"CS/ECE 374 A": Algorithms & Models of Computation, Spring 2025
Midterm 1 Solutions — Feb 24, 2025

Name:	
NetID:	

- Please *clearly PRINT* your name and your NetID in the boxes above.
- This is a closed-book but you are allowed a 1 page (2 sides) hand written cheat sheet that you have to submit along with your exam. If you brought anything except your writing implements, put it away for the duration of the exam. In particular, you may not use *any* electronic devices.
- **Please read the entire exam before writing anything.** Please ask for clarification if any question is unclear. The exam has 6 problems, each worth 10 points.
- **You have 150 minutes (2.5 hours) for the exam.**
- If you run out of space for an answer, continue on the back of the page, or on the blank pages at the end of this booklet, **but please tell us where to look.**
- **Write everything inside the box around each page.** Anything written outside the box may be cut off by the scanner.
- **Proofs are required only if we specifically ask for them.** You may state and use (without proof or justification) any results proved in class or in the problem sets unless we explicitly ask you for one.
- You can do hard things!
- **Do not cheat.** You know the student code and all that jazz. Grades do matter, but not as much as you may think, and your values are more important.

1 Regular Expression

Assume $\Sigma = \{0,1\}$. Give regular expressions for the two languages below and **briefly explain how your expressions work.**

(a) (5 pts) All strings which contain the substring 01010 and which have an even number of blocks of 0's. Recall that a block (or a run) is a non-empty maximal substring of the same symbol.

Solution:

$$1^*(0^+1^+0^+1^+)^*0^+1010^+(1^+0^+)(1^+0^+1^+0^+)^*1^* \; + \\ 1^*(0^+1^+0^+1^+)^*(0^+1^+)0^+1010^+(1^+0^+1^+0^+)^*1^*$$

The 0^+1010^+ in each term ensures that our string contains the substring 01010, as well as capturing any additional zeros in the blocks immediately before and after this substring. Since this adds three blocks of zeros, we know that the rest of the string has to have an odd number of blocks of zeros. The first term captures the case where the string before 0^+1010^+ has an even number of blocks of zeros and the string after 0^+1010^+ has an odd number of these blocks, while the second term captures the reverse of this. ■

Rubric: 5 points: standard regular expression rubric, scaled.

(b) (5 pts) $\{1^{2n}w0^n \mid n \geq 2, w \in \{0,1\}^*\}$

Solution:

$$1111(0+1)^*00$$

Note that if we can write $x = 1^{2n}w0^n$ for some $n \geq 2$, we can in fact write $x = 1^4w'0^2$ by taking $w' = 1^{2n-4}w0^{n-2}$. Since all strings of the form $1^4w'0^2$ are in the language (by taking $n = 2$ in the definition), we have that this language is just all strings that start with (at least) four ones and end with (at least) two zeros, which is exactly what our regular expression captures. ■

Rubric: 5 points: standard regular expression rubric, scaled.

Rubric: Standard regular expression rubric. 10 points =

- 2 points for a syntactically correct regular expression.
- 4 points for a brief English explanation of your regular expression. This is how you argue that your regular expression is correct.
  - For longer expressions, you should explain each of the major components of your expression, and separately explain how those components fit together.
  - We do not want a transcription; don't just translate the regular-expression notation into English.
- 4 points for correctness.
  - -4 for incorrectly answering $\emptyset$ or Σ^*.

- ⋆ -1 for a single mistake: one typo, excluding exactly one string in the target language, or including exactly one string not in the target language. (The incorrectly handled string is almost always the empty string ϵ.)
- ⋆ -2 for incorrectly including/excluding more than one but a finite number of strings.
- ⋆ -4 for incorrectly including/excluding an infinite number of strings.

– Regular expressions that are more complex than necessary may be penalized. Regular expressions that are significantly too complex may get no credit at all. On the other hand, minimal regular expressions are not required for full credit.

2 DFA Design

Give DFAs for the two languages below and **briefly describe the meaning of each state**.

(a) (5 pts) Strings in $\{0,1\}^*$ whose first character equals its second-to-last character. Examples of strings in this language include 00101, 11111, and 10. Examples of strings not in this language include ϵ, 1100, and 10101.

Solution: We give a DFA that keeps track of the first character we read as well as the last two characters we read, and uses these to check the given condition. Using formal tuple notation, this DFA is given by:

- $Q = \{\epsilon, 0, 1\} \cup \{(a, bc) \mid a, b, c \in \{0,1\}\}$
- $s = \epsilon$
- $A = \{(a, bc) \mid a = b\}$
- $\delta(q, d) = \begin{cases} d & \text{if } q = \epsilon \\ (a, ad) & \text{if } q = a \\ (a, cd) & \text{if } q = (a, bc) \end{cases}$

Meaning of the states:

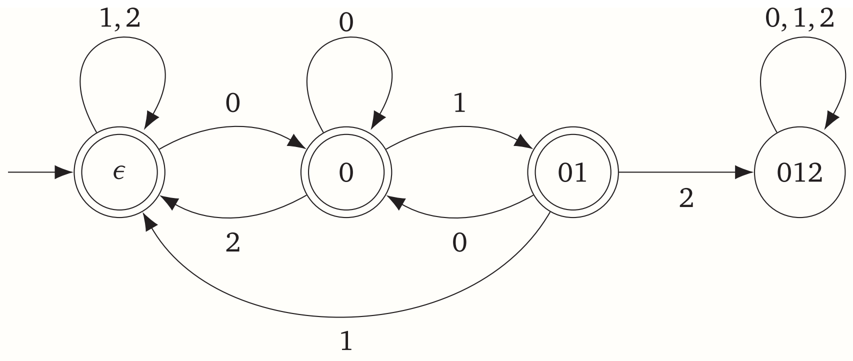
- ϵ: We have not read any characters yet.
- 0: We have read exactly one character, and it was a 0.
- 1: We have read exactly one character, and it was a 1.
- (a, bc): We have read at least two characters, the first character was a, and the most recent two characters have been bc.

■

Rubric: 5 points: standard DFA rubric, scaled. A drawing is also possible for this problem, though it will be a bit unwieldy at 11 states.

(b) (5 pts) Strings in $\{0,1,2\}^*$ that do *not* contain the substring 012.

Solution: We give a DFA that tracks how much of the substring 012 we've seen so far, and only rejects if we ever see the whole thing at once. Formally, we can draw this as follows:



Meaning of the states:

- 012: We have seen the full substring 012.
- 01: We haven't previously seen 012, but the last two characters were 01.
- 0: We haven't previously seen 012, but the last character we read was 0.
- ϵ: We haven't previously seen 012, and the most recent characters we've read don't match with any prefix of 012.

■

Rubric: 5 points: standard DFA rubric, scaled.

Standard DFA/NFA design rubric. 10 points =

- 2 points for an unambiguous description of a DFA or NFA, including the states set Q, the start state s, the accepting states A, and the transition function δ.
  - **Drawings:**
    - Use an arrow from nowhere to indicate the start state s.
    - Use doubled circles to indicate accepting states A.
    - If $A = \varnothing$, say so explicitly.
    - If your drawing omits a junk/trash/reject/hell state, say so explicitly.
    - **Draw neatly!** If we can't read your solution, we can't give you credit for it.
  - **Text descriptions:** You can describe the transition function either using a 2d array, using mathematical notation, or using an algorithm.
    - You must explicitly specify $\delta(q, a)$ for *every* state q and *every* symbol a.
    - If you are describing an NFA with ε-transitions, you must explicitly specify $\delta(q, \varepsilon)$ for *every* state q.
    - If you are describing a DFA, then every value $\delta(q, a)$ must be a single state.
    - If you are describing an NFA, then every value $\delta(q, a)$ must be a set of states.
    - In addition, if you are describing an NFA with ε-transitions, then every value $\delta(q, \varepsilon)$ must be a set of states.
  - **Product constructions:** You must give a complete description of each of the DFAs you are combining (as either drawings, text, or recursive products), together with the accepting states of the product DFA. In particular, we will not assume that product constructions compute intersections by default.

- 4 points for *briefly* explaining the purpose of each state *in English*. This is how you argue that your DFA or NFA is correct.
  - For product constructions, explaining the states in the factor DFAs is both necessary and sufficient.
  - **Yes, we mean it.** A perfectly correct drawing of a perfectly correct DFA with no state explanation is worth at most 6 points.
- 4 points for correctness.
  - −1 for a single mistake: a single misdirected transition, a single missing or extra accepting state, rejecting exactly one string that should be accepted, or accepting exactly one string that should be accepted. (The incorrectly accepted/rejected string is almost always the empty string ε.)
  - −4 for incorrectly accepting every string, or incorrectly rejecting every string.
  - −2 for incorrectly accepting/rejecting more than one but a finite number of strings.
  - −4 for incorrectly accepting/rejecting an infinite number of strings.
- DFAs or NFAs that are more complex than necessary may be penalized. DFAs or NFAs that are *significantly* more complex than necessary may get no credit at all. On the other hand, *minimal* DFAs are *not* required for full credit, unless the problem explicitly asks for them.
- Half credit for describing an NFA when the problem asks for a DFA.

3 NFA Design

(10 pts) Let $k \geq 1$ be an integer and let

$L_k = \{w \in \{0,1\}^* : |w| \geq k$ and number of 0s and 1s differ by at most two in the last k bits of $w\}$

For example if $k = 5$ then the strings 0000011 and 11010 are in L_5 while 11110 and 100000 are not in L_5. Describe an NFA for L_k with $O(k^2)$ states and **briefly describe the meaning of each state**.

Solution: We define an NFA $N = (Q, \delta, s, A)$ as follows, with all unspecified transitions going to $\emptyset$:

- $Q = \{s\} \cup \{-k, \dots, 0, \dots, k\} \times \{1, \dots, k\}$
- $\delta(q,a) = \begin{cases} \{s,(1,1)\} & \text{if } q = s, a = 0 \\ \{s,(-1,1)\} & \text{if } q = s, a = 1 \\ \{(i+1,j+1)\} & \text{if } q = (i,j), a = 0, (i+1,j+1) \in Q \\ \{(i-1,j+1)\} & \text{if } q = (i,j), a = 1, (i-1,j+1) \in Q \end{cases}$
- $s = s$
- $A = \{(q,r) \mid q \in \{-2,-1,0,1,-2\}, r = k\}$

Our NFA guesses when the last k bits begins, and checks to make sure that the number of zeros and ones differ by at most two in those last k bits. The meaning of each state is as follows:

- s: We have guessed that we haven't yet started the last k bits.
- (i,j) We guessed that the last k bits started j bits ago, and since then the number of zeros we've seen minus the number of ones is j.

This construction uses $1 + (2k+1) \cdot k = O(k^2)$ states. (This is not the only correct construction.) ■

Standard DFA/NFA design rubric. 10 points =

- 2 points for an unambiguous description of a DFA or NFA, including the states set Q, the start state s, the accepting states A, and the transition function δ.
  - **Drawings:**
    - ⋆ Use an arrow from nowhere to indicate the start state s.
    - ⋆ Use doubled circles to indicate accepting states A.
    - ⋆ If $A = \varnothing$, say so explicitly.
    - ⋆ If your drawing omits a junk/trash/reject/hell state, say so explicitly.
    - ⋆ **Draw neatly!** If we can't read your solution, we can't give you credit for it.
  - **Text descriptions:** You can describe the transition function either using a 2d array, using mathematical notation, or using an algorithm.
    - ⋆ You must explicitly specify $\delta(q,a)$ for *every* state q and *every* symbol a.

    - If you are describing an NFA with ε-transitions, you must explicitly specify $\delta(q, \varepsilon)$ for *every* state q.
    - If you are describing a DFA, then every value $\delta(q, a)$ must be a single state.
    - If you are describing an NFA, then every value $\delta(q, a)$ must be a set of states.
    - In addition, if you are describing an NFA with ε-transitions, then every value $\delta(q, \varepsilon)$ must be a set of states.
  - **Product constructions:** You must give a complete description of each of the DFAs you are combining (as either drawings, text, or recursive products), together with the accepting states of the product DFA. In particular, we will not assume that product constructions compute intersections by default.

- 4 points for *briefly* explaining the purpose of each state *in English*. This is how you argue that your DFA or NFA is correct.
  - For product constructions, explaining the states in the factor DFAs is both necessary and sufficient.
  - **Yes, we mean it.** A perfectly correct drawing of a perfectly correct DFA with no state explanation is worth at most 6 points.

- 4 points for correctness.
  - −1 for a single mistake: a single misdirected transition, a single missing or extra accepting state, rejecting exactly one string that should be accepted, or accepting exactly one string that should be accepted. (The incorrectly accepted/rejected string is almost always the empty string ε.)
  - −4 for incorrectly accepting every string, or incorrectly rejecting every string.
  - −2 for incorrectly accepting/rejecting more than one but a finite number of strings.
  - −4 for incorrectly accepting/rejecting an infinite number of strings.

- DFAs or NFAs that are more complex than necessary may be penalized. DFAs or NFAs that are *significantly* more complex than necessary may get no credit at all. On the other hand, *minimal* DFAs are *not* required for full credit, unless the problem explicitly asks for them.
- Half credit for describing an NFA when the problem asks for a DFA.

4 Context-Free Grammars

Give a context-free grammar for the two languages below, where the terminal set T is $\{0,1\}$. In order to get full credit you need to **briefly explain how your grammar works, and the role of each non-terminal**.

(a) (5pts) $L = \{0^i 1 0^j 1 0^k \mid i + k \geq 3j\}$

Solution: We give our grammar as follows:

$$\begin{aligned}
S &\rightarrow S_1 \mid 0S \mid S0 && \{0^i 1 0^j 1 0^k \mid i+k=j\}\\
S_1 &\rightarrow AB \mid 0A0B00 \mid 00A0B0 && \{0^i 1 0^j 1 0^k \mid i+k=j\}\\
A &\rightarrow 000A0 \mid 1 && \{0^{3a} 1 0^a \mid a \geq 0\}\\
B &\rightarrow 0B000 \mid 1 && \{0^b 1 0^{3b} \mid b \geq 0\}
\end{aligned}$$

To build this grammar, we start with a non-terminal S_1 to build all strings of the given form where $i+k$ is exactly equal to $3j$. Since $i+k$ must be zero mod 3, we have three cases to consider for the values of i and k mod 3: both are zero, i is 1 while k is 2, and i is 2 while k is 1. These are covered, respectively, by the three cases in the production rules for S_1. Finally, to create strings where $i+k$ can be equal or greater than $3j$, we can simply take any string built by S_1 and add any number of zeros to its beginning and end, which is what the production rules for S do. ■

Rubric: 5 points: standard CFG rubric, scaled.

(b) (5 pts) All strings that have an odd number of 0's and end in 01.

Solution: Note that this is in fact a regular language. Since we want an even number of zeros before the final two characters, one can write a regular expression for this language as $(1^*01^*0)^*1^*01$. We write the following grammar to reflect this regular expression.

$$\begin{aligned}
S &\rightarrow A01 && (1^*01^*0)^*1^*01\\
A &\rightarrow B0B0A \mid B && (1^*01^*0)^*1^*\\
B &\rightarrow 1B \mid \varepsilon && 1^*
\end{aligned}$$

The production rules for B ensure that it can build strings made up of any number of ones. Similarly, the production rules for A ensure that it can build strings made up of any number of copies of strings of the form $B0B0$ with an additional B at the very end. Since B builds strings of the form 1^*, the previous sentence means that that A builds strings of the form $(1^*01^*0)^*1^*$. Finally, S adds on the 01 we need at the end. ■

Solution: We give the following grammar:

$$
\begin{aligned}
S &\to E01 && \{w \mid \#(0,w) \text{ is odd and } w \text{ ends in } 01 \\
E &\to 1E \mid 0D \mid \epsilon && \{w \mid \#(0,w) \text{ is even}\} \\
D &\to 1D \mid 0E && \{w \mid \#(0,w) \text{ is odd}\}
\end{aligned}
$$

This grammar works by first defining two non-terminals E and D to capture strings with Even and odD numbers of zeros. These production rules work by considering cases for how many zeros the remainder of the string will have depending on if the first character is a 1 or a 0 (or as a base case in the even case, if the string is empty). Once we've built these, we know that in order for a string to end in 01 and have an odd number of zeros, it has to be of the form $w01$ where w is a string with an even number of zeros. ■

Rubric: 5 points: standard CFG rubric, scaled.

Rubric: Standard context-free grammar rubric 10 points =

- 6 points for the CFG:
  - + 2 for giving a syntactically correct CFG
  - + 4 if the CFG generates the target language
- 4 points for explaining why the CFG is correct.
  - ⋆ Must briefly describe the role of each non-terminal, including what language it generates.
  - ⋆ We explicitly do *not* want a formal proof of correctness, just a few sentences of explanation.

5 Fooling Sets and Non-Regularity

(10 pts) Prove that the language

$$L = \{w \in \{0,1\}^* \mid w \text{ is a palindrome and } w \text{ has at least three 0s}\}$$

is not regular by providing a fooling set for it and showing that every pair of strings in your fooling set can be distinguished.

Solution: We consider the infinite set $F = \{0^{3+i} \mid i \geq 0\}$. We claim that it is a fooling set for L. Let x, y be two distinct strings in F, where we can write $x = 0^{3+i}$ and $y = 0^{3+j}$ for some $i \neq j$. We consider the distinguishing suffix $z = 10^{3+i}$.

- Since $xz = 0^{3+i}10^{3+i}$ is a palindrome and has at least three zeros, $xz \in L$.
- Note that $yz = 0^{3+j}10^{3+i}$ is not a palindrome since $i \neq j$ and hence $yz \notin L$.

Since x, y were chosen arbitrarily, every pair of distinct strings in F are distinguishable with respect to L. F is infinite, so this means that L is not regular. ■

Rubric: Standard fooling set rubric 10 points =

- 4 points for the fooling set:
  - + 2 for explicitly describing the proposed fooling set F.
  - + 2 if the proposed set F is actually a fooling set for the target language.
  - − No credit for the proof if the proposed set is not a fooling set.
  - − No credit for the *problem* if the proposed set is finite (unless you give a fooling set of size at least n for every possible $n \in \mathbb{N}$).
- 6 points for the proof:
  - ∘ The proof must correctly consider *arbitrary* pairs of distinct strings $x, y \in F$.
    - − No credit for the proof unless both x and y are *always* in F.
    - − No credit for the proof unless x and y can be *any* pair of distinct strings in F.
  - + 2 for explicitly describing a suffix z that distinguishes x and y.
  - + 2 for proving either $xz \in L$ or $yz \in L$.
  - + 2 for proving either $yz \notin L$ or $xz \notin L$.

6 Language Transformation

(10 pts) Let $\Sigma = \{0, 1\}$. For a language $L \subseteq \Sigma^*$ we define an operation **deletepropermid** as follows:

$$\textbf{deletepropermid}(L) = \{uw \mid uvw \in L \text{ and } u, v, w \in \Sigma^*, |v| \geq 2\}.$$

Prove that if L is regular then **deletepropermid**(L) is also regular.

Solution: Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA accepting L. We will construct a NFA $N = (Q_N, \Sigma, \delta_N, s_N, A_N)$ that accepts **deletepropermid**(L) which will prove its regularity. The machine N is constructed intuitively by guessing where to insert v into its input, as well as guessing what state M would go to after reading v (conditioned on $|v| \geq 2$). Formally, we define N as follows:

- $Q_N = Q \times \{\text{before}, \text{after}\}$
- $s_N = (s, \text{before})$
- $A_N = A \times \{\text{after}\}$.
- $\delta_N((q, \text{before}), \epsilon) = \{(q', \text{after}) \mid \text{there is string } v,\ |v| \geq 2,\ \delta^*(q, v) = q'\}$
  $\delta_N((q, \text{before}), a) = \{(\delta(q, a), \text{before})\}$
  $\delta_N((q, \text{after}), \epsilon) = \emptyset$
  $\delta_N((q, \text{after}), a) = \{(\delta(q, a), \text{after})\}$

(We note that the definition of $\delta_N((q, \text{before}), \epsilon)$ can be a bit tricky to come up with. One could also make a definition that uses ϵ transitions to guess one character of v at a time; this would require adding a counter to the state to make sure we add at least 2 characters for v.) ■

Standard language transformation rubric (automata). For problems worth 10 points:

+ 2 for a formal, complete, and unambiguous description of the output automaton M', including the states, the start state(s), the accepting states, and the transition function, as functions of an *arbitrary* given DFA M. The description must state whether the output automaton is a DFA or an NFA.

+ 2 for a *brief* English explanation of the output automaton. We explicitly do *not* want a formal proof of correctness, or an English *transcription*, but a few sentences explaining how your machine works and justifying its correctness. What is the overall idea? What do the states represent? What is the transition function doing? Why these accepting states?

+ 6 for correctness

  + 1 for correct states — Almost always a product of the states Q of the given DFA with other side information; does the side information make sense? Could you build a transformation using *only* this side information?

  + 1 for correct start state(s)

  + 1 for correct accepting states

  + 3 for correct transition function

    — 1 for a single minor mistake

- Double-check correctness when the input language is $\emptyset$, or $\{\epsilon\}$, or $\mathbf{1}^*$, or Σ^*.
- Partial credit should be awarded relative to the *most similar correct solution*. For example, if a given incorrect solution can be fixed either by changing the accepting states or by changing the transition function, it should get partial credit for a good transition function.

Standard language transformation rubric (regular expressions). For problems worth 10 points:

+ 2 for a formal, complete, and unambiguous description of the output regular expression r', as a function of an *arbitrary* given regular expression r. This description can be recursive, in which case it needs to specify what to do in each base case and each inductive case.
+ 2 for a *brief* English explanation of the output expression. We explicitly do *not* want a formal proof of correctness, or an English *transcription*, but a few sentences explaining how your expression works and justifying its correctness. What is the overall idea? Why these particular modifications? If your construction is recursive, you should briefly justify each base case and each recursive case.
+ 6 for correctness
  - For recursive constructions: 1 point per case ($\emptyset$, ϵ, character, $r_1 + r_2$, $r_1 r_2$, $(r_1)^*$). Half credit on a case if it contains a single minor mistake.
  - For direct constructions:
    + 3 points for accepting all strings in the target language.
      - No credit if this is only because the expression always accepts every string.
      - Half credit if this is true up to a minor mistake.
    + 3 points for accepting *only* strings in the target language.
      - No credit if this is only because the expression always rejects every string.
      - Half credit if this is true up to a minor mistake.

- Double-check correctness when the input language is $\emptyset$, or $\{\epsilon\}$, or $\mathbf{1}^*$, or Σ^*.
- Partial credit should be awarded relative to the *most similar correct solution*.