HW1CSE105W25: Homework assignment 1

CSE105W25

Due: January 16th 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.

Resources: To review the topics for this assignment, see the class material from Weeks 0 and 1 and Review Quiz 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. Chapter 0 exercises 0.1, 0.2, 0.3, 0.5, 0.6, 0.9. Chapter 1 exercises 1.19, 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. On the "graded for correctness" questions, you may only collaborate with CSE 105 students in your group; if your group has questions about a problem, you may ask in drop-in help hours or post a private post (visible only to the Instructors) on Piazza. On the "graded for completeness" questions, you may collaborate with all other CSE 105 students this quarter, and you may make public posts about these questions on Piazza.

All submitted homework for this class must be typed. You can use a word processing editor if you like (Microsoft Word, Open Office, Notepad, Vim, Google Docs, etc.) but you might find it useful to take this opportunity to learn LaTeX. LaTeX is a markup language used widely in computer science and mathematics. The homework assignments are typed using LaTeX and you can use the source files as templates for typesetting your solutions. To generate state diagrams of machines, you can (1) use the LaTex tikzpicture environment (see templates in the class notes), or (2) use the software tools Flap.js or JFLAP described in the class syllabus (and include a screenshot in your PDF), or (3) you can carefully and clearly hand-draw the diagram and take a picture and include it in your PDF. We recommend that you submit early drafts to Gradescope so that in case of any technical difficulties, at least some of your work is present. You may update your submission as many times as you'd like up to the deadline.

Integrity reminders

- Problems should be solved together, not divided up between the partners. The homework is designed to give you practice with the main concepts and techniques of the course, while getting to know and learn from your classmates.
- On the "graded for correctness" questions, you may only collaborate with CSE 105 students in your group. You may ask questions about the homework in office hours (of the instructor, TAs, and/or tutors) and on Piazza (as private notes viewable only to the Instructors). You *cannot* use any online resources about the course content other than the class material from this quarter – this is primarily to ensure that we all use consistent notation and definitions (aligned with the textbook) and also to protect the learning experience you will have when the 'aha' moments of solving the problem authentically happen.
- Do not share written solutions or partial solutions for homework with other students in the class who are not in your group. Doing so would dilute their learning experience and detract from their success in the class.

You will submit this assignment via Gradescope (https://www.gradescope.com) in the assignment called "hw1CSE105W25".

Assigned questions

1. **Strings and languages: finding examples and edge cases** (12 points):

(a) (*Graded for completeness*) [1] Give five (different) example alphabets that are meaningful or useful to you in some way. Specify them formally, either with roster notation (which means listing all and only distict elements between { and } and separated by commas) or with another approach to precisely define all and only the elements of the alphabet.

(b) (*Graded for completeness*) Give an example of a finite set over an alphabet and an infinite set over an alphabet. You get to choose the alphabet, and you get to choose the sets.

[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.

The goal is to practice communicating your choices and definitions with clear and precise notation. One habit that will be useful (for this course, and beyond), is to think of your response for each question as a well-formed paragraph: include all the information that is relevant so that your solution is self-contained, and so that each sentence is grammatically constructed.

(c) (*Graded for correctness*) [2] Define an alphabet Σ_1 and an alphabet Σ_2 and a language L_1 over Σ_1 that is also a language over Σ_2 and a language L_2 over Σ_2 that is **not** a language over Σ_1. 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 L_1 is a language over both Σ_1 and Σ_2 and a description of why the given example L_2 is a language over Σ_2 and not over Σ_1.

2. **Regular expressions** (20 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.

(c) (*Graded for correctness*) For this question, the alphabet is $\{a, b, c\}$. A friend is trying to design a regular expression that describes the set of all strings over this alphabet that end in c. Classify each of the following attempts as

- Correct. Explain why.
- Error Type 1: Incorrect, because (even though each string that is in the language described by the regular expression ends in c) there is a string that ends in c that is not in the language described by the regular expression. Give this example string and explain why it proves we're in this case.
- Error Type 2: Incorrect, because (even though each string that ends in c is in the language described by the regular expression), there is a string in the language described by the regular expression that does not end in c. Give this example string and explain why it proves we're in this case.
- Error Type 3: Incorrect, because there are two counterexample strings, one which is a string that ends in c that is not in the language described by the regular expression and one which is in the language described by the regular expression but does not end in c.Give both example strings and describe why each has the given property.

[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.

Worked example for reference: Consider the regular expression $(a \cup b \cup c)^*$. This regular expression has **Error Type 2** because it describes the set of all strings over $\{a, b, c\}$, so even though each string that ends in c is in this language, there is an example, say ab that is a string in the language described by the regular expression (because we consider the string formed as a result of the Kleene star operation which has 2 slots and where the first slot matches the a in $a \cup b \cup c$ and the second slot matches b in $a \cup b \cup c$) but does not end in c (it ends in b).

i. The regular expression is
$$(a \cup b)^* \circ c$$

ii. The regular expression is
$$(a \circ b \circ c)^*$$

iii. The regular expression is
$$a^*c \ \cup \ b^*c \ \cup \ c^*c$$

3. **Functions over languages** (18 points):

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

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

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

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

Also, recall the set operations union and intersection: for any sets X and Y

$$X \cup Y = \{w \mid w \in X \text{ or } w \in Y\}$$

$$X \cap Y = \{w \mid w \in X \text{ and } w \in Y\}$$

(a) (*Graded for completeness*) Specify an example language A over $\{0, 1\}$ such that

$$SUBSTRING(A) = EXTEND(A)$$

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 $SUBSTRING(A)$, $EXTEND(A)$ (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 $\{0, 1\}$ such that

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

and

$$SUBSTRING(B) \cup EXTEND(B) = \{0,1\}^*$$

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 $\{0,1\}^*$ (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 $\{0,1\}$ such that

$$SUBSTRING(C) \neq \{0,1\}^*$$

and

$$SUBSTRING(C) = C^*$$

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 $SUBSTRING(C)$, C^* (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.

(d) (*Graded for correctness*) Specify an example **finite** language D over $\{0,1\}$ such that

$$EXTEND(D) \neq \{0,1\}^*$$

and

$$EXTEND(D) = D^*$$

or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your example language D and a precise and clear description of the result of computing $EXTEND(D)$, D^* (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.

HW2CSE105W25: Homework assignment 2 Due: January 30th 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 and Week 3. 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, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19, 1.20, 1.21, 1.22, 1.23. Chapter 1 problem 1.31, 1.36, 1.37.

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

All submitted homework for this class must be typed. You can use a word processing editor if you like (Microsoft Word, Open Office, Notepad, Vim, Google Docs, etc.) but you might find it useful to take this opportunity to learn LaTeX. LaTeX is a markup language used widely in computer science and mathematics. The homework assignments are typed using LaTeX and you can use the source files as templates for typesetting your solutions. To generate state diagrams of machines, you can (1) use the LaTex tikzpicture environment (see templates in the class notes), or (2) use the software tools Flap.js or JFLAP described in the class syllabus (and include a screenshot in your PDF), or (3) you can carefully and clearly hand-draw the diagram and take a picture and include it in your PDF. We recommend that you submit early drafts to Gradescope so that in case of any technical difficulties, at least some of your work is present. You may update your submission as many times as you'd like up to the deadline.

Integrity reminders

- Problems should be solved together, not divided up between the partners. The homework is designed to give you practice with the main concepts and techniques of the course, while getting to know and learn from your classmates.

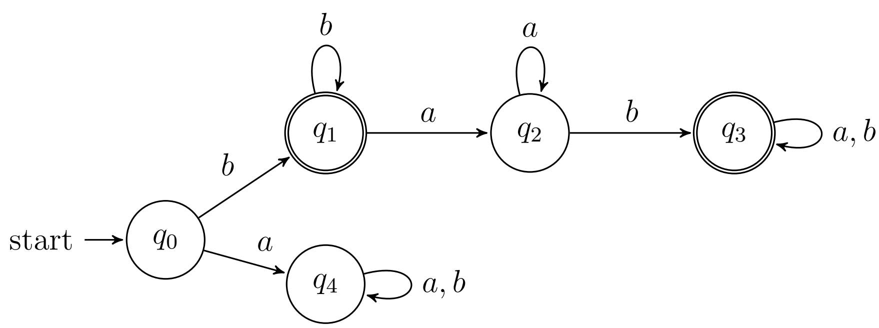
- On the "graded for correctness" questions, you may only collaborate with CSE 105 students in your group. You may ask questions about the homework in office hours (of the instructor, TAs, and/or tutors) and on Piazza (as private notes viewable only to the Instructors). You *cannot* use any online resources about the course content other than the class material from this quarter – this is primarily to ensure that we all use consistent notation and definitions (aligned with the textbook) and also to protect the learning experience you will have when the 'aha' moments of solving the problem authentically happen.

- Do not share written solutions or partial solutions for homework with other students in the class who are not in your group. Doing so would dilute their learning experience and detract from their success in the class.

You will submit this assignment via Gradescope (https://www.gradescope.com) in the assignment called "hw2CSE105W25".

Assigned questions

1. **Finite automata** (10 points): Consider the finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ whose state diagram is depicted below



(a) (*Graded for completeness*) [3] Write the formal definition of this automaton. In other words, give the five defining parameters Q, Σ, δ, q_0, F so that they are consistent with the state diagram of M.

(b) (*Graded for correctness*) [4] Give a regular expression R so that $L(R) = L(M)$. In other words, we want a regular expression that describes the language recognized by this finite

[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

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.

Ungraded bonus: can you find more than one such regular expression?

(c) (*Graded for completeness*) Keeping the same set of states Q, input alphabet Σ, same start state q_0, and same transition function δ, choose a new set of accepting states F_{new1} so that the new finite automaton $M_1 = (Q, \Sigma, \delta, q_0, F_{new1})$ that results recognizes a **proper superset** of $L(M)$, or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your choice of F_{new1} *and* a precise and clear explanation of why every string that is accepted by M is also accepted by M_1 *and* an example of a string that is accepted by M_1 and is rejected by M; or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

(d) (*Graded for correctness*) Keeping the same set of states Q, input alphabet Σ, same start state q_0, and same transition function δ, choose a new set of accepting states F_{new2} so that the new finite automaton $M_2 = (Q, \Sigma, \delta, q_0, F_{new2})$ that results recognizes a **nonempty proper subset** of $L(M)$, or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your choice of F_{new2} *and* an example string accepted by M_2 *and* a precise and clear explanation of why every string that is accepted by M_2 is also accepted by M *and* an example of a string that is accepted by M and is rejected by M_2; or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

2. **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,

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

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.

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*) 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 correctness*) Design a DFA that recognizes the set of binary (base 2) expansions of positive integers that are less than 10. Your DFA must use **fewer than ten states**. 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.

(c) (*Graded for completeness*) Find a positive integer B greater than 1 so that there is a DFA that recognizes the set of base B expansions of positive integers that are less than 10 and it uses as few states as possible. A complete solution will include the state diagram of your DFA and a brief justification of your choice of base.

Hint: sometimes rewriting the defining membership condition for a set in different ways helps us find alternate representations of that set.

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

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

$$L_1 = \{w \in \{a, b, c\}^* \mid w \text{ starts with } a \textbf{ and } \text{ends with } a\}$$

A complete solution will include the state diagram of your NFA and a brief justification of your construction that explains 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.

(b) (*Graded for correctness*) Design a NFA that recognizes the language

$$L_2 = \{w \in \{a, b, c\}^* \mid w \text{ has no consecutive repeated characters}\}$$

For example, the empty string, a, bac, and $abca$ are each elements of this language but aa and abb and $abbc$ are not elements of this language.

A complete solution will include the state diagram of your NFA and a brief justification of your construction that explains 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*) Consider the language

$$L_1 \cup L_2 = \{w \in \{a,b,c\}^* \mid w \text{ starts with } a \text{ and ends with } a \\ \textbf{or} \text{ has no consecutive repeated characters}\}$$

Give at least two representations of this language among the following:

- A regular expression that describes $L_1 \cup L_2$
- A DFA that recognizes $L_1 \cup L_2$
- A NFA that recognizes $L_1 \cup L_2$

You can design your automata directly or use the constructions from class and chapter 1 in the book to build these automata from automata for the simpler languages.

A complete solution will include at least two of the representations as well as a brief justification of each construction.

(d) (*Graded for completeness*) Consider the language

$$L_1 \cap L_2 = \{w \in \{a,b,c\}^* \mid w \text{ starts with } a \text{ and ends with } a \\ \textbf{and} \text{ has no consecutive repeated characters}\}$$

Give at least two representations of this language among the following:

- A regular expression that describes $L_1 \cap L_2$
- A DFA that recognizes $L_1 \cap L_2$
- A NFA that recognizes $L_1 \cap L_2$

You can design your automata directly or use the constructions from class and chapter 1 in the book to build these automata from automata for the simpler languages.

A complete solution will include at least two of the representations as well as a brief justification of each construction.

4. **General constructions** (13 points): In this question, you'll practice working with formal general constructions for automata and translating between state diagrams and formal definitions.

Recall the definitions of operations we've talked about that produce new languages from old: for each language L over an alphabet Σ, we have the associated sets of strings (also over Σ)

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

and

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

and

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

Also, recall the set operations union and intersection: for any sets X and Y

$$X \cup Y = \{w \mid w \in X \text{ or } w \in Y\}$$

$$X \cap Y = \{w \mid w \in X \text{ and } w \in Y\}$$

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be DFA.

For simplicity, assume that $Q_1 \cap Q_2 = \emptyset$ and that $q_0 \notin Q_1 \cup Q_2$.

Consider the following definitions of new automata parameterized by these DFA:

- The NFA $N_\alpha = (Q_1 \cup Q_2 \cup \{q_0\}, \Sigma, \delta_\alpha, q_0, F_1 \cup F_2)$ with the transition function given by

$$\delta_\alpha(\ (q,x)\ ) = \begin{cases} \{q_1, q_2\} & \text{if } q = q_0,\ x = \varepsilon \\ \emptyset & \text{if } q = q_0,\ x \in \Sigma \\ \{\delta_1(\ (q,x)\ )\} & \text{if } q \in Q_1,\ x \in \Sigma \\ \{\delta_2(\ (q,x)\ )\} & \text{if } q \in Q_2,\ x \in \Sigma \\ \emptyset & \text{if } q \in Q_1 \cup Q_2,\ x = \varepsilon \end{cases}$$

- The NFA $N_\beta = (Q_1 \times Q_2, \Sigma, \delta_\beta, (q_1, q_2), F_1 \times F_2)$ with the transition function given by

$$\delta_\beta(\ (\ (r,s)\ , x\ )\ ) = \{(\ \delta_1(\ (r,x)\ ), \delta_2(\ (s,x)\ )\ )\}$$

  and

$$\delta_\beta(\ (\ (r,s)\ , \varepsilon\ )\ ) = \emptyset$$

  for $r \in Q_1, s \in Q_2, x \in \Sigma$.

- The NFA $N_\gamma = (Q_1 \cup \{q_0\}, \Sigma, \delta_\gamma, q_0, \{q \in Q_1 \mid \exists w \in \Sigma^*(\delta_1^*((q,w)) \in F_1)\})$, and

$$\delta_\gamma((q,a)) = \begin{cases} \{\delta_1((q,a))\} & \text{if } q \in Q_1,\ a \in \Sigma \\ \{q' \in Q_1 \mid \exists w \in \Sigma^*(\delta_1^*((q_1,w)) = q')\} & \text{if } q = q_0,\ a = \varepsilon \\ \emptyset & \text{if } q = q_0,\ a \in \Sigma \\ \emptyset & \text{if } q \in Q_1,\ a = \varepsilon \end{cases}$$

  Hint: the notation δ_1^* refers to the iterated transition function.

(a) (*Graded for correctness*) Illustrate the construction of N_α by defining a specific pair of example DFAs M_1 and M_2 and applying the construction above to create the new NFA N_α. Your example DFA should

- Have the same input alphabet as each other,
- Each have exactly three states (all reachable from the respective start state),
- Accept at least one string and reject at least one string,
- Recognize different languages from one another, and
- Not have any states labelled q_0, and
- Not share any state labels.

Apply the construction above to create the new NFA. A complete submission will include the state diagrams of your example DFA M_1 and M_2 and the state diagram of the NFA N_α resulting from this construction and a precise and clear description of $L(M_1)$ and $L(M_2)$ and $L(N_\alpha)$, 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) (*Graded for correctness*) Illustrate the construction of N_β by defining a specific pair of example DFAs M_1 and M_2 and applying the construction above to create the new NFA N_β. Your example DFA should

- Have the same input alphabet as each other,
- Each have exactly two states (all reachable from the respective start state),
- Accept at least one string and reject at least one string,
- Recognize different languages from one another, and
- Not have any states labelled q_0, and
- Not share any state labels.

Apply the construction above to create the new NFA. A complete submission will include the state diagrams of your example DFA M_1 and M_2 and the state diagram of the NFA N_β resulting from this construction and a precise and clear description of $L(M_1)$ and $L(M_2)$ and $L(N_\beta)$, 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.

(c) (*Graded for correctness*) Illustrate the construction of N_γ by defining a specific example DFA M_1 and applying the construction above to create the new NFA N_γ. Your example DFA should

- Have exactly four states (all reachable from the respective start state),
- Accept at least one string and reject at least one string,
- 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 DFA M_1 and the state diagram of the NFA N_γ resulting from this construction and a precise and clear description of $L(M_1)$ and $L(N_\gamma)$, 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.

(d) (*Graded for completeness*) If possible, associate each construction above with one of the operations whose definitions we recalled at the start of the question. For example, is it the case that (for all choices of DFA M_1 and M_2) $L(N_\alpha) = L(M_1) \cup L(M_2)$? or $L(N_\alpha) = L(M_1) \cap L(M_2)$? etc.

A complete solution will consider each of the constructions $N_\alpha, N_\beta, N_\gamma$ in turn, and for each, either name the operation that's associated with the construction (and explain why) or explain why none of the operations mentioned is associated with the construction.

HW3CSE105W25: Homework assignment 3 Due: February 6th at 5pm, via Gradescope

In this assignment,

You will demonstrate the richness of the class of regular languages, as well as its boundaries, and explore push-down automata and their design.

Resources: To review the topics for this assignment, see the class material from Week 3 and Week 4. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Chapter 1 and Section 2.2. Chapter 1 exercises 1.28, 1.29, 1.30. Chapter 1 problem 1.53, 1.54, 1.55. Chapter 2 exercises 2.7, 2.10.

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

All submitted homework for this class must be typed. You can use a word processing editor if you like (Microsoft Word, Open Office, Notepad, Vim, Google Docs, etc.) but you might find it useful to take this opportunity to learn LaTeX. LaTeX is a markup language used widely in computer science and mathematics. The homework assignments are typed using LaTeX and you can use the source files as templates for typesetting your solutions. To generate state diagrams of machines, you can (1) use the LaTex tikzpicture environment (see templates in the class notes), or (2) use the software tools Flap.js or JFLAP described in the class syllabus (and include a screenshot in your PDF), or (3) you can carefully and clearly hand-draw the diagram and take a picture and include it in your PDF. We recommend that you submit early drafts to Gradescope so that in case of any technical difficulties, at least some of your work is present. You may update your submission as many times as you'd like up to the deadline.

Integrity reminders

- Problems should be solved together, not divided up between the partners. The homework is designed to give you practice with the main concepts and techniques of the course, while

getting to know and learn from your classmates.

- On the "graded for correctness" questions, you may only collaborate with CSE 105 students in your group. You may ask questions about the homework in office hours (of the instructor, TAs, and/or tutors) and on Piazza (as private notes viewable only to the Instructors). You *cannot* use any online resources about the course content other than the class material from this quarter – this is primarily to ensure that we all use consistent notation and definitions (aligned with the textbook) and also to protect the learning experience you will have when the 'aha' moments of solving the problem authentically happen.

- Do not share written solutions or partial solutions for homework with other students in the class who are not in your group. Doing so would dilute their learning experience and detract from their success in the class.

You will submit this assignment via Gradescope (https://www.gradescope.com) in the assignment called "hw3CSE105W25".

Assigned questions

1. **Static analysis** (10 points): In software engineering, static analysis is an approach to debugging and testing where the properties of a piece of code are inferred without actually running it. In the context of finite automata, we can think of static analysis as the process of inferring properties of the language recognized by a finite automaton from properties of the graph underlying its state diagram. The Pumping Lemma is one example of static analysis. In this question, you'll explore other examples of how properties of the graph underlying the state diagram of a machine can give us information about the language recognized by the machine.

(a) (*Graded for completeness*) [5] Suppose you are given an NFA N_0 over an alphabet Σ and each accepting state in N_0 is not reachable from the start state of N_0. What can you conclude about the language of the NFA?

(b) (*Graded for correctness*) [6] Prove or disprove: For any alphabet Σ and any DFA M over Σ, if every state in M is accepting then $L(M) = \Sigma^*$. A complete answer will clearly indicate whether the statement is true or false and then will justify with a complete and correct argument.

[5]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.

[6]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) (*Graded for correctness*) Prove or disprove: For any alphabet Σ and any NFA N over Σ, if every state in N is accepting then $L(N) = \Sigma^*$. A complete answer will clearly indicate whether the statement is true or false and then will justify with a complete and correct argument.

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 3 and for which 5 is a pumping length and 4 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 5 is a pumping length, (3) a justification for why 4 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:

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

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 .

Recall the definitions of operations we've talked about that produce new languages from old: for each language L over an alphabet Σ, we have the associated sets of strings (also over Σ)

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

and

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

and

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

Also, recall the set operations union and intersection: for any sets X and Y

$$X \cup Y = \{w \mid w \in X \text{ or } w \in Y\}$$

$$X \cap Y = \{w \mid w \in X \text{ and } w \in Y\}$$

(a) (*Graded for correctness*) Use the general constructions that we developed to prove the closure of the class of regular languages under various operations to produce the state diagram of a NFA that recognizes the language

$$(\ SUBSTRING(\{0, 01, 111\}) \ )^*$$

Hint: Question 4 from Homework 2 might be helpful.

For full credit, submit (1) a state diagram of an NFA that recognizes $\{0, 01, 111\}$, (2) a state diagram of an NFA that recognizes $SUBSTRING(\{0, 01, 111\})$, and (3) a state diagram of an NFA that recognizes $(\ SUBSTRING(\{0, 01, 111\}) \ )^*$, and a brief justification of each state diagram that references the language being recognized or the general constructions being used.

(b) (*Graded for completeness*) Prove that the class of nonregular languages over $\{0, 1\}$ is not closed under the $SUBSTRING$ operation by giving an example language A that is nonregular but for which $SUBSTRING(A)$ is regular. A complete solution will give (1) a clear and precise description of the language, (2) a justification for why it is nonregular (either by proving this directly or by referring to specific examples from the class or textbook), (3) a description of the result of applying the $SUBSTRING$ operation to the language, and (4) a justification for why this resulting language is regular.

(c) (*Graded for correctness*) Prove that the class of nonregular languages over $\{0, 1\}$ is not closed under the $EXTEND$ operation by giving an example language B that is nonregular but for which $EXTEND(B)$ is regular. A complete solution will give (1) a clear and precise description of the language, (2) a justification for why it is nonregular (either by proving this directly or by referring to specific examples from the class or textbook), (3) a description of the result of applying the $EXTEND$ operation to the language, and (4) a justification for why this resulting language is regular.

(d) (*Graded for completeness*) Prove that the class of nonregular languages over $\{0, 1\}$ is not closed under the Kleene star operation by giving an example language C that is nonregular but for which C^* is regular. A complete solution will give (1) a clear and precise description of the language, (2) a justification for why it is nonregular (either by proving this directly or by referring to specific examples from the class or textbook), (3) a description of the result of applying the Kleene star operation to the language, and (4) a justification for why this resulting language is regular.

5. **Regular and nonregular languages and Push-down automata (PDA)** (6 points): On page 7 of the week 4 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*) Pick one of the regular languages and design a regular expression that describes it and a DFA that recognizes it. Briefly justify your regular expression by connecting the subexpressions of it to the intended language and referencing relevant definitions. Briefly justify your DFA 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.

(b) (*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.

HW4CSE105W25: Homework assignment 4 Due: February 20th 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 explore recognizable and decidable languages.

Resources: To review the topics for this assignment, see the class material from Weeks 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, 2.9, 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. On the "graded for correctness" questions, you may only collaborate with CSE 105 students in your group; if your group has questions about a problem, you may ask in drop-in help hours or post a private post (visible only to the Instructors) on Piazza. On the "graded for completeness" questions, you may collaborate with all other CSE 105 students this quarter, and you may make public posts about these questions on Piazza.

All submitted homework for this class must be typed. You can use a word processing editor if you like (Microsoft Word, Open Office, Notepad, Vim, Google Docs, etc.) but you might find it useful to take this opportunity to learn LaTeX. LaTeX is a markup language used widely in computer science and mathematics. The homework assignments are typed using LaTeX and you can use the source files as templates for typesetting your solutions. To generate state diagrams of machines, you can (1) use the LaTex tikzpicture environment (see templates in the class notes), or (2) use the software tools Flap.js or JFLAP described in the class syllabus (and include a screenshot in your PDF), or (3) you can carefully and clearly hand-draw the diagram and take a picture and include it in your PDF. We recommend that you submit early drafts to Gradescope so that in case of any technical difficulties, at least some of your work is present. You may update your submission as many times as you'd like up to the deadline.

Integrity reminders

- Problems should be solved together, not divided up between the partners. The homework is designed to give you practice with the main concepts and techniques of the course, while

getting to know and learn from your classmates.

- On the "graded for correctness" questions, you may only collaborate with CSE 105 students in your group. You may ask questions about the homework in office hours (of the instructor, TAs, and/or tutors) and on Piazza (as private notes viewable only to the Instructors). You *cannot* use any online resources about the course content other than the class material from this quarter – this is primarily to ensure that we all use consistent notation and definitions (aligned with the textbook) and also to protect the learning experience you will have when the 'aha' moments of solving the problem authentically happen.
- Do not share written solutions or partial solutions for homework with other students in the class who are not in your group. Doing so would dilute their learning experience and detract from their success in the class.

You will submit this assignment via Gradescope (https://www.gradescope.com) in the assignment called "hw4CSE105W25".

Assigned questions

1. **Regular and nonregular languages and context-free grammars (CFG)** (6 points): On page 7 of the week 4 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*) [7] Pick one of the regular languages and design a context-free grammar that generates it. Briefly justify your grammar by describing the role of each of the rules and connecting it to the intended language and referencing relevant definitions.

(b) (*Graded for completeness*) Pick one of the nonregular languages and design a context-free grammar that generates it. Briefly justify your grammar by describing the role of each of the rules and connecting it to the intended language and referencing relevant definitions.

2. **General constructions for context-free languages** (15 points):

In class in week 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.

[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.

(a) (*Graded for completeness*) 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*) [8] 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

$$\{w \in \{a, b\}^* \mid w \text{ has at least one } a \text{ and does not end in } bb\}$$

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.*

(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 $\delta_2 : Q_2 \times \Sigma_\varepsilon \times \Gamma_{2\varepsilon} \to \mathcal{P}(Q_2 \times \Gamma_{2\varepsilon})$

[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.

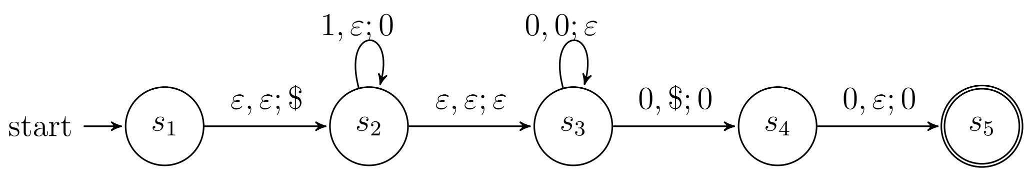
given by

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

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.

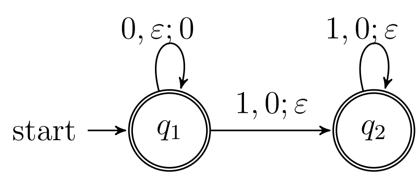
i. (*Graded for correctness*) Consider the PDA M_1 with input alphabet $\{0,1\}$ and stack alphabet $\{0,1,\$\}$ whose state diagram is



Draw the state diagram for the PDA M_2 that results from applying the construction to M_1. Also, give an example string of length 4 that is accepted by both M_1 and M_2 and justify your choice by describing an accepting computation for each of the PDAs on your input string.

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 concatenation 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 to prove this by showing that

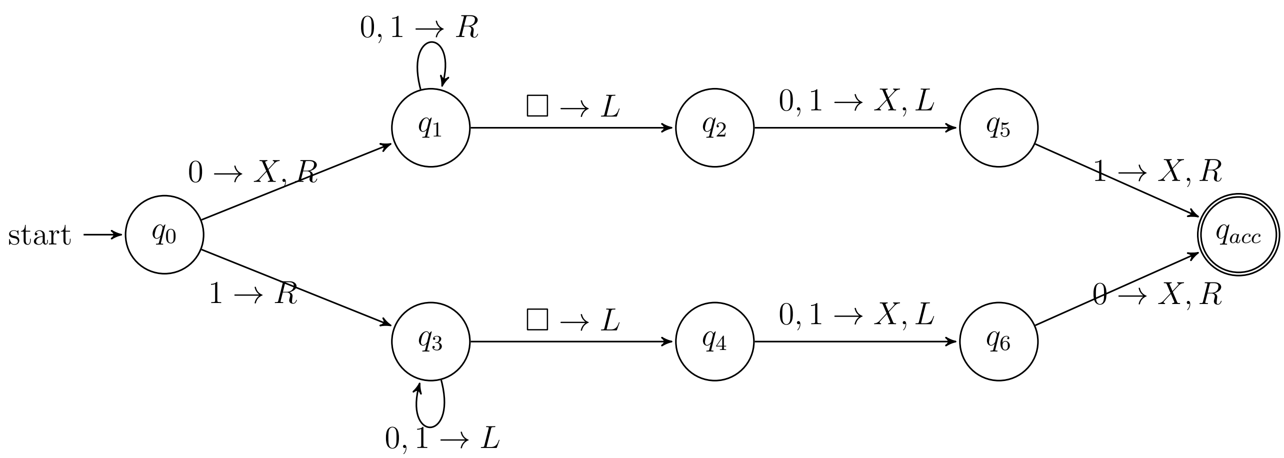
$$L(M_1) \circ L(N)$$

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. Then, 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.

A complete response will give an example string that witnesses that $L(M_1) \circ L(N)$ is not equal to the language recognized by the PDA resulting from the wrong construction (described above) **and** the state diagram of the PDA that results from applying that construction to M_2 and N (instead of M_1), with a brief justification about why that approach works.

3. **Turing machines** (9 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 3 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.

4. **Implementation-level descriptions of deciders and recognizers** (12 points): Consider the language

$$\{a^i b^j \mid i \geq 0, j > 1\}$$

over the alphabet $\{a, b\}$.

(a) (*Graded for correctness*) Give an example of a Turing machine that **decides** this language. A complete solution will include **both** a state diagram and an implementation-level description of this Turing machine, along with a brief explanation of why it recognizes this language, and why it is a decider.

(b) (*Graded for correctness*) Give an example of a Turing machine that **recognizes but does not decide** this language. A complete solution will include **both** a state diagram and an implementation-level description of this Turing machine, along with a brief explanation of why it recognizes this language, and why it is not a decider.

5. **Classifying languages** (8 points): Our first example of a more complicated Turing machine was of a Turing machine that recognized the language $\{w\#w \mid w \in \{0,1\}^*\}$ (Figure 3.10 in the textbook), 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*) Choose an example string of length 2 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*) 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.

(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 $b \to b, R$).

HW5CSE105W25: Homework assignment 5 Due: February 27th at 5pm, via Gradescope

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

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

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

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

All submitted homework for this class must be typed. You can use a word processing editor if you like (Microsoft Word, Open Office, Notepad, Vim, Google Docs, etc.) but you might find it useful to take this opportunity to learn LaTeX. LaTeX is a markup language used widely in computer science and mathematics. The homework assignments are typed using LaTeX and you can use the source files as templates for typesetting your solutions. To generate state diagrams of machines, you can (1) use the LaTex tikzpicture environment (see templates in the class notes), or (2) use the software tools Flap.js or JFLAP described in the class syllabus (and include a screenshot in your PDF), or (3) you can carefully and clearly hand-draw the diagram and take a picture and include it in your PDF. We recommend that you submit early drafts to Gradescope so that in case of any technical difficulties, at least some of your work is present. You may update your submission as many times as you'd like up to the deadline.

Integrity reminders

- Problems should be solved together, not divided up between the partners. The homework is designed to give you practice with the main concepts and techniques of the course, while getting to know and learn from your classmates.

- On the "graded for correctness" questions, you may only collaborate with CSE 105 students in your group. You may ask questions about the homework in office hours (of the instructor, TAs, and/or tutors) and on Piazza (as private notes viewable only to the Instructors). You *cannot* use any online resources about the course content other than the class material from this quarter – this is primarily to ensure that we all use consistent notation and definitions (aligned with the textbook) and also to protect the learning experience you will have when the 'aha' moments of solving the problem authentically happen.
- Do not share written solutions or partial solutions for homework with other students in the class who are not in your group. Doing so would dilute their learning experience and detract from their success in the class.

You will submit this assignment via Gradescope (https://www.gradescope.com) in the assignment called "hw5CSE105W25".

Assigned questions

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

(a) (*Graded for completeness*) [9] Let's define a new machine model, and call it the **May-stay** machine. The May-stay machine model is the same as the usual Turing machine model, except that on each transition, the tape head may move L, move R, or Stay.

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

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

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

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

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

[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.

(b) (*Graded for correctness*) [10] Let's define a new machine model, and call it the **Double-move** machine. The Double-move machine model is the same as the usual Turing machine model, except that on each transition, the tape head may move L, move R one cell, or move R two cells. Formally: a Double-move machine is given by the 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where Q is a finite set with $q_0 \in Q$ and $q_{accept} \in Q$ and $q_{reject} \in Q$ and $q_{accept} \neq q_{reject}$, Σ and Γ are alphabets and $\Sigma \subseteq \Gamma$ and $\square \in \Gamma$ and $\square \notin \Sigma$, and the transition function has signature

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

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

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

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

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

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

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

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

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

[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.

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

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

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

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

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

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

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

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

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

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

and

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

and

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

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

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

$M_a =$ "On input w :
1. For $n = 1, 2, \ldots$
2. For $j = 1, 2, \ldots n$
3. For $k = 1, 2, \ldots, n$
4. Run the computation of M on $s_j w s_k$ for at most n steps
5. If that computation halts and accepts within n steps, accept.
6. Otherwise, continue with the next iteration of this inner loop"

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

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

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

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

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

4. **Computational problems** (8 points): Recall the definitions of some example computational problems from class

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

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

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

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

$$
\begin{aligned}
M_{DFA} = \text{“On input } \langle D \rangle \text{ where } D \text{ is a DFA :}\\
&1.\ \text{for } i = 1, 2, 3, \ldots\\
&2.\quad \text{Run } D \text{ on } s_i\\
&3.\quad \text{If it accepts, accept.}\\
&4.\quad \text{If it rejects, go to the next iteration of the loop”}
\end{aligned}
$$

and

$$
\begin{aligned}
M_{TM} = \text{“On input } \langle T \rangle \text{ where } T \text{ is a Turing machine :}\\
&1.\ \text{for } i = 1, 2, 3, \ldots\\
&2.\quad \text{Run } T \text{ for } i \text{ steps on each input } s_1, s_2, \ldots, s_i \text{ in turn}\\
&3.\quad \text{If } T \text{ has accepted any of these, accept.}\\
&4.\quad \text{Otherwise, go to the next iteration of the loop”}
\end{aligned}
$$

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

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

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

is decidable.

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

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

is decidable.

HW6CSE105W25: Homework assignment 6 Due: March 13, 2025 at 5pm, via Gradescope

In this assignment,

You will practice analyzing, designing, and working with reductions to compare the difficulty level of computational problems. 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 8 and 9. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Sections 4.2, 5.3, 5.1. Chapter 4 exercises 4.9, 4.12. Chapter 5 exercises 5.4, 5.5, 5.6, 5.7. Chapter 5 problems 5.22, 5.23, 5.24, 5.28

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

All submitted homework for this class must be typed. You can use a word processing editor if you like (Microsoft Word, Open Office, Notepad, Vim, Google Docs, etc.) but you might find it useful to take this opportunity to learn LaTeX. LaTeX is a markup language used widely in computer science and mathematics. The homework assignments are typed using LaTeX and you can use the source files as templates for typesetting your solutions. To generate state diagrams of machines, you can (1) use the LaTex tikzpicture environment (see templates in the class notes), or (2) use the software tools Flap.js or JFLAP described in the class syllabus (and include a screenshot in your PDF), or (3) you can carefully and clearly hand-draw the diagram and take a picture and include it in your PDF. We recommend that you submit early drafts to Gradescope so that in case of any technical difficulties, at least some of your work is present. You may update your submission as many times as you'd like up to the deadline.

Integrity reminders

- Problems should be solved together, not divided up between the partners. The homework is designed to give you practice with the main concepts and techniques of the course, while

getting to know and learn from your classmates.

- On the “graded for correctness” questions, you may only collaborate with CSE 105 students in your group. You may ask questions about the homework in office hours (of the instructor, TAs, and/or tutors) and on Piazza (as private notes viewable only to the Instructors). You *cannot* use any online resources about the course content other than the class material from this quarter – this is primarily to ensure that we all use consistent notation and definitions (aligned with the textbook) and also to protect the learning experience you will have when the ‘aha’ moments of solving the problem authentically happen.
- Do not share written solutions or partial solutions for homework with other students in the class who are not in your group. Doing so would dilute their learning experience and detract from their success in the class.

You will submit this assignment via Gradescope (https://www.gradescope.com) in the assignment called “hw6CSE105W25”.

Assigned questions

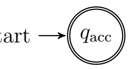
1. **What’s wrong with these reductions? (if anything)** (16 points): Suppose your friends are practicing coming up with mapping reductions $A \leq_m B$ and their witnessing functions $f : \Sigma^* \to \Sigma^*$. For each of the following attempts, determine if it has error(s) or is correct. Do so by labelling each attempt with all and only the labels below that apply, and justifying this labelling.

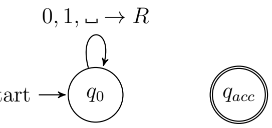
- *Error Type 1:* The given function can’t witness the claimed mapping reduction because there exists an $x \in A$ such that $f(x) \notin B$.
- *Error Type 2:* The given function can’t witness the claimed mapping reduction because there exists an $x \notin A$ such that $f(x) \in B$.
- *Error Type 3:* The given function can’t witness the claimed mapping reduction because the specified function is not computable.
- *Correct:* The claimed mapping reduction is true and is witnessed by the given function.

Clearly present your answer by providing a brief (3-4 sentences or so) justification for whether **each** of these labels applies to each example.

(a) (*Graded for completeness*) [11] $A_{\mathrm{TM}} \leq_m HALT_{\mathrm{TM}}$ and

$f(x) =$

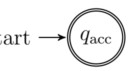
- ⟨

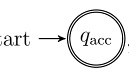
, ε⟩ if $x = \langle M, w \rangle$ for a Turing machine M and string w and $w \in L(M)$
- ⟨

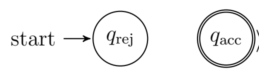
⟩ otherwise

(b) (*Graded for completeness*) $A_{\mathrm{TM}} \leq_m EQ_{\mathrm{TM}}$ with

$f(x) =$

- ⟨

, M_w⟩ if $x = \langle M, w \rangle$ for a Turing machine M and string w
- ⟨

, 

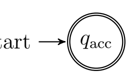
⟩ otherwise.

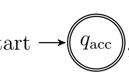
Where for each Turing machine M, we define

$M_w =$ "On input y
1. Simulate M on w.
2. If it accepts, accept.
3. If it rejects, reject."

(c) (*Graded for correctness*) [12] $HALT_{\mathrm{TM}} \leq_m EQ_{\mathrm{TM}}$ with

$f(x) =$

- ⟨

, M_w⟩ if $x = \langle M, w \rangle$ for a Turing machine M and string w
- ⟨

, 

⟩ otherwise.

Where for each Turing machine M, we define

$M_w =$ "On input y
1. If y is not the empty string, accept.
2. Else, simulate M on w.
3. If it accepts, accept.
4. If it rejects, reject."

---

[11] 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.

[12] 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.

(d) (*Graded for correctness*) $\{ww \mid w \in \{0,1\}^*\} \leq \Sigma^*$ and $f(x) = 11$ for each $x \in \{0,1\}^*$.

(e) (*Graded for correctness*) $\Sigma^* \leq_m \{ww \mid w \in \{0,1\}^*\}$ and $f(x) = 11$ for each $x \in \{0,1\}^*$.

2. **Using mapping reductions** (14 points): Consider the following computational problems we've discussed

$$\begin{aligned} A_{TM} &= \{\langle M, w\rangle \mid M \text{ is a Turing machine, } w \text{ is a string and } M \text{ accepts } w\} \\ HALT_{TM} &= \{\langle M, w\rangle \mid M \text{ is a Turing machine, } w \text{ is a string and } M \text{ halts on } w\} \\ E_{TM} &= \{\langle M\rangle \mid M \text{ is a Turing machine and } L(M) = \emptyset\} \\ EQ_{TM} &= \{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are both Turing machines and } L(M_1) = L(M_2)\} \end{aligned}$$

and the new computational problem

$$\begin{aligned} IncludesEmptyString_{TM} = \{\langle M\rangle \mid\ & M \text{ is a Turing machine and} \\ & M \text{ accepts the empty string (and maybe other strings too)}\} \end{aligned}$$

(a) (*Graded for correctness*) Give an example of a string that is an element of $IncludesEmptyString_{TM}$ and a string that is not an element of $IncludesEmptyString_{TM}$ and briefly justify your choices.

(b) (*Graded for completeness*) Prove that $IncludesEmptyString_{TM}$ is not decidable by showing that $A_{TM} \leq_m IncludesEmptyString_{TM}$.

(c) (*Graded for correctness*) Give a different proof that $IncludesEmptyString_{TM}$ is not decidable by showing that $HALT_{TM} \leq_m IncludesEmptyString_{TM}$.

(d) (*Graded for completeness*) Is $IncludesEmptyString_{TM}$ recognizable? Justify your answer.

3. **Using mapping reductions** (14 points): Consider the following computational problems we've discussed

$$\begin{aligned} A_{TM} &= \{\langle M, w\rangle \mid M \text{ is a Turing machine, } w \text{ is a string and } M \text{ accepts } w\} \\ HALT_{TM} &= \{\langle M, w\rangle \mid M \text{ is a Turing machine, } w \text{ is a string and } M \text{ halts on } w\} \\ E_{TM} &= \{\langle M\rangle \mid M \text{ is a Turing machine and } L(M) = \emptyset\} \\ EQ_{TM} &= \{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are both Turing machines and } L(M_1) = L(M_2)\} \end{aligned}$$

and the new computational problem

$$NotIncludesEmptyString_{TM} = \{\langle M\rangle \mid M \text{ is a Turing machine and } M \text{ does not accept the empty string}\}$$

(a) (*Graded for correctness*) Prove that $NotIncludesEmptyString_{TM}$ is not the complement of E_{TM} and is also not the complement of $IncludesEmptyString_{TM}$.

(b) (*Graded for completeness*) Prove that $NotIncludesEmptyString_{TM}$ is not decidable by showing that $\overline{HALT_{TM}} \leq_m NotIncludesEmptyString_{TM}$.

(c) (*Graded for correctness*) Give a different proof that $NotIncludesEmptyString_{TM}$ is not decidable by showing that $\overline{A_{TM}} \leq_m NotIncludesEmptyString_{TM}$.

(d) (*Graded for completeness*) Is $NotIncludesEmptyString_{TM}$ recognizable? Justify your answer.

4. **Examples of languages** (6 points):

For each part of the question, use precise mathematical notation or English to define your examples and then briefly justify why they work.

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

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

and

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

and

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

(a) (*Graded for correctness*) Two undecidable languages L_1 and L_2 over the same alphabet whose union $L_1 \cup L_2$ is co-recognizable, or write **NONE** if there is no such example (and explain why).

(b) (*Graded for correctness*) An unrecognizable language L_3 for which $EXTEND(L_3)$ is regular or write **NONE** if there is no such example (and explain why).

(c) (*Graded for completeness*) A co-recognizable language L_4 that is NP-complete, or write **NONE** if there is no such example (and explain why). Recall the definition: A language L over an alphabet Σ is called **co-recognizable** if its complement, defined as $\Sigma^* \setminus L = \{x \in \Sigma^* \mid x \notin L\}$, is Turing-recognizable.

This part of the question uses definitions from Week 10 of the course.

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: Checking the consistency of two regular properties, and
Task 2: Illustrating a mapping reduction

In each task, you'll be implementing some of the theoretical concepts we've talked about in a programming language of your choosing, and then presenting your reasoning and demonstrating your code. 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.

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 should be the architect of your approach to the project. You can refer to 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. However, do not copy / screenshot material that was not produced by you directly to your project writeup. Instead, you can refer to it in your own words and include a citation to the resource you referenced.

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. Also, a word of caution, make sure you validate and check any code produced by these aids. Last quarter there were a lot of examples of project submissions that fed the prompt of the project directly to a LLM code generator and got wrong implementations back.

Submitting the project You will submit a PDF plus a video file (.mp4) for each. All file submissions will be in Gradescope. Upload the four files themselves. It is your responsibility to ensure that the files are playable within Gradescope. No Google Drive links, YouTube links, or .mov files.

Your videos: 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`

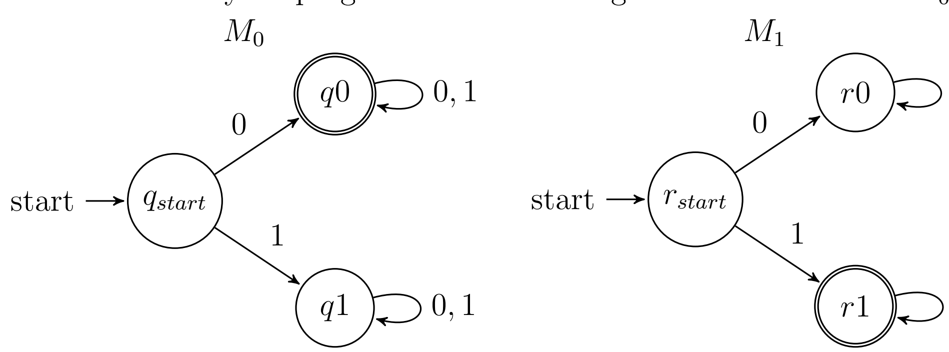
Please send an email to the instructor (minnes@ucsd.edu) if you have concerns about the video / screencast components of this project or cannot complete projects in this style for some reason.

Task 1: Checking the consistency of two regular properties When we have a list of desired properties, it's helpful to know whether there are any examples that satisfy *all* of them at once; in other words, whether the properties are consistent with each other. For example, the properties of a string starting with 0 and a string starting with 1 are *not* consistent, because there isn't any example of a string that simultaneously starts with 0 and with 1. In this part of the project, you'll use the decidability of the emptiness problem for DFA to build an algorithm that checks if two given regular properties are consistent.

Specifically:

1. Write a program in Java, Python, JavaScript, C++ , or another programming language of your choosing that checks the consistency of two arbitrary regular properties. The function input must be a **pair of strings** and part of your work in this program is to design string representations for any arbitrary DFA since those DFAs will be how you represent the properties. The function output must be a **boolean**: true (if the pair of strings represent consistent regular properties) or false (if the pair of strings do not represent consistent regular properties).

   - You might find it useful to use the algorithm for building a DFA that recognizes the **intersection** of the languages of two DFA and the algorithm for testing the **emptiness** of the language of a DFA.
   - One test case for your program is the following: Consider the DFA M_0 and M_1

     and let w_0 be the string representing M_0 and let w_1 be the string representing M_1. Then the result of your program on the input w_0, w_1 should be **false** because the properties represented by M_0 and M_1 are inconsistent.[13]

- The algorithm you implement needs to work with any pair of strings given as input (you should first parse each string in the input to see if it is formatted to represent a DFA). Your explanation of the algorithm should be such that most programmers can replicate the algorithm correctly. 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 or as an appendix in your PDF submission, 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.

2. To demonstrate your program, you will show how it runs (at least) twice: once with the test input w_0, w_1 described above, and once with test input x_0, x_1 that you define to demonstrate when the program outputs **true**. Your choice of x_0, x_1 needs to satisfy the following conditions:

   - x_0 and x_1 each represent DFA over the same fixed alphabet,
   - the DFA represented by x_0 and x_1 do not recognize the same language,
   - the DFA represented by x_0 and x_1 do not recognize the empty set or the set of all strings or the set of all strings starting with 0 or the set of all strings starting with 1.

   As part of your demonstration, describe your choice of x_0 and x_1, clearly specifying the DFAs they represent and the languages recognized by these DFAs. Each demo run of the program should include:

   - Side-by-side view of an English / mathematical formulation of the properties and DFA corresponding to the demo run, along with their string representation as input strings to the program.
   - Talk-aloud trace of the running of your program on the input pair of strings representing the two properties for the demo. During the trace, talk about the software design choices you made (e.g. which data structures are you using, etc.) and how they impact the program. Also, give credit to any resources you used to make these design choices or develop the code.
   - Recording of actually running your program on the input pair of strings for this demo, including interpreting the output the program gives and connecting it to whether the properties represented by the input are consistent or not.

[13] To see why, notice that $L(M_0)$ is the set of strings that start with 0 and $L(M_1)$ is the set of strings that start with 1.

Checklist for submission For this task, you will submit a PDF file plus a 3-5 minute video.

(PDF) Submit a single PDF file that clearly describes the design choices you made, includes all relevant code, and includes all relevant information (definition, representation, justification) about the examples used to demonstrate your program and screenshots from running your program on the examples. The writeup should use precise language and notation for all terms and clearly communicate the goal and approach of your program.

(PDF) The documentation for your program should include a description of how input strings are parsed to represent DFA, in general and for the specific examples of w_0, w_1, x_0, x_1.

(Video) The video should 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) The video includes the full program demo twice (once with the input w_0, w_1 and once with the input x_0, x_1), and includes the a mathematical and/or English definition of the regular properties you are using, connected to their representations as strings when input to the program, and the talk-aloud trace of running the program on these inputs and its output connecting back to the notion of consistency of regular properties.

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.

Possible extensions: If you're enjoying working on this and want to go deeper, here are a few additions you can consider. You will not be graded on any of these, and you should still make sure your project has the core functionality described above, but these extensions give you an opportunity to explore further.

- Build a preprocesing step so that the regular properties can be expressed using NFA or regular expressions (in addition to DFA). You'll need to think about how to represent NFA and/or regular expressions as strings, and then how to convert them to DFA.
- Extend your work so that your program can test for consistency of more than two regular properties.

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** from an undecidable language of your choice to

$$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are Turing machines and } L(M_1) = L(M_2)\}$$

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 an undecidable language (other than EQ_{TM}) that we discussed in class or in the homework or in review quizzes or in the textbook . *Note: if you'd like to consider an undecidable language we have not discussed instead, please check with Prof. Minnes first. You must do so no later than the start of Week 10.*

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.

    - You may use our class material for ideas on the algorithm that your program will implement. The algorithm you implement needs to be general enough to decide whether each input string is in the language or not. Your explanation of the algorithm should be such that most programmers can replicate the algorithm correctly.
    - 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 or as an appendix in your PDF submission, 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 $X \leq_m EQ_{TM}$, 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.

Checklist for submission For this task, you will submit a PDF file plus a 3-5 minute video.

(PDF) Submit a single PDF file that clearly describes the mapping reduction, including defining the undecidable language you chose and the computable function that you will implement to witness its mapping reduction to EQ_{TM}, and includes all relevant code and documentation documentation for the program computing the function witnessing this mapping reduction. In particular, include a description of how input strings are parsed and how output strings correspond to input strings and a clear specification of your two example strings, explaining which is is a positive instance (and why) and which is a negative instance (and why not). The writeup should use precise language and notation for all terms and clearly communicate the goal and approach of your program.

(Video) The video should 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) The video includes the mapping reduction you will be working with, and the 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). The full program demo should be part of the video, with an explanation of the code and the software design choices you made and any resources you used, and then live screencasts running your code on each of your example inputs. 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.

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.