CS 374A Midterm 1 Review

ACM @ UIUC

September 27, 2025

Disclaimers and Logistics

- **Disclaimer:** We are all current or past CAs, but we have NOT seen the exam. We have no idea what the questions are. However, we've taken the course and reviewed Jeff's previous exams, so we have a good idea of what it'll look like.
- 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 a practice exam, then review individual topics by request.
  - Questions are designed to be written in the same style as Jeff's previous exams but to be *slightly* harder, so don't worry if you don't get everything right away!
- 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.

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 $\ldots$, so $P(x)$ holds.

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

Inductive Step: If $x \neq 0, \epsilon,$ whatever your base case is, then $\ldots$, so by the inductive hypothesis, $\ldots$, 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: $L(r_1 r_2) = \{xy : x \in L_1, y \in L_2\}$

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: $L(r_1 r_2) = \{xy : x \in L_1, y \in L_2\}$
- 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

$$\bigcup_{i=1}^{\infty} \{0^i 1^i\} = \{0^n 1^n : n \geq 1\}$$

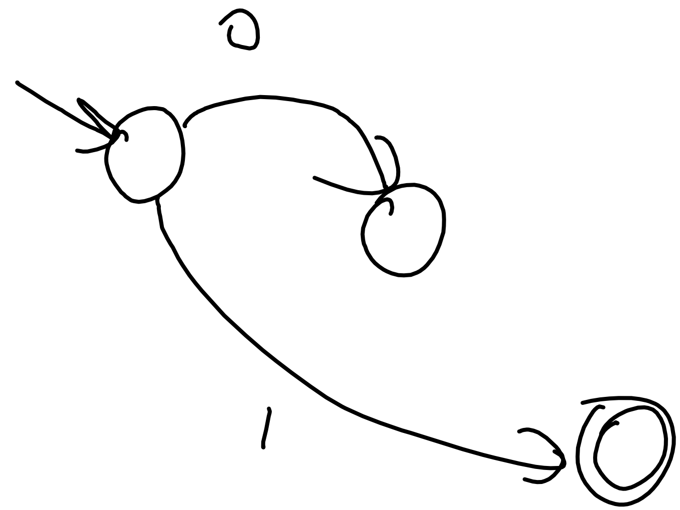
Regular Languages/Expressions

- 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: $L(r_1 r_2) = \{xy : x \in L_1, y \in L_2\}$
- 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: $L(r_1 r_2) = \{xy : x \in L_1, y \in L_2\}$
- Closed under Union ($\cup$), intersection ($\cap$), concatenation ($\cdot$), kleene star (*), complement (^{C}), set difference ($\backslash$), 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 ϵ).
  - 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
  - Σ - input alphabet, usually $\{0, 1\}$
  - $s \in Q$ - start state
  - $\delta : Q \times \Sigma \to Q$ - transition function

DFAs

- DFA $M = (Q, A, \Sigma, s, \delta)$
  - Q - FINITE set of states
  - $A \subseteq Q$ - accepting states
  - Σ - input alphabet, usually $\{0, 1\}$
  - $s \in Q$ - start state
  - $\delta : Q \times \Sigma \to Q$ - 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 ϵ? Make that the start state. Make sure that if L accepts ϵ, you accept your start state.
  - How does reading in a 0 or 1 change each state? Define δ accordingly.

DFAs

- DFA $M = (Q, A, \Sigma, s, \delta)$
  - Q - FINITE set of states
  - $A \subseteq Q$ - accepting states
  - Σ - input alphabet, usually $\{0, 1\}$
  - $s \in Q$ - start state
  - $\delta : Q \times \Sigma \to Q$ - 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 ϵ? Make that the start state. Make sure that if L accepts ϵ, you accept your start state.
  - How does reading in a 0 or 1 change each state? Define δ 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
    - $A = \{(q_1, q_2) | q_1 \in A_1$ **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
  - Σ - input alphabet, usually $\{0, 1\}$
  - $s \subseteq Q$ - start state(s)
  - $\delta : Q \times (\Sigma \cup \epsilon) \rightarrow \underline{2^Q}$ - transition function

$$\delta(q, 1) = q_2$$

$$\delta(q, 1) = \{q_1, q_2, q_3\}$$

NFAs

- NFA $N = (Q, A, \Sigma, s, \delta)$
  - Q - FINITE set of states
  - $A \subseteq Q$ - accepting states
  - Σ - input alphabet, usually $\{0, 1\}$
  - $s \subseteq Q$ - start state(s)
  - $\delta : Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$ - 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
  - Σ - input alphabet, usually $\{0, 1\}$
  - $s \subseteq Q$ - start state(s)
  - $\delta : Q \times (\Sigma \cup \epsilon) \to 2^Q$ - 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
  - Σ - input alphabet, usually $\{0, 1\}$
  - $s \subseteq Q$ - start state(s)
  - $\delta : Q \times (\Sigma \cup \epsilon) \to 2^Q$ - 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 $i < j$.

nonregular language $\leftrightarrow$ inf. fooling set

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

Template adapted from *Fall 2025 Homework 3 solutions*.

Let $F = \left\{ \boxed{0^*} \right\}$, which is infinite.

Let $x, y \in F$ with $x \neq y$. Thus, $x = \boxed{0^i}$ and $y = \boxed{0^j}$, where $\boxed{i \neq j}$.

Let $z = \boxed{1^i}$. 1^j

- Then, $xz \in L$ because $\boxed{0^i 1^i \in L}$.
- Then, $yz \notin L$ because $\boxed{0^j 1^i \notin L \quad (\text{b.c. } i \neq j)}$.

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 in the results of $f(w) = x$, so we need to undo f to find w from x and then push w through the original DFA M to check acceptance
  2. $L' = \{w | f(w) \in L\}$ - M' reads in strings w, so we need to apply f to w to find $f(w) = x$ and then push x through the original DFA M to check acceptance
- 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 δ so it passes w or x back to the original DFA - potentially using nondeterminism or epsilon transitions

Context-Free Languages/Grammars

- Formally, a context-free grammar is defined by
  - V - nonterminals/variables
  - T - terminals/symbols (aka Σ)
  - $S \in V$ - start variable
  - P - set of production rules $A \to \alpha$ with $A \in V$ and $\alpha \in \underline{(V \cup T)^*}$

Context-Free Languages/Grammars

- Formally, a context-free grammar is defined by
  - V - nonterminals/variables
  - T - terminals/symbols
  - $S \in V$ - start variable
  - P - set of production rules $A \to \alpha$ with $A \in V$ and $\alpha \in (V \cup T)^*$
- Context-free grammars often build from the outside in, peeling away layer by layer

$A \to 0B1$

$S \to S_1 S_2$

Context-Free Languages/Grammars

- Formally, a context-free grammar is defined by
  - V - nonterminals/variables
  - T - terminals/symbols
  - $S \in V$ - start variable
  - P - set of production rules $A \to \alpha$ with $A \in V$ and $\alpha \in (V \cup T)^*$
- Context-free grammars often build from the outside in, peeling away layer by layer
- CFLs are only closed under union, kleene star, and concatenation. CFLs are *not* closed under intersection or complement.

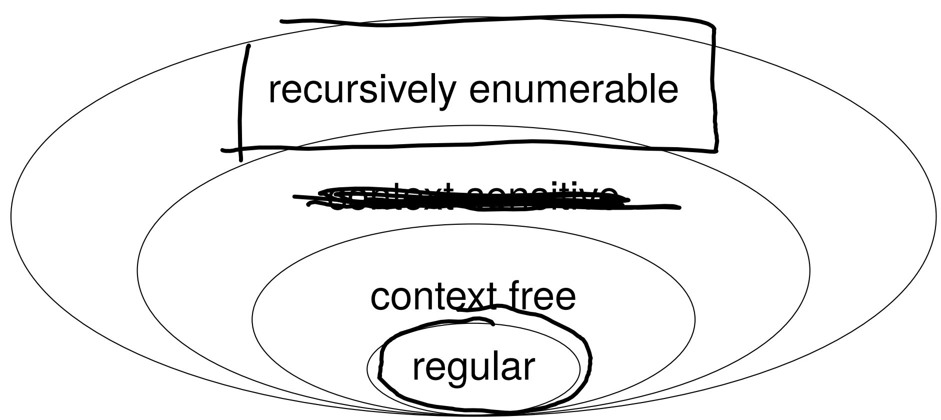
$(0+1)^* \setminus L(G)$ ← not nec. CF!

Context-Free Languages/Grammars

$\rightarrow \{a^m b^n c^n\}$

$\rightarrow \{a^n b^n c^m\}$

- Formally, a context-free grammar is defined by
  - V - nonterminals/variables
  - T - terminals/symbols
  - $S \in V$ - start variable
  - P - set of production rules $A \to \alpha$ with $A \in V$ and $\alpha \in (V \cup T)^*$
- Context-free grammars often build from the outside in, peeling away layer by layer
- CFLs are only closed under union, kleene star, and concatenation. CFLs are *not* closed under intersection or complement.



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.

Every language has a fin. fooling set! $\emptyset$ $\{1\}$

(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.

True.

(c) The language $L = \{\boxed{0^i 1^j} 0^k : i = j \text{ and } \boxed{k \equiv i} \pmod{374}\}$ is context-free.

True. 374 cases for congruence.

$i=j$, $i,j,k \equiv 1 \bmod 374$

(d) For context-free languages L_1, L_2, the language $L = (L_1^* L_2) \cup (L_1 L_2^*)$ is context-free.

True. Closed under $\cup$, $\circ$, $*$

(e) **(Fall 2024)** For **all** regular languages L_R and context free languages L_C, $L_R \setminus L_C$ is context free.

False.

(f) Suppose $L_1, L_2, \ldots$ 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 (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.

True

(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.

True

CF ∩ Reg = CF

(c) The language $L = \{0^i 1^j 0^k : i = j \text{ and } k \equiv i \pmod{374}\}$ is context-free.

True

(d) For context-free languages L_1, L_2, the language $L = (L_1^* L_2) \cup (L_1 L_2^*)$ is context-free.

True

$L_C \setminus L_R = \underbrace{L_C} \cap \underbrace{((0+1)^* \setminus L_R)} = CF$

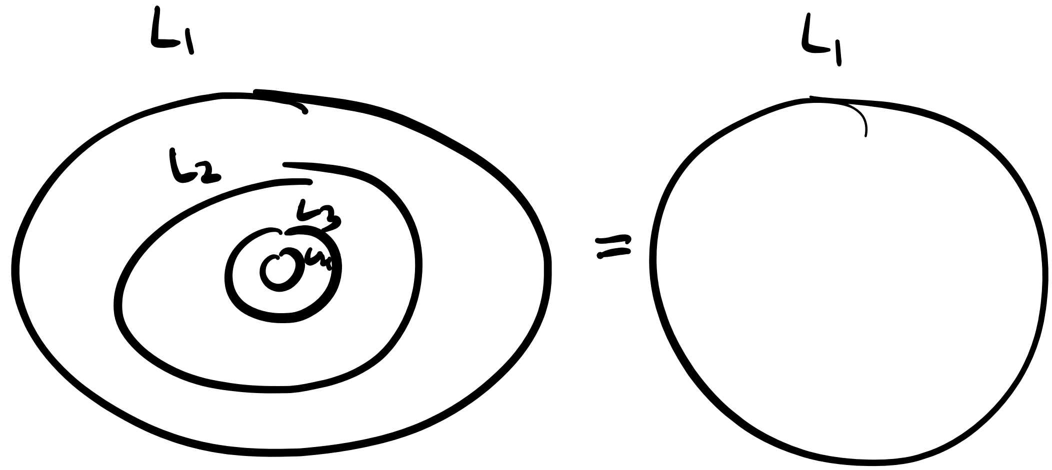
(e) **(Fall 2024)** For **all** regular languages L_R and context free languages L_C, $L_R \setminus L_C$ is context free.

False. CFL aren't closed under compl.: $(0+1)^* \setminus L_C$

(f) Suppose $L_1, L_2, \ldots$ 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.

True: $\bigcup L_i = L_1$ False w/o $L_i \supseteq L_{i+1}$

$L_i = \{0^i 1^i\}$ $\bigcup_{i=1}^{\infty} L_i = \{0^i 1^i \mid i \geq 1\}$



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 $\{xx^R y : x, y \in \{0,1\}^*\}$ is regular.

$(0+1)^*$ true

(h) If L is regular, then $\textsc{SelfFold}(L) = \{a_1 a_n a_2 a_{n-1} \cdots a_{\lceil \frac{n}{2} \rceil} : a_1 a_2 \cdots a_n \in L\}$ is regular.

true

(i) Consider the language $L = \{1^x 2^y 3^z : y = x + z\}$. There exists a distinguishing suffix between the strings $\underbrace{111}_{3}\underbrace{222222}_{6}\underbrace{3}_{1}$ and $\underbrace{}_{0}\underbrace{222}_{3}\underbrace{3}_{1}$.

False

(j) Let M_1, M_2 be arbitrary NFAs with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then $L(M_1) \cap L(M_2) = \varnothing$.

False

(k) Consider an infinite sequence of regular languages $L_1, L_2, \ldots$ s.t. $L_{i-1} \subseteq L_i$. The language $\cup_{i=1}^{\infty} L_i$ is context free.

False

$\{0^i 1^i 2^i\}$

$\{012\}$

$\{012, 001122\}$

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

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

$\{1xyx \mid x, y \in \Sigma^*\}$

$x = \epsilon$

= set of all strings that start with 1

$1(0+1)^*$

$F = \{0^n 1\}$

$x = 0^i 1$

$y = 0^j 1$

$i < j$ WLOG

$z = 0^i$

$0^i 1 0^i \in L$

$0^j 1 0^i \notin L$

$$\{w \in \Sigma^* : |w| \geq 374 \text{ and } \underbrace{\text{last 374 chars. of } w \text{ have \#0s = \#1s}}_{L}\}$$

$$w$$

$$\underbrace{\quad}_{(0+1)^*} \underbrace{\quad}_{\text{last 374}}$$

$$(0+1)^* \cdot \overbrace{\text{finite}}^{2^{374}}$$

$$\{0^p 1^q 0^r \mid r = p+q\}$$

$1^n 0^n$ $\quad r = q$ $\quad F = \{1^n\}$

$1^i \quad 1^j$

0^i

$1^i 0^i \in L$

$1^j 0^i \notin L$

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

(Spring 2025) For a language $L \subseteq \Sigma^*$, we define operation DeleteProperMid:

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

Prove that if L is regular, then DeleteProperMid(L) is also regular.

Given a DFA $M = (Q, s, A, \delta)$ for L, we define an NFA $M' = (Q', s', A', \delta')$ for DPM(L)

u

guess

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

(Fall 2024) For a string w, let swap(w) denote the set of all strings formed by selecting any number of disjoint pairs of consecutive bits in w, and then swapping the bits within each pair. For example, swap$(01) = \{01, 10\}$ and

$$\text{swap}(0101) = \{0101, \underline{10}01, 0\underline{01}1, 01\underline{10}, \underline{1010}\}$$

For a language L, let $\textsc{Swap}(L) = \bigcup_{w \in L} \underline{\text{swap}(w)}$. Prove that if L is regular, so is $\textsc{Swap}(L)$.

Since L is regular, $\exists$ a DFA $M = (Q, \Sigma, \delta, s, A)$ s.t. $L(M) = L$. We'll def an NFA $N = (Q', \Sigma, \delta', s', A')$.

- $Q' = Q \times \{\varepsilon, 0, 1\}$
- $s' = (s, \varepsilon)$
- $A' = A \times \{\varepsilon\}$

$\delta'((q, \varepsilon), a) = \{(\delta(q, a), \varepsilon), (q, a)\}$ $\forall q \in Q, a \in \Sigma$

$\delta'((q, a), b) = \{(\underbrace{\delta(\underbrace{\delta(q, b)}, a)}, \varepsilon)\}$ $\forall q \in Q$, $a, b \in \Sigma$

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

(Spring 2025 Homework) We define an operation DELETEONES that removes all 1s in a string. For example,

$$\textsc{DeleteOnes}(101010010001) = 0000000$$

Prove that for a regular language L,

$$\textsc{DeleteOnes}(L) = \{\textsc{DeleteOnes}(w) \mid w \in L\}$$

is also regular.

We define NFA $M' = (Q', s', A, \delta')$

$Q' = Q$

$s' = s$

$A' = A$

$\delta'(q, 0) = \{\delta(q, 0)\}$

$\delta'(q, \varepsilon) = \{\delta(q, 1)\}$

$\forall q \in Q$

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)

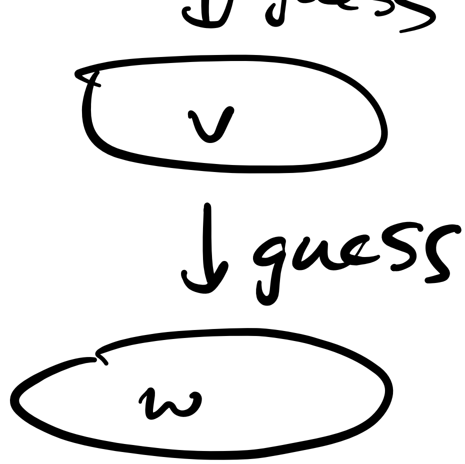
(g) The language $\{xx^R y : x, y \in \{0,1\}^*\}$ is regular.

(h) If L is regular, then $\textsc{SelfFold}(L) = \{a_1 a_n a_2 a_{n-1} \cdots a_{\lceil \frac{n}{2} \rceil} : a_1 a_2 \cdots a_n \in L\}$ 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 $L_1, L_2, \ldots$ s.t. $L_{i-1} \subseteq L_i$. The language $\cup_{i=1}^{\infty} L_i$ is context free.



$Q' = Q \times \{u, v, w\}$

$s' = (s, u)$

$A' = A \times \{w\}$

$\delta'((q,u),a) = \{(\delta(q,a), u)\}$ for $q \in Q$, $a \in \Sigma$

$\delta'((q,w),a) = \{(\delta(q,a), w)\}$ "

$\delta'((q,u),\varepsilon) = \{(\delta(q,a), v) : \underline{a \in} \Sigma\}$

$\delta'((q,v),\varepsilon) = \{(\delta(q,a), v), (\delta(q,a), w) : a \in \Sigma\}$

All missing transitions go to $\emptyset$

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 $\{w \in \Sigma^* \mid \#_0(w) \geq 2$ 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 $\{w \in \Sigma^* \mid \#_0(w) \geq 2$ 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) $\{ww^R : w \in \{0,1\}^*$ and $|ww^R| \equiv 1 \pmod 3\}$

CFGs

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

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

(b) $\{x\$y : x, y \in \{0,1\}^*$ and $\#_0(x) = \#_1(y)\}$ $\quad (\Sigma = \{0, 1, \$\})$

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) $\{0^x 1^y 2^z : x - y = z\}$

b. $S \rightarrow 0S1 \mid 1S \mid S0 \mid \$$

001\$1100

S0

S0

0S1

0S1

1S

Feedback

go.acm.illinois.edu/374A_feedback