable function that witnesses it using a programming language of your choice (aka a high-level description of a Turing machine that computes it). You will then demonstrate how your construction works for some test examples.

Specifically:

1. Choose a mapping reduction we discussed in class or in the homework or in review quizzes or in the textbook where both sets being compared are undecidable. *Note: if you'd like to consider a mapping reduction we have not discussed instead, please check with Prof. Minnes first. You must do so no later than the start of Week 9.*

2. Write a program in Java, Python, JavaScript, C++ , or another programming language of your choosing that implements a computable function witnessing this mapping reduction. The function input must be a **string** and the function output must be a **string**. Part of your work in this program is to design string representations for arbitrary instances of the model of computation the computational problems being compared in the mapping reduction. Your function will need to be able to process *any* string as input.

    - You may use our class notes and the textbook for ideas on the algorithm that your program will implement.
    - 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. Your code needs to be well-organized and well-documented. As a header in your code file, include a comment block describing any resources that were used to help generate your code, including any and all prompts used in interactions with LLM coding tools.

3. To demonstrate your program, you will need to run it for an example positive and negative instance. That is to say, if you are implementing a computable function witnessing $X \leq_m Y$, you will select one string that is in X and one string that is not in X, and you will demonstrate running your program on each of these strings and explain why the output of the function is good.

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 explaining your code design and demonstrating its functionality.

Checklist for submission For this task, you will submit a PDF plus a video file.

(PDF) Writeup includes a clear specification of mapping reduction being witnessed, and both sets in the reduction are undecidable.

(PDF) Documentation for program computing the function witnessing this mapping reduction: include a description of how input strings are parsed and how output strings correspond to input strings.

(PDF) Clear specification of two example strings, explaining which is is a positive instance (and why) and which is a negative instance (and why not).

(PDF) Project submission includes a printout of code for program computing the function witnessing the mapping reduction, as well as screen shots demonstrating running your program on your example strings.

(PDF) Project writeup is typed or clearly hand drawn with precise language and notation for all terms.

(Video) Start with your face and your student ID visible 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.

(Video) Present the mapping reduction you will be working with, and example strings that you will be using, including explanations of why you chose this reduction and these strings (and why one of the strings is a positive instance and the other is a negative instance).

(Video) Show on the screen and explain the code for your program, including the software design choices you made (e.g. which data structures are you using, etc.) and any resources you used. The video should clearly describe which programming language was chosen for the implementation and gives the reasons why.

(Video) Demonstrate running your code on each of your example inputs. The video should include screencasts of running the code live. Explain why the output of your program is what you would expect, by connecting the output of the program to the definition of the mapping reduction and your chosen parsing of input strings.

(Video) Logistics: video needs to load correctly, be between 3 and 5 minutes, show your face and ID, and you introduce yourself audibly while on screen.

Note: Clarity and brevity are both important aspects of your video. In previous years, we've seen students speed up their videos to get below the 5 minute upper bound. This is ok so long as it doesn't compromise clarity. If the graders need to slow your video down to understand it, it may not earn full credit.