

# CS 374A Midterm 1 Review

ACM @ UIUC

Feburary 21st, 2026



# intro (end of the world)

- We are here as members of the ACM academic committee, rather than in our capacity as CAs. As a result, we'll only focus on **content** rather than exam-specific questions.
- Exam-specific questions are better suited for the class Edstem, Discord, office hours, or homework party.
- This review session is being recorded. Recordings and slides will be distributed on EdStem after the end.
- **Agenda:** We'll review all topics likely to be covered, then go through practice problems, then review individual topics by request.
- Please let us know if we're going too fast/slow, not speaking loud enough/speaking too loud, etc.
- If you have a question anytime during the review session, please ask! Someone else almost surely has a similar question.
- We'll provide a feedback form at the end of the session.

# Content Review

breathin'

# Induction

Template

Let  $x$  be an *arbitrary* string/integer/etc.

**Inductive Hypothesis:** Assume for all  $k$  s.t.  $k$  is shorter/smaller/etc. than  $x$  that  $P(k)$  (what we're trying to prove) holds.

**Base Case:** If  $x = 0, \epsilon$ , whatever your base case is, then . . . , so  $P(x)$  holds.

**Inductive Step:** If  $x \neq 0, \epsilon$ , whatever your base case is, then . . . , so by the inductive hypothesis, . . . , so  $P(x)$  holds.

Thus, by the principle of induction,  $P(x)$  holds.

# Induction

## Template

Let  $x$  be an *arbitrary* string/integer/etc.

**Inductive Hypothesis:** Assume for all  $k$  s.t.  $k$  is shorter/smaller/etc. than  $x$  that  $P(k)$  (what we're trying to prove) holds.

**Base Case:** If  $x = 0, \epsilon$ , whatever your base case is, then . . . , so  $P(x)$  holds.

**Inductive Step:** If  $x \neq 0, \epsilon$ , whatever your base case is, then . . . , so by the inductive hypothesis, . . . , so  $P(x)$  holds.

Thus, by the principle of induction,  $P(x)$  holds.

Some tips:

- Always use strong induction.
- Write out your IH, base case, and inductive step out explicitly.
- Think about what you would like to know about your smaller/shorter numbers/strings.

# Regular Languages/Expressions

- Built inductively on 3 operations:
  - + is the union operator.  $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
  - \* is the Kleene star.  $L(r_1^*) = L(r_1)^*$
  - () are used to group expressions
  - (implicit) concatenation operator:  $L(r_1r_2) = \{xy : x \in L_1, y \in L_2\}$

# Regular Languages/Expressions

- Built inductively on 3 operations:
  - + is the union operator.  $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
  - \* is the Kleene star.  $L(r_1^*) = L(r_1)^*$
  - () are used to group expressions
  - (implicit) concatenation operator:  $L(r_1r_2) = \{xy : x \in L_1, y \in L_2\}$
- Closed under Union ( $\cup$ ), intersection ( $\cap$ ), concatenation ( $\cdot$ ), kleene star ( $*$ ), complement ( $^C$ ), set difference ( $\setminus$ ), and reverse ( $^R$ )
  - ... but only finitely many applications of these operations

# Regular Languages/Expressions

- Built inductively on 3 operations:
  - + is the union operator.  $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
  - \* is the Kleene star.  $L(r_1^*) = L(r_1)^*$
  - () are used to group expressions
  - (implicit) concatenation operator:  $L(r_1r_2) = \{xy : x \in L_1, y \in L_2\}$
- Closed under Union ( $\cup$ ), intersection ( $\cap$ ), concatenation ( $\cdot$ ), kleene star (\*), complement ( $^C$ ), set difference ( $\setminus$ ), and reverse ( $^R$ )
  - ... but only finitely many applications of these operations
- If trying to guess whether or not a language is regular, think about memory. DFAs only get finite memory!
  - You don't get to look back indefinitely.
  - If your language requires you to track a number or string indefinitely, it is not regular!

# Regular Languages/Expressions

- Built inductively on 3 operations:
  - + is the union operator.  $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
  - \* is the Kleene star.  $L(r_1^*) = L(r_1)^*$
  - () are used to group expressions
  - (implicit) concatenation operator:  $L(r_1r_2) = \{xy : x \in L_1, y \in L_2\}$
- Closed under Union ( $\cup$ ), intersection ( $\cap$ ), concatenation ( $\cdot$ ), kleene star (\*), complement ( $^C$ ), set difference ( $\setminus$ ), and reverse ( $^R$ )
  - ... but only finitely many applications of these operations
- If trying to guess whether or not a language is regular, think about memory. DFAs only get finite memory!
  - You don't get to look back indefinitely.
  - If your language requires you to track a number or string indefinitely, it is not regular!
- **Regex Design Tips:**
  - What strings are in your language? Which ones aren't? Note edge cases (specifically check  $\epsilon$ ).
  - Look for patterns and substrings that you definitely need to include or repeat.

# DFAs

- DFA  $M = (Q, A, \Sigma, s, \delta)$ 
  - $Q$  - FINITE set of states
  - $A \subseteq Q$  - accepting states
  - $\Sigma$  - input alphabet, usually  $\{0, 1\}$
  - $s \in Q$  - start state
  - $\delta : Q \times \Sigma \rightarrow Q$  - transition function

# DFAs

- DFA  $M = (Q, A, \Sigma, s, \delta)$ 
  - $Q$  - FINITE set of states
  - $A \subseteq Q$  - accepting states
  - $\Sigma$  - input alphabet, usually  $\{0, 1\}$
  - $s \in Q$  - start state
  - $\delta : Q \times \Sigma \rightarrow Q$  - transition function
- **Tips for Creating DFAs:**
  - Define your states exactly! What does it mean to be at each state?
  - Based on these definitions, when should you accept? Define  $A$  accordingly.
  - What state represents  $\epsilon$ ? Make that the start state. Make sure that if  $L$  accepts  $\epsilon$ , you accept your start state.
  - How does reading in a 0 or 1 change each state? Define  $\delta$  accordingly.

# DFAs

- DFA  $M = (Q, A, \Sigma, s, \delta)$ 
  - $Q$  - FINITE set of states
  - $A \subseteq Q$  - accepting states
  - $\Sigma$  - input alphabet, usually  $\{0, 1\}$
  - $s \in Q$  - start state
  - $\delta : Q \times \Sigma \rightarrow Q$  - transition function
- **Tips for Creating DFAs:**
  - Define your states exactly! What does it mean to be at each state?
  - Based on these definitions, when should you accept? Define  $A$  accordingly.
  - What state represents  $\epsilon$ ? Make that the start state. Make sure that if  $L$  accepts  $\epsilon$ , you accept your start state.
  - How does reading in a 0 or 1 change each state? Define  $\delta$  accordingly.
- Notes:
  - To go from  $L$  to its complement, just switch the accepting and non-accepting states.
  - Every DFA is automatically an NFA.
  - Every regular language can be represented by a DFA. Every DFA represents a regular language.

# Product Constructions

- Say I can build DFA  $M_1$  keeping track of one property and DFA  $M_2$  keeping track of another property. What if I want a DFA  $M$  that keeps track of both properties?
- You can combine the information of both DFAs into one product DFA.
- The accept states in your new DFA define whether I only accept strings with both properties or strings with one or the other or some other logical operation.

# Product Constructions

- Say I can build DFA  $M_1$  keeping track of one property and DFA  $M_2$  keeping track of another property. What if I want a DFA  $M$  that keeps track of both properties?
- You can combine the information of both DFAs into one product DFA.
- The accept states in your new DFA define whether I only accept strings with both properties or strings with one or the other or some other logical operation.

## Template

1. Define (drawing/formal) your first DFA  $M_1 = (Q_1, A_1, \Sigma, s_1, \delta_1)$ .
2. Define (drawing/formal) your second DFA  $M_2 = (Q_2, A_2, \Sigma, s_2, \delta_2)$ .
3. Define the product DFA  $M = (Q, A, \Sigma, s, \delta)$  as follows:
  - ▶  $Q = Q_1 \times Q_2 = \{(q_1, q_2) | q_1 \in Q_1, q_2 \in Q_2\}$  - each state is a tuple
  - ▶  $s = (s_1, s_2)$  - just the tuple containing both start states
  - ▶  $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$  - apply the first transition function to the first state and the second transition function to the second state
  - ▶  $A = \{(q_1, q_2) | q_1 \in A_1 \text{ and/or/etc. } q_2 \in A_2\}$  - check if the first state is accepted by the first DFA, check if the second state is accepted by the second DFA, and accept based on the problem statement

# NFAs

- NFA  $N = (Q, A, \Sigma, s, \delta)$ 
  - $Q$  - FINITE set of states
  - $A \subseteq Q$  - accepting states
  - $\Sigma$  - input alphabet, usually  $\{0, 1\}$
  - $s \in Q$  or  $s \subseteq Q$  - start state(s) — specify if using multiple
  - $\delta : Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$  - transition function

# NFAs

- NFA  $N = (Q, A, \Sigma, s, \delta)$ 
  - $Q$  - FINITE set of states
  - $A \subseteq Q$  - accepting states
  - $\Sigma$  - input alphabet, usually  $\{0, 1\}$
  - $s \in Q$  or  $s \subseteq Q$  - start state(s) — specify if using multiple
  - $\delta : Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$  - transition function
- Changes
  - You can now have multiple start states!
  - We've added epsilon transitions.
  - We can transition to multiple different states.

# NFAs

- NFA  $N = (Q, A, \Sigma, s, \delta)$ 
  - $Q$  - FINITE set of states
  - $A \subseteq Q$  - accepting states
  - $\Sigma$  - input alphabet, usually  $\{0, 1\}$
  - $s \in Q$  or  $s \subseteq Q$  - start state(s) — specify if using multiple
  - $\delta : Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$  - transition function
- Changes
  - You can now have multiple start states!
  - We've added epsilon transitions.
  - We can transition to multiple different states.
- **Tips for Creating NFAs:**
  - Make sure that your transition function consistently leads to a SET of states (whether that set is empty, has 1 state, or multiple states).
  - Use epsilon transitions to jump from one state to another without reading anything (usually when you want to be able to transition from one phase to another).

# NFAs

- NFA  $N = (Q, A, \Sigma, s, \delta)$ 
  - $Q$  - FINITE set of states
  - $A \subseteq Q$  - accepting states
  - $\Sigma$  - input alphabet, usually  $\{0, 1\}$
  - $s \in Q$  or  $s \subseteq Q$  - start state(s) — specify if using multiple
  - $\delta : Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$  - transition function
- Changes
  - You can now have multiple start states!
  - We've added epsilon transitions.
  - We can transition to multiple different states.
- **Tips for Creating NFAs:**
  - Make sure that your transition function consistently leads to a SET of states (whether that set is empty, has 1 state, or multiple states).
  - Use epsilon transitions to jump from one state to another without reading anything (usually when you want to be able to transition from one phase to another).
- Notes
  - Every NFA can be converted to a DFA through power set construction. The DFA states are the power set of the NFA states.

# Fooling Sets

- DFAs only care about which state you're in, and not how you got there
  - If two strings result in the same DFA state, any additional suffix added to both will also result in both strings being in the same state.

# Fooling Sets

- DFAs only care about which state you're in, and not how you got there
  - If two strings result in the same DFA state, any additional suffix added to both will also result in both strings being in the same state.
  - Thus, if we have  $x, y$ , and we know that there exists a **distinguishing suffix**  $z$  s.t.  $xz \in L, yz \notin L$ , then  $x, y$  must be in *different* states for *any* DFA that accepts  $L$

# Fooling Sets

- DFAs only care about which state you're in, and not how you got there
  - If two strings result in the same DFA state, any additional suffix added to both will also result in both strings being in the same state.
  - Thus, if we have  $x, y$ , and we know that there exists a **distinguishing suffix**  $z$  s.t.  $xz \in L, yz \notin L$ , then  $x, y$  must be in *different* states for *any* DFA that accepts  $L$
- A **fooling set** is a set of strings where there exists a distinguishing suffix between every pair of strings
- Myhill-Nerode: min DFA size = max fooling set size
  - Thus, languages with infinite fooling sets are *not* regular

# Fooling Sets

- DFAs only care about which state you're in, and not how you got there
  - If two strings result in the same DFA state, any additional suffix added to both will also result in both strings being in the same state.
  - Thus, if we have  $x, y$ , and we know that there exists a **distinguishing suffix**  $z$  s.t.  $xz \in L, yz \notin L$ , then  $x, y$  must be in *different* states for *any* DFA that accepts  $L$
- A **fooling set** is a set of strings where there exists a distinguishing suffix between every pair of strings
- Myhill-Nerode: min DFA size = max fooling set size
  - Thus, languages with infinite fooling sets are *not* regular
- **Tips and Tricks**
  - If  $L$  needs to keep track of a value with no bound, create a fooling set around the part you count up.
  - If you're using strings of the form  $1^k, 0^p$ , etc. when sampling elements of your fooling set  $a^i, a^j$ , you may assume WLOG that  $i < j$ .

# Prove $\{0^n 1^n \mid n \geq 0\}$ is irregular.

Template adapted from *Fall 2025 Homework 3 solutions*.

Let  $F = \left\{ \boxed{\text{ }} \right\}$ , which is infinite.

Let  $x, y \in F$  with  $x \neq y$ . Thus,  $x = \boxed{\text{ }}$  and  $y = \boxed{\text{ }}$ , where  $\boxed{\text{ }}$ .

Let  $z = \boxed{\text{ }}$ .

- Then,  $xz \in L$  because  $\boxed{\text{ }}$ .

- Then,  $yz \notin L$  because  $\boxed{\text{ }}$ .

Thus,  $z$  is a distinguishing suffix for  $x$  and  $y$ , so  $F$  is a fooling set for  $L$ .

Since  $F$  is infinite,  $L$  is not a regular language.

**Note:** We can also pick  $z$  so that  $xz \notin L$  and  $yz \in L$ . Just do whichever is easier.

# Language Transformations

- We have a language  $L$  we know is regular.
- We have a function  $f$  from strings to strings:  $f(w) = x$ .
- We define a transformed language in one of the following formats:
  1.  $L' = \{f(w) | w \in L\}$
  2.  $L' = \{w | f(w) \in L\}$
- We want to show that  $L'$  is regular by making a DFA/NFA that accepts  $L'$ .

# Language Transformations

- We have a language  $L$  we know is regular.
- We have a function  $f$  from strings to strings:  $f(w) = x$ .
- We define a transformed language in one of the following formats:
  1.  $L' = \{f(w) | w \in L\}$
  2.  $L' = \{w | f(w) \in L\}$
- We want to show that  $L'$  is regular by making a DFA/NFA that accepts  $L'$ .

## Template

- Since  $L$  is regular, there is a DFA  $M = (Q, A, \Sigma, s, \delta)$  that accepts it.
- Now, we create a DFA/NFA  $M' = (Q', A', \Sigma, s', \delta')$  that accepts  $L'$ 
  1.  $L' = \{f(w) : w \in L\}$  -  $M'$  reads  $f(w)$ ; we need to check if the original string  $w$  is accepted by  $M$
  2.  $L' = \{w : f(w) \in L\}$  -  $M'$  reads  $w$ ; we need to check if  $f(w)$  is accepted by  $M$
- Define the states  $Q'$  by thinking about what information you need to interpret the next letter you read - usually  $Q \times \{\text{Information needed to convert } x \text{ to } w \text{ or } w \text{ to } x\}$
- Define the transition function  $\delta$  so it passes  $w$  or  $x$  back to the original DFA - potentially using nondeterminism or epsilon transitions

# Context-Free Languages/Grammars

- Nonterminals and productions: e.g.,

$$S \rightarrow G \mid L \mid 0S1$$

$$G \rightarrow 0G \mid 0$$

$$L \rightarrow L1 \mid 1$$

- Closed under union, concatenation, and Kleene star (Every regular language is a CFL)
- **Not closed** under complement or intersection

# Context-Free Languages/Grammars

- Nonterminals and productions: e.g.,

$$S \rightarrow G \mid L \mid 0S1$$

$$G \rightarrow 0G \mid 0$$

$$L \rightarrow L1 \mid 1$$

- Closed under union, concatenation, and Kleene star (Every regular language is a CFL)
- **Not closed** under complement or intersection
- Tips for constructing CFGs:
  - Think recursively! Given a language, how can strings in the language be broken down?
  - Match outer structure first
  - For languages like  $0^n 1^n$ , generate matched symbols together
  - Split into cases as needed (in the example above, we split into greater and less than cases)

# Practice Problems

main thing

# Short Answer T/F (1)

For each of the following, determine if the statement is **true** or **false**, and give a one-sentence explanation of your answer. (These are intentionally tricky)

- (a) For all languages  $L$ , if  $L$  is irregular, then  $L$  has a finite fooling set.
- (b) If  $M$  is a minimal DFA that decides a language  $L$ , and running  $M$  on strings  $x$  and  $y$  result in states  $q$  and  $q'$ , respectively, where  $q \neq q'$ , then there exists a distinguishing suffix between  $x$  and  $y$  in  $L$ .
- (c) The language  $L = \{0^i 1^j 0^k : i = j \text{ and } k \equiv i \pmod{374}\}$  is context-free.
- (d) For context-free languages  $L_1, L_2$ , the language  $L = (L_1^* L_2) \cup (L_1 L_2^*)$  is context-free.
- (e) (**Fall 2024**) For **all** regular languages  $L_R$  and context free languages  $L_C$ ,  $L_R \setminus L_C$  is context free.
- (f) Suppose  $L_1, L_2, \dots$  is an infinite sequence of regular languages, where  $L_i \supseteq L_{i+1}$  for all  $i \geq 1$ . Then,  $\bigcup_{i=1}^{\infty} L_i$  is regular.

# Short Answer T/F (2)

For each of the following, determine if the statement is **true** or **false**, and give a one-sentence explanation of your answer. (These are intentionally tricky)

- (g) The language  $\{xx^Ry : x, y \in \{0, 1\}^*\}$  is regular.
- (h) If  $L$  is regular, then  $\text{SELFOLD}(L) := \{a_1a_n a_2 a_{n-1} \cdots a_{\lceil \frac{n}{2} \rceil} : a_1a_2 \cdots a_n \in L\}$  is regular.
- (i) Consider the language  $L = \{1^x 2^y 3^z : y = x + z\}$ . There exists a distinguishing suffix between the strings 1112222223 and 2223.
- (j) Let  $M_1, M_2$  be arbitrary NFAs with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then  $L(M_1) \cap L(M_2) = \emptyset$ .
- (k) Consider an infinite sequence of regular languages  $L_1, L_2, \dots$  s.t.  $L_i \subseteq L_{i+1}$  for all  $i \geq 1$ . The language  $\cup_{i=1}^{\infty} L_i$  is context free.

# Regular or Not?

For each of the following languages, either *prove* that the language is regular, or *prove* that it is not regular (**Hint:** exactly two of the four languages are regular). For all questions,  $\Sigma = \{0, 1\}$ .

- $\{1xyx \mid x, y \in \Sigma^*\}$
- $\{x1xy \mid x, y \in \Sigma^*\}$
- $\{w \in \Sigma^* : |w| \geq 374 \text{ and last 374 characters of } w \text{ have equal number of 0s and 1s}\}$
- **(Fall 2021 Conflict)**  $\{0^p 1^q 0^r \mid r = p + q\}$

# Language Transformations ( $\Sigma = \{0, 1\}$ )

(Spring 2025) For a language  $L \subseteq \Sigma^*$ , we define operation  $\text{BYE}$ :

$$\text{BYE}(L) := \{uw : uvw \in L \text{ and } u, v, w \in \Sigma^*, |v| \geq 2\}$$

For example, if  $L = \{0110, 01, 101\}$ , then

$$\text{BYE}(L) = \{\epsilon, 0, 1, 00, 01, 10\}$$

Intuitively,  $\text{BYE}(L)$  is the set of all strings obtained by deleting a contiguous substring of length  $\geq 2$  from some string in  $L$ . **Prove** that if  $L$  is regular, then  $\text{BYE}(L)$  is also regular.

# Language Transformations ( $\Sigma = \{0, 1\}$ )

# Language Transformations ( $\Sigma = \{0, 1\}$ )

For a language  $L \subseteq \{0, 1\}^*$ , define:

$$\text{WARM}(L) := \{x1^k y : xy \in L, k \geq 1\}$$

That is,  $\text{WARM}(L)$  inserts exactly one substring of 1s into each string from  $L$ .

For example, Let  $L = \{0, 01\}$ .

$$\text{WARM}(L) = \{10, 011, 0111, 110, 0111, 1011, \dots\}$$

**Prove** that if  $L$  is regular, then  $\text{WARM}(L)$  is also regular.

# Language Transformations ( $\Sigma = \{0, 1\}$ )

# Language Transformations ( $\Sigma = \{0, 1\}$ )

(Fall 2024 HW) For two strings  $s, t \in \Sigma^*$ , we define  $\text{YesAnd}$  to be the set of strings:

$$\text{YesAnd}(s, t) = \begin{cases} \{s\} & \text{if } t = \varepsilon \\ \{t\} & \text{if } s = \varepsilon \\ \{a \cdot w \mid w \in \text{YesAnd}(s', t)\} \cup \{b \cdot w \mid w \in \text{YesAnd}(s, t')\} & \text{if } s = as', t = bt' \end{cases}$$

For example, let  $s = 00$ , and  $t = 111$ . Then:

$$\text{YesAnd}(00, 111) = \{00111, 01011, 01101, 01110, 10011, 10101, 10110, 11001, 11010, 11100\}$$

Intuitively,  $\text{YesAnd}(s, t)$  is the set of all strings formed by interleaving the characters of  $s$  and  $t$ , preserving the order within each string.

For two regular languages  $L_1, L_2 \subseteq \Sigma^*$ , **prove** that

$$\text{SIDEToSIDE}(L_1, L_2) := \{w \in \text{YesAnd}(s, t) : s \in L_1, t \in L_2\}$$

is also regular.

# Language Transformations ( $\Sigma = \{0, 1\}$ )

# DFAs/NFAs/Regexes

With  $\Sigma = \{0, 1\}$ ,

- (a) Write a regex for strings with no even-length runs.

# DFAs/NFAs/Regexes

With  $\Sigma = \{0, 1\}$ ,

- (a) Write a regex for strings with no even-length runs.
- (b) Write a DFA *and* regex for  $\{w \in \Sigma^* \mid \#_0(w) \geq 2 \text{ or } \#_1(w) \geq 2\}$ .

# DFAs/NFAs/Regexes

With  $\Sigma = \{0, 1\}$ ,

- (a) Write a regex for strings with no even-length runs.
- (b) Write a DFA *and* regex for  $\{w \in \Sigma^* \mid \#_0(w) \geq 2 \text{ or } \#_1(w) \geq 2\}$ .
- (c) All strings that do not contain 010 as a substring.

# CFGs

Show that the following languages are context-free by providing grammars.

- (a)  $\{ww^R : w \in \{0, 1\}^*\text{ and }|ww^R| \equiv 1 \pmod{3}\}$

# CFGs

Show that the following languages are context-free by providing grammars.

- (a)  $\{ww^R : w \in \{0, 1\}^* \text{ and } |ww^R| \equiv 1 \pmod{3}\}$
- (b)  $\{x\$y : x, y \in \{0, 1\}^* \text{ and } \#_0(x) = \#_1(y)\}$   $(\Sigma = \{0, 1, \$\})$

# CFGs

Show that the following languages are context-free by providing grammars.

- (a)  $\{ww^R : w \in \{0, 1\}^*\text{ and }|ww^R| \equiv 1 \pmod{3}\}$
- (b)  $\{x\$y : x, y \in \{0, 1\}^*\text{ and }#_0(x) = #_1(y)\}$  ( $\Sigma = \{0, 1, \$\}$ )
- (c)  $\{0^x 1^y 2^z : x - y = z\}$

# Feedback (thank u, next)



[go.acm.illinois.edu/374A\\_feedback](http://go.acm.illinois.edu/374A_feedback)