

ECE-374-B: Algorithms and Models of Computation, Fall 2025

Midterm 1 – September 25, 2025

-
- You can do hard things! Grades do matter, but not as much as you may think, but then life is uncertain anyway, so what.
 - **Don't cheat.** The consequence for cheating is far greater than the reward. Just try your best and you'll be fine.
 - **Please read the entire exam before writing anything.** Most problems have multiple parts. Make sure you check the front and back of all the pages!
 - This is a closed-book exam. At the end of the exam, you'll find a multi-page cheat sheet. *Do not tear out the cheatsheet!* It messes with the auto-scanner. No outside material is allowed on this exam.
 - You should write your answers legibly and in the space given for the question. Overly verbose answers will be penalized.
 - Scratch paper is available on the back of the exam. *Do not tear out the scratch paper!* It messes with the auto-scanner.
 - **You have 75 minutes (1.25 hours) for the exam.** Manage your time well. *Do not spend too much time on questions you do not understand and focus on answering as much as you can!*
 - Proofs are required only if we specifically ask for them. Even then, none of the questions require long inductive proofs. You are only required to give a short explanation of why your answer is correct.
-

Name: _____

NetID: _____

1 Short Answer (Regular) - 15 points

Unless the question asks for it, no explanation is required for your answers for full credit. Keep any explanations of your answers to 2 sentences maximum.

- a. Write the **recursive** definition for the following language ($\Sigma = \{0, 1\}$):

$$L_{1a} = \{0^n 1^n | n \geq 0\}$$

Solution:

Base case: $\varepsilon \in L_{1a}$

Recursive case: for a $w \in L_{1a}$, $0w1 \in L_{1a}$



- b. Write the regular expression for the following languages ($\Sigma = \{0, 1\}$):

- i $L_{1bi} = \{w | w \in \Sigma^*, w \text{ does not contain the subsequence } 0000\}$

Solution: $L_{1bi} = 1^*(0 + \varepsilon)1^*(0 + \varepsilon)1^*(0 + \varepsilon)1^*$



- ii $L_{1bii} = \{w | w \in \Sigma^*, w \text{ all strings with alternating } 0\text{'s and } 1\text{'s}\}$ ¹

Solution: $L_{1bii} = (0 + \varepsilon)(10)^*(1 + \varepsilon)$

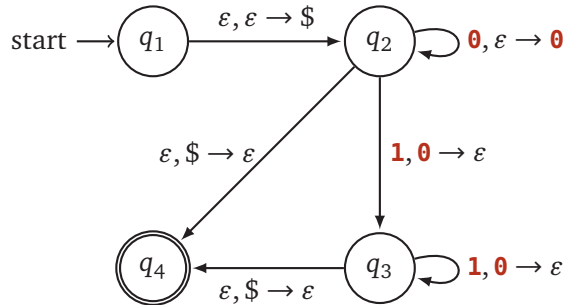


¹strings where no adjacent symbols are the same

2 Short Answer (Automata fun) - 10 points

Unless the question asks for it, no explanation is required for your answers for full credit. Keep any explanations of your answers to 2 sentences maximum.

a. What language does the push-down automata shown below represent?²



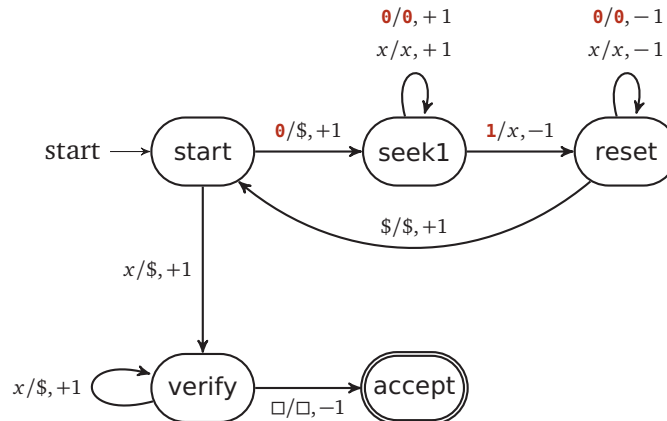
Solution: The above PDA represents the language:

$$L = \{0^n 1^n | n \geq 0\} \quad (1)$$

The PDA for this language works by first reading and pushing the 0's onto the stack. Once the 0's are done being read, the machine matches all the 0's with the same number of 1's. Therefore, this is the language where the number of 0's and 1's are the same. There is also the option to not push any 0's onto the stack and directly go to the accept state. Therefore we can also accept the empty string. ■

²I don't want a English description of what the PDA does, I want a definition of a language: $L = \{..\}$

b. What language does the Turing machine below represent?



Solution: The above Turing machine represents the language:

$$L = \{0^n 1^n | n > 0\} \quad (2)$$

There are 5 states being used in this machine and they are as follows:

- **start:** This state is the beginning of the tape. Scanning to the right, if it reads a 0 mark it as “read” by replacing it with \$ and move to **seek1**, else if it reads see a x move to **verify** since it means that all 0’s have been read.
- **seek1:** Scan right looking for a 1. Skip over symbols until it reads a 1, which it marks as “read” by replacing it with x and then moves to **reset**.
- **reset:** After having found a 1, move all the way back to the left to until you are at the rightmost \$ (which represents the previous 0 which was marked as “read”), and return back to **start** and start scanning to the right.
- **verify:** Scan right and verify that the rest of the tape only has x’s (meaning that all the 1’s were read and there were no 1’s that were unread). In other words, check that the number of 1’s matches up with the number of 0’s, and if true, move to **accept**.
- **accept:** The accept state, after having read the language $\{0^n 1^n | n > 0\}$.

Since the only way to move to **verify** is by scanning a “read” 1, which by construction, must be matched with a “read” 0, the language cannot accept the empty string. ■

3 Checking transformer models- 15 points

I want to prove that $L' = \{1w \mid w \in L, \}$ is regular, given L is regular. I remember seeing a similar question on the SP 22-Exam. I passed my proof request, along with the sample-exam problem and solution to two different LLMs. Letting $M = (Q, \Sigma, \delta, s, A)$, the LLMs produced the following language transformations:

GPT-5 Response: <https://chatgpt.com/share/68d16412-7184-800f-8569-40e03b8228d5>

$$\begin{aligned} Q' &= Q \times \{\text{pre}, \text{post}\} \\ s' &= (s, \text{pre}) \\ A' &= \{(q, \text{post}) \mid q \in A\} \\ \delta'((q, \text{post}), a) &= \{(\delta(q, a), \text{post})\}, \quad a \in \Sigma \\ \delta'((q, \text{pre}), \varepsilon) &= \{(\delta(q, 1), \text{post})\} \end{aligned}$$

Grok Response: <https://grok.com/c/d22ef5b8-4235-4743-9f28-69071d312a7a>

$$\begin{aligned} Q' &= Q \cup \{s'\} \\ s' &= s' \\ A' &= A \\ \delta'(s', 1) &= \{s\} \\ \delta'(q, a) &= \{\delta(q, a)\}, \quad q \in Q, a \in \Sigma \end{aligned}$$

Spoiler alert: Both these solutions are correct!

- a. Suppose we need to simulate the NFA resulting from one of the above transformations in a memory constrained environment. Which LLM's NFA is more memory-efficient? Succinctly (1-2 sentences) justify your answer.

Solution: use sum of states and transitions as a proxy for memory.

$$\begin{aligned} Q'_{\text{GPT}} &= 2|Q|, \\ Q'_{\text{Grok}} &= |Q| + 1, \end{aligned}$$

$$\begin{aligned} \delta'_{\text{GPT}} &= |\delta| + |Q|, \\ \delta'_{\text{Grok}} &= |\delta| + 1 \end{aligned}$$

$$|Q'_{\text{Grok}}| + |\delta'_{\text{Grok}}| < |Q'_{\text{GPT}}| + |\delta'_{\text{GPT}}|$$

$$\therefore \text{Mem}_{\text{Grok}} < \text{Mem}_{\text{GPT}}$$

■

- b. We decide to run incremental subset construction on each of the above NFAs. Which one leads to a smaller DFA?

Solution: This one has *more layers than an onion*. Each demarkate a unique level of conceptual understanding. Nicely done Sumedh.

Wrong Answer 0: Because Grok's NFA is smaller, its DFA must be.

$Q'_{\text{Grok}} < Q'_{\text{GPT}} \Rightarrow \text{DFA}_{\text{Grok}}$ is smaller.

Error: Directly compares NFA size from part a without recognizing that this problem asks about DFAs.

.... peeling one layer deeper

Wrong Answer 1: Since Grok's powerset is smaller, its DFA must be.

$Q_{\text{Grok,DFA}} \subseteq \mathcal{P}(Q \cup \{s'\}) = 2^{|Q|+1},$

$Q_{\text{GPT,DFA}} \subseteq \mathcal{P}(Q \times \{\text{pre}, \text{post}\}) = 2^{2|Q|},$

$\nRightarrow |Q_{\text{Grok,DFA}}| < |Q_{\text{GPT,DFA}}|,$

Error: You can take a smaller subset of a larger set. Fails to note that incremental subset is a greedy exploration algorithm and need not select elements from $\mathcal{P}(Q)$.

.... peeling one layer deeper

Almost Correct Answer: Grok and GPT produce the same-size DFA.

Intuition: Grok adds s' and then proceeds with standard DFA execution for L .

GPT uses $(s, \text{pre}) \xrightarrow{\epsilon} (\delta(s, 1), \text{post})$ and proceeds with the DFA execution of L **after** 1

So they are be the same.

Error: What about the fail state(s)?

.... peeling one layer deeper

Correct Answer: GPT's DFA is strictly smaller by 1.

Reason: Grok's s' on 0 yields \emptyset (dead state).

For GPT, ϵ -closure(s, pre) = $\{(s, \text{pre}), (\delta(s, 1), \text{post})\} \Rightarrow$ no 0 edge from (s, pre)

\Rightarrow no dead state for GPT.

$|\text{DFA}_{\text{Grok}}| = |\text{DFA}_{\text{GPT}}| + 1.$ ■

- c. We now want to build an automaton for $\overline{L'}$. Suppose we take Grok's response and swap accepting and rejecting states. Would this work?

Solution: This one has layers too, not as many as an onion. Maybe a shallot?

Almost Correct Answer: Yes! Every transition of Grok's NFA leads to a set containing a single element...wait that's just a DFA right? And for DFAs, flipping accepting and rejecting states **does** produce the complement!.

Error: *What happens on inputs with a prefix of 0? There is no transition defined and thus we reject!*

Correct Answer: No! Although Grok's automaton resembles a DFA, the transition from s' on input 0 is undefined. Hence, swapping accepting and rejecting states does not yield the complement, since some inputs have no valid transition.

Wrong Answer: Yes because of closure. Since L' is regular $\overline{L'}$ is regular

Error: No mention of the correctness of state swapping...

Wrong Answer: Yes. For NFAs/DFAs you can freely swap accepting and rejecting states.

Error: Swapping the accepting and rejecting states in a NFA does not always produce the complement. ■

- d. Reread the first sentence of the problem. The goal was: "Prove that $L' = \{1w \mid w \in L\}$ is regular, assuming L is." Prove this in ≤ 3 lines using regular closure properties.

Solution: Concatenation of two regular languages.

Correct Answer: 1 is regular because it is finite

L is regular (as stated in problem)

The concatenation of two regular languages ($\{1\} \cdot L$) is regular.

Wrong Answer: 1 is regular, $1 \cup L$ is regular.

Error: Union and concatenation are NOT the same.

Wrong Answer 2: L is regular, $1 \cap L$ is regular

Error: Intersection and concatenation are NOT the same. ■

4 Language Transformation - 15 points

A run in a string w is a maximal non-empty substring of w in which all symbols are equal. For example, the string 011111100010000 consists of five runs(0, 111111, 000, 1, 0000).

For any string $w \in \{0, 1\}^*$, let $\text{compact}(w)$ denote the string obtained by replacing each run with a single symbol from that run. For example, $\text{compact}(\epsilon) = \epsilon$ and $\text{compact}(1101111001) = 10101$

Let L be an arbitrary regular language.

Prove that the language $\text{compact}(L) = \{\text{compact}(w) | w \in L\}$ is regular. ³

Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L . We construct an NFA $M' = (Q', s', A', \delta')$ with $\text{compact}(L)$ as follows.

Intuitively, to decide if a string w should be accepted by M' or not, we should try to **expand every single bit in w (0 into 0^+ and 1 into 1^+) and check if any of the resulting strings is accepted by M** . Besides, if w does not have alternating bit, it should get automatically rejected.

To do this, we just need to **keep track of the latest bit M' reads** so that we can decide whether we should expand it to 0^+ or 1^+ , and it also helps to verify if the next bit is valid or not.

- For all $q \in Q$ and $a \in \{0, 1\}$, the state (q, a) means (the simulation of) M is in state q and the previous bit M' has read is a , if there is any, and the next input bit M' is expecting is \bar{a} . **Note: $\bar{0} = 1$ and $\bar{1} = 0$**
- The state s' means the starting state of M' where no bit has been read yet.

$$Q' = Q \times \{0, 1\} \cup \{s'\}$$

$$s' = s'$$

$$A' = A \times \{0, 1\}$$

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

$$\delta'((q, a), \epsilon) = (\delta(q, a), a)$$

$$\delta'(s', \epsilon) = \{s' \times \{0, 1\}\}$$

$$\delta'((q, a), a) = \emptyset$$

$$\Sigma' = \Sigma$$

- The first six lines are all you need for full credit, the second-last line is not necessary as this is an NFA and all un-specified transition will just be treated as resulting in a trash-state. The last line is trivial.

³Hint: Consider the string w described above. We can break w down into runs of **0**'s and **1**'s as described in the problem. So we to construct a new machine for $\text{compact}(L)$ that accepts the first **0**, then ϵ 's in the places of any subsequent **0**'s, then accepts a **1**, then ϵ 's instead of the following **1**'s, then a **0**, then ϵ 's instead of the following **0**'s, etc.

- Since we can't have multiple starting state, we just create a new starting state with ϵ transition to $\{s \times \{0, 1\}\}$, which allows the first bit to be either 0 or 1. We are using epsilon transition as we might have $\epsilon \in L$ which implies $s \in A$.

I will also address some common mistakes from off-track solution explaining why these methods do not work:

- It is true that any string that belongs to $\text{compact}(L)$ has alternating bit, but the reverse implication is incorrect. Think about the case where $L = \{000\}$, clearly $w = 101$ has alternating bit while we also have $w \notin \text{compact}(L)$. So, there is no regular expression that can be use to describe $\text{compact}(L)$ if we don't know what L is.
- Same reason as above, it's impossible to draw one DFA/NFA out without knowing L . It's only possible to use language transformation and use 5-tuples to define the NFA for L' .
- Proof in the form of claiming $L = L_1 \cdot L_2 \cdot \dots \cdot L_n$ where $L_i \in \{0^*, 1^*\}$ and saying replacing every run with one digit from that run will cause the resulting language be regular makes no sense. First, regular language is not closed under deletion/replacement. And n here is not bounded, union/intersection/concatenation is only closed under finite number of times.
- Assume L_1 is a regular language and L_2, L_3 are two arbitrary languages. $L_1 \subset L_2$ and $L_3 \subset L_1$ doesn't tell anything about the regularity of L_2, L_3 .
- We titled this question "Language Transformation", which should remind you of Lab 3. Submissions that do not use language transformation will receive no credit, as that indicates either an attempt to take a shortcut or a lack of understanding of the topic. Submissions that are incorrect but demonstrate some application of language transformation concepts may receive partial credit, evaluated on a case-by-case basis.



5 Language classification I (2 parts) - 15 points

Let $\Sigma = \{0, 1\}$, and define the following languages:

$$L_x = \{w \in \Sigma^* \mid \text{every } 0 \text{ is immediately followed by at least two } 1\text{'s}\}$$

$$L_y = \{0^n 1^n \mid n \geq 0\}$$

Let $L_5 = L_x \cap L_y$.

- a. Is L_5 regular? Indicate whether or not by circling one of the choices below. Either way, prove it.

Solution: ☒ regular ☐ not regular

Let's break down L_x first. Strings in L_x would either have zero or more repetitions of **1**'s or **011**'s, since **0** must be followed by at least two **1**'s. It is a regular language with expression $(1 + 011)^*$.

Next, let's consider L_y . It is not a regular language. Why? Because $F = \{0^i \mid i \geq 0\}$ is a fooling set for L_y . We can prove it by considering a pair of strings $0^i, 0^j \in F$ such that $i \neq j$. For $w = 1^i$, $0^i w \in L_y$ and $0^j w \notin L_y$.

Since L_y is not regular, we can not use the closure properties to prove regularity of L_5 . This applies to proving non-regularity as well, because the intersection of a regular language and a non-regular language is not necessarily non-regular.

Instead, we can determine the strings in L_5 by examining the intersection. L_y contains strings with **0**'s followed by **1**'s, and at least two **1**'s must follow a **0**, $L_x \cap L_y$ contains only the empty string, i.e., $L_5 = \{\epsilon\}$. Note that L_5 is not an empty set; it contains the empty string. Because both languages contain the empty string.

Finite languages are regular. Hence, L_5 is regular. ■

- b. Is L_5 context-free? Indicate whether or not by circling one of the choices below. Either way, prove it.

Solution: ☒ context-free ☐ not context-free

Since L_5 is regular, it also will be context-free. So proof for 5.a should suffice.

Alternatively, we can use the closure properties to prove it as follows:

- We have already proved that L_x is regular. See 5.a.
- L_y can be generated by $S \rightarrow \epsilon | 0S1$, which proves that L_y is context-free.
- The intersection of a context-free language and a regular language is always context-free. Hence, L_5 is context-free.

We can also write a CFG for L_5 to prove that it is context-free. We have already derived that $L_5 = \{\epsilon\}$ (in 5.a). The CFG will have the following production rule: $S \rightarrow \epsilon$. ■

6 Language classification II (2 parts) - 15 points

Let $\Sigma_6 = \{0, 1\}$ and

$$L_6 = \{0^n 1^n w \mid w \in \Sigma^* \text{ and } n \geq 1\}^4$$

- a. Is L_6 regular? Indicate whether or not by circling one of the choices below. Either way, prove it.

Solution: regular

☒ not regular

We can use fooling set to prove this language is not regular.

Let's consider the fooling set $F = \{0^i \mid n > 1\}$.

Let x and y be arbitrary strings in F .

Then $x = 0^i$, $y = 0^j$ for some positive integers $i < j$.

Let $z = 1^i = 1^i w^*$, where w^* is ϵ .

Then $xz = 0^i 1^i \in L_6$.

And $yz = 0^j 1^i \notin L_6$, because $i \neq j$.

Thus, F is a fooling set for L_6 .

Because F is infinite, L_6 is not regular.

We need $i < j$ because if $i > j$, we have j zeros and i ones in $yz = 0^j 1^i$, we can treat the last $i - j$ ones as w^* , so in this case, $yz = 0^j 1^i \in L_6$

■

- b. Is L_6 context-free? Indicate whether or not by circling one of the choices below. Either way, prove it.

Solution:

☒ context-free

☐ not context-free

If you have a correct argument for regularity, you could have just said the “language is regular and therefore, context-free.” But let's say you didn't want to take that risk; here's how you prove context-free-ness directly:

We can prove this language is context free by constructing corresponding PDA.

$A \rightarrow SR$ (A is the start symbol)

$S \rightarrow 0S1 \mid 01$ (forces at least one matched 0/1 pair)

$R \rightarrow 0R \mid 1R \mid \epsilon$ (any binary suffix)

■

⁴Yes this is very *similar* to a lab problem, but it is different. Regurgitating a memorized answer will get you in trouble. Take a moment and *think*.

7 Language classification III (3 parts) - 15 points

During the language classification portion of the exam, when Sumedh wasn't looking, a student took out his cellphone asked GPT-5 to help classify the following language as regular, irregular, context-free, or not-context-free.

$$L_7 = \{w \in \{0, 1\}^* \mid |w| > 3 \text{ and every run of 1s in } w \text{ has odd length}\}$$

- a. After looking at the paper to the left of him, the student guesses that the language is regular, so he tries to come up with a regex to prove it! He asks Chat-GPT the following.

Regex Prompt

Write a regular expression for strings where every run of 1's is odd and is > 3 .

GPT-5 Regex: <https://chatgpt.com/c/68cdda9d-d5eo-832f-88df-c9d12778a754>

$0^*((111(11)^+)0^*)$

Is this regular expression correct or incorrect. If correct, succinctly explain why. Otherwise, provide a single counterexample.

Solution: The regular expression is **incorrect**. Consider the strings such as **1110111**, **0000**, **1110**. The strings belong to L_7 , but are not represented by the given regular expression. A valid counterexample must be in the language L_7 , and is not represented by the given regular expression. Strings such as **000**, **0011** are not valid counterexamples.

■

- b. The student suddenly gets a suspicion that the language might be incorrect and asks GPT-4o for a fooling set proof for irregularity.

Regex Prompt

Consider the set of binary strings where every run of 1's is odd, but is also strictly greater than 3. Prove that this language is irregular via fooling sets. Give me the prefix and suffixes necessary.

Fooling Set Proof: <https://chatgpt.com/c/68ce1216-95d4-8327-ab8c-19c3f1313488>

$F = \{x_n = 1^{2n+1} \mid n = 0, 1, 2, 3, \dots\}$.

Consider $x_i = 1^{2i+1}$, $x_j = 1^{2j+1}$, where $i \neq j$.

Use the suffix $y = 0$. If $i \geq 2$, then $x_i y \in L$

If $j < 2$, then $x_j y \notin L$

Thus, we have constructed an infinite fooling for L . L is not regular.

Is the following proof correct or incorrect? If incorrect, identify the error.

Solution: The proof is **incorrect**. Consider $x_i = 11111$ and $x_j = 111$. Then, for the suffix $y = 0$, both $x_i y$ and $x_j y$ are in L . Moreover, for $i, j > 1$, x_i and x_j are not distinguishable. Therefore, F is not a valid fooling set. ■

- c. After all this, really there is only one question left: Is L_7 regular or not? Either way, prove it!

Solution: (Closure Property)

not regular

regular

Let L_x and L_y be arbitrary languages where

$$L_x = \{w \in \{0, 1\}^* \mid |w| > 3\}$$

$$L_y = \{w \in \{0, 1\}^* \mid \text{every run of 1s in } w \text{ has odd length}\}$$

such that $L_7 = L_x \cap L_y$

The regular expression for L_x is $(0 + 1)(0 + 1)(0 + 1)(0 + 1)(0 + 1)^*$

The regular expression for L_y can be obtained by first thinking about how to obtain odd runs of 1s $\rightarrow (1(11)^*)$. Furthermore, we have to make sure that there is at least one 0 between the runs of 1s. After this, a potential edge case could also be how to accept strings like 000000 and to let the string end with 0s or with a final odd run of 1s.

Based on the above we can obtain the regular expression for L_y as

$$0^*(1(11)^*0^+)^*(\epsilon + 1(11)^*)0^*$$

Since we are able to find regular expressions for both L_x and L_y , we know that these languages are regular. Furthermore, since \cap is closed under regular languages and $L_7 = L_x \cap L_y$, we can say that L_7 is regular.

■

Solution: (DFA/Regex construction)

not regular

regular

Another way to solve the question is to construct an NFA, DFA or regex for the entire problem.

Lets define a DFA $M(Q, s, A, \delta, \Sigma)$ which accepts the Language L_7

$$Q = \{(a, b) | a \in \{Z, O, E, D\}, b \in \{c_0, c_1, c_2, c_3, c_{\geq 4}\}\}$$

$$s = (Z, c_0)$$

$$A = \{(Z, c_{\geq 4}), (O, c_{\geq 4})\}$$

Z = character reading was a **0**

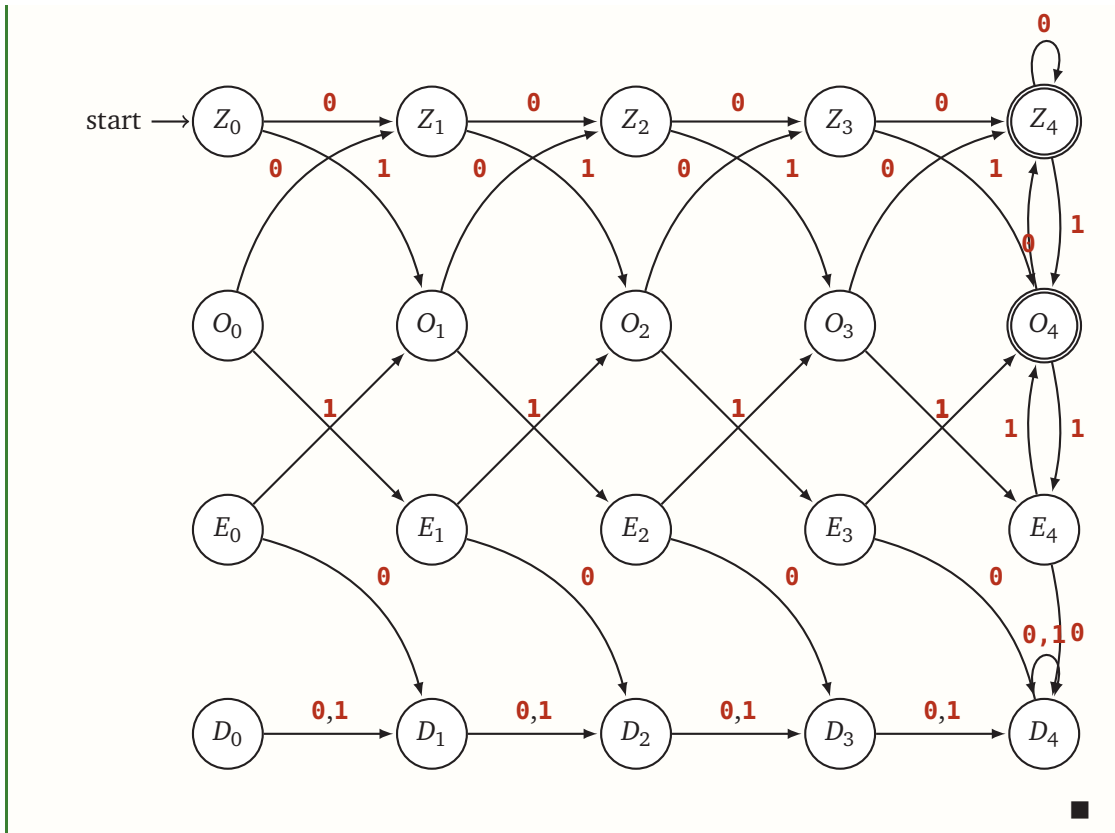
O = reading a 1-run with an **odd** count

E = reading a 1-run with an **even** count

D = sink state (an even run of **1**s was found)

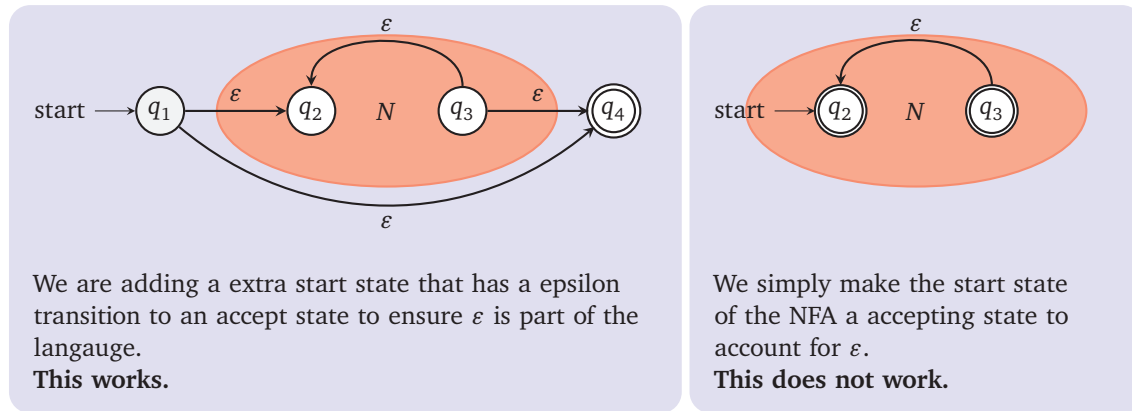
The $\{c_0, c_1, c_2, c_3, c_{\geq 4}\}$ values represent the length of the total string.

Based on the above, we can construct a DFA as follows:



EXTRA CREDIT - 5 POINTS

In our lecture (and cheatsheet) we constructed a transformation for L^* from the NFA of L to show that regular languages are closed under Kleene star. But in this transformation I specified that a new start state was required. Can you show me why?



In other words, **show me an example** of a NFA (which represents a language L) where the left transform represents L^* but the right transformation does not represent L^* .

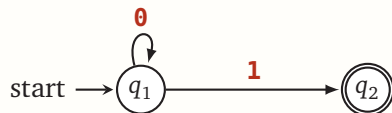
Solution: Given an NFA N that accepts an arbitrary regular language L , the left transform for $L = L^*$ introduces an extra start state with ϵ -transitions

The question asks why we **must** add this new state, i.e., why the tempting shortcut “just make the old start state accepting” can fail.

(There can be many counterexamples to this question, one common counterexample that is easy to understand is down below).

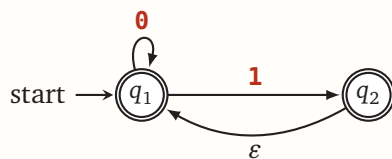
Let $L = 0^*1$

A very small NFA for L would look as follows:



This NFA accepts exactly zero or more **0**'s followed by a single **1**. Note that q_1 has an incoming transition (the **0**-loop) (Examples: ϵ , **1**, **01**, **001**, ...)

Now according to right transform given to us, we turn the start state into an accepting state (q_1) and add an ϵ -transition from the final state back to the start state.



According to the new NFA above, every string 0^k with $k \geq 0$ is accepted by staying at q_1 the whole time. The NFA recognizes languages strictly larger than L^*

In particular $0 \in L(N_{\text{wrong}})$ but as we saw from the left transform $0 \notin L^*$.

$L(N_{\text{wrong}})$ is introducing new strings not in L^* . Hence, the right transform is incorrect and extra state is required. ■

This page is for additional scratch work!

ECE 374 B Language Theory: Cheatsheet

1 Languages and strings

Languages

- An *alphabet* Σ is a **finite** set of symbols.

Definitions A *string* in Σ^* is a **finite** sequence of symbols in Σ .

- A *language* is L is a set of strings over some alphabet.

All languages represent mathematical problems.
Example: multiplication of two integers:

$$L_{MULT2} = \left\{ \begin{array}{ccc} 1 \times 1|1, & 1 \times 2|2, & 1 \times 3|3, \dots \\ 2 \times 1|2, & 2 \times 2|4, & 2 \times 3|6, \dots \\ \vdots & \vdots & \vdots \\ n \times 1|n, & n \times 2|2n, & n \times 3|3n, \dots \end{array} \right\} \quad (1)$$

Language operations

- For languages A, B the *concatenation* of A, B is $AB = \{xy \mid x \in A, y \in B\}$.
- For languages A, B , their *union* is $A \cup B$, *intersection* is $A \cap B$, and *difference* is $A \setminus B$ (also written as $A - B$).
- For language $A \subseteq \Sigma^*$ the *complement* of A is $\bar{A} = \Sigma^* \setminus A$.
- Σ^n is the set of all strings of length n .
- $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$ is the set of all strings over Σ .
- $\Sigma^+ = \bigcup_{n \geq 1} \Sigma^n$ is the set of non-empty strings over Σ .

Strings

- The *length* of a string w (denoted by $|w|$) is the number of symbols in w .
- For integer $n \geq 0$, Σ^n is set of all strings over Σ of length n . Σ^* is the set of all strings over Σ .

Definitions

- Σ^* is the set of all strings of all lengths including empty string.
- ϵ is a *string* containing no symbols.
- \emptyset is the *empty set*. It contains no strings.

- If x and y are strings then xy denotes their concatenation. Recursively:

- $xy = y$ if $x = \epsilon$
- $xy = a(wy)$ if $x = aw$

- v is *substring* of $w \iff$ there exist strings x, y such that $w = xvy$.

- If $x = \epsilon$ then v is a *prefix* of w
- If $y = \epsilon$ then v is a *suffix* of w

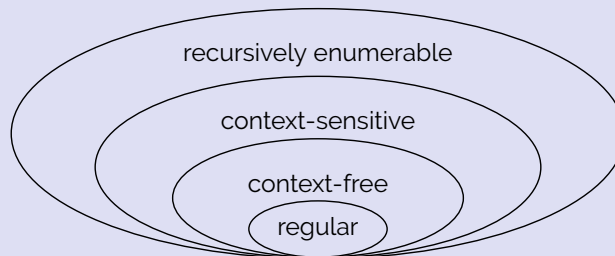
- A *subsequence* of a string $w = w_1w_2 \dots w_n$ is either a subsequence of $w_2 \dots w_n$ or w_1 followed by a subsequence of $w_2 \dots w_n$.

- If w is a string then w^n is defined inductively as follows:
 $w^n = \epsilon$ if $n = 0$ or $w^n = ww^{n-1}$ if $n > 0$

String operations

2 Overview of language complexity

Overview



Grammar	Languages	Production Rules	Automaton	Examples
Type-0	recursively enumerable	$\gamma \rightarrow \alpha$ (no constraints)	Turing machine	$L = \{w \mid w \text{ is a TM which halts}\}$
Type-1	context-sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$	linear bounded nondeterministic Turing machine	$L = \{a^n b^n c^n \mid n > 0\}$
Type-2	context-free	$A \rightarrow \alpha$	nondeterministic pushdown automata	$L = \{a^n b^n \mid n > 0\}$
Type-3	regular	$A \rightarrow aB$	finite state machine	$L = \{a^n \mid n > 0\}$

Meaning of symbols:

- a - terminal
- A, B - variables
- α, β, γ - strings in $\{a \cup A\}^*$ where α, β are maybe empty, γ is never empty

^aTable borrowed from Wikipedia: https://en.wikipedia.org/wiki/Chomsky_hierarchy

3 Regular languages

Regular language - overview

A language is regular if and only if it can be obtained from finite languages by applying

- union,
- concatenation or
- Kleene star

finitely many times. All regular languages are representable by regular grammars, DFAs, NFAs and regular expressions.

Regular expressions

Useful shorthand to denotes a language.

A regular expression r over an alphabet Σ is one of the following:

Base cases:

- \emptyset the language \emptyset
- ε denotes the language $\{\varepsilon\}$
- a denote the language $\{a\}$

Inductive cases: If r_1 and r_2 are regular expressions denoting languages L_1 and L_2 respectively (i.e., $L(r_1) = L_1$ and $L(r_2) = L_2$) then,

- $r_1 + r_2$ denotes the language $L_1 \cup L_2$
- $r_1 \cdot r_2$ denotes the language $L_1 L_2$
- r_1^* denotes the language L_1^*

Examples:

- 0^* - the set of all strings of 0s, including the empty string
- $(00000)^*$ - set of all strings of 0s with length a multiple of 5
- $(0 + 1)^*$ - set of all binary strings

Nondeterministic finite automata

NFAs are similar to DFAs, but may have more than one transition destination for a given state/character pair.

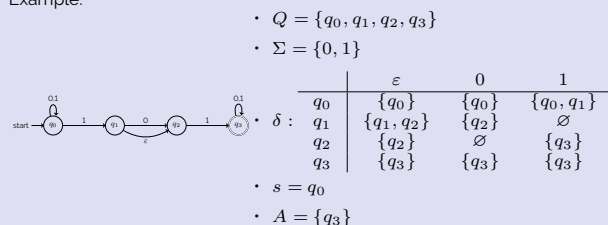
An NFA N accepts a string w iff some accepting state is reached by N from the start state on input w .

The language accepted (or recognized) by an NFA N is denoted $L(N)$ and defined as $L(N) = \{w \mid N \text{ accepts } w\}$.

A nondeterministic finite automaton (NFA) $N = (Q, \Sigma, s, A, \delta)$ is a five tuple where

- Q is a finite set whose elements are called states
- Σ is a finite set called the input alphabet
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q)$ is the transition function (here $\mathcal{P}(Q)$ is the power set of Q)
- s and Σ are the same as in DFAs

Example:



For NFA $N = (Q, \Sigma, \delta, s, A)$ and $q \in Q$, the ε -reach(q) is the set of all states that q can reach using only ε -transitions.

Inductive definition of $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$:

- if $w = \varepsilon$, $\delta^*(q, w) = \varepsilon\text{-reach}(q)$
- if $w = a$ for $a \in \Sigma$, $\delta^*(q, a) = \varepsilon\text{-reach}\left(\bigcup_{p \in \varepsilon\text{-reach}(q)} \delta(p, a)\right)$
- if $w = ax$ for $a \in \Sigma, x \in \Sigma^*$: $\delta^*(q, w) = \varepsilon\text{-reach}\left(\bigcup_{p \in \varepsilon\text{-reach}(q)} \left(\bigcup_{r \in \delta^*(p, a)} \delta^*(r, x)\right)\right)$

Regular closure

Regular languages are closed under union, intersection, complement, difference, reversal, Kleene star, concatenation, etc.

Deterministic finite automata

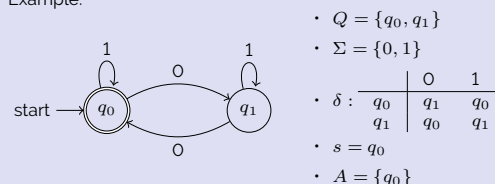
DFAs are finite state machines that can be represented as a directed graph or in terms of a tuple.

The language accepted (or recognized) by a DFA M is denoted by $L(M)$ and defined as $L(M) = \{w \mid M \text{ accepts } w\}$.

A deterministic finite automaton (DFA) $M = (Q, \Sigma, s, A, \delta)$ is a five tuple where

- Q is a finite set whose elements are called states
- Σ is a finite set called the input alphabet
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function
- $s \in Q$ is the start state
- $A \subseteq Q$ is the set of accepting/final states

Example:



Every string has a unique walk along a DFA. We define the extended transition function as $\delta^* : Q \times \Sigma^* \rightarrow Q$ defined inductively as follows:

- $\delta^*(q, w) = q$ if $w = \varepsilon$
- $\delta^*(q, w) = \delta^*(\delta(q, a), x)$ if $w = ax$.

Can create a larger DFA from multiple smaller DFAs. Suppose

- $L(M_0) = \{w \text{ has an even number of 0s}\}$ (pictured above) and
- $L(M_1) = \{w \text{ has an even number of 1s}\}$.

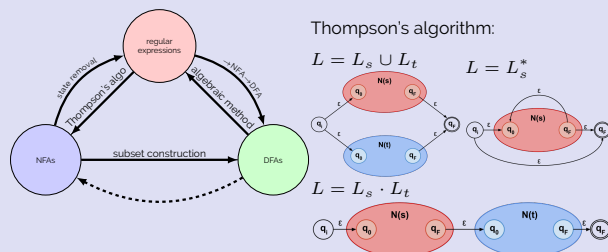
$L(M_C) = \{w \text{ has even number of 0s and 1s}\}$

Suppose $M_0 = (Q_0, \Sigma, s_0, A_0, \delta_0)$ and $M_1 = (Q_1, \Sigma, s_1, A_1, \delta_1)$. Then

- $Q = Q_0 \times Q_1 = \{(q_0, q_1) \mid q_0 \in Q_0, q_1 \in Q_1\}$
 - $s = (s_0, s_1)$
 - $\delta : Q \times \Sigma \rightarrow Q$, where $\delta((q_0, q_1), a) = (\delta_0(q_0, a), \delta_1(q_1, a))$
 - $A = \{(q_0, q_1) \mid q_0 \in A_0 \text{ and } q_1 \in A_1\}$
-

Regular language equivalences

A regular language can be represented by a regular expression, regular grammar, DFA and NFA.



Arden's rule: If $R = Q + RP$ then $R = QP^*$.

Fooling sets

Some languages are not regular (Ex. $L = \{0^n 1^n \mid n \geq 0\}$).

Two states $p, q \in Q$ are distinguishable if there exists a string $w \in \Sigma^*$, such that

$$\delta^*(p, w) \in A \text{ and } \delta^*(q, w) \notin A.$$

or

Two states $p, q \in Q$ are equivalent if for all strings $w \in \Sigma^*$, we have that

$$\delta^*(p, w) \in A \iff \delta^*(q, w) \in A.$$

$$\delta^*(p, w) \notin A \text{ and } \delta^*(q, w) \in A.$$

For a language L over Σ a set of strings F (could be infinite) is a fooling set or distinguishing set for L if every two distinct strings $x, y \in F$ are distinguishable.

4 Context-free languages

Context-free languages

A language is context-free if it can be generated by a context-free grammar. A context-free grammar is a quadruple $G = (V, T, P, S)$

- V is a finite set of *nonterminal (variable) symbols*
- T is a finite set of *terminal symbols* (alphabet)
- P is a finite set of *productions*, each of the form $A \rightarrow \alpha$ where $A \in V$ and α is a string in $(V \cup T)^*$. Formally, $P \subseteq V \times (V \cup T)^*$.
- $S \in V$ is the *start symbol*

Example: $L = \{ww^R \mid w \in \{0, 1\}^*\}$ is described by $G = (V, T, P, S)$ where V, T, P and S are defined as follows:

- $V = \{S\}$
- $T = \{0, 1\}$
- $P = \{S \rightarrow \varepsilon \mid 0S0 \mid 1S1\}$
(abbreviation for $S \rightarrow \varepsilon, S \rightarrow 0S0, S \rightarrow 1S1$)
- $S = S$

Pushdown automata

A pushdown automaton is an NFA with a stack.

The language $L = \{0^n 1^n \mid n \geq 0\}$ is recognized by the pushdown automaton:

A *nondeterministic pushdown automaton (PDA)* $P = (Q, \Sigma, \Gamma, \delta, s, A)$ is a **six** tuple where

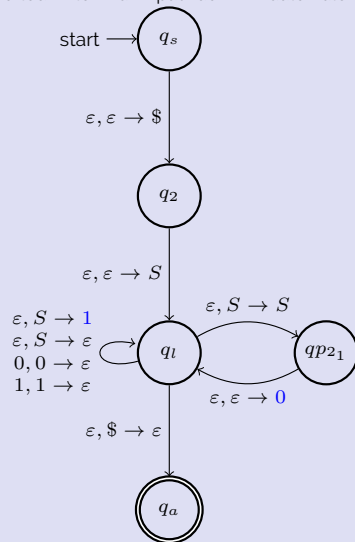
- Q is a finite set whose elements are called *states*
- Σ is a finite set called the *input alphabet*
- Γ is a finite set called the *stack alphabet*
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$ is the *transition function*
- s is the start state
- A is the set of accepting states

In the graphical representation of a PDA, transitions are typically written as $\langle \text{input read} \rangle, \langle \text{stack pop} \rangle \rightarrow \langle \text{stack push} \rangle$.

A CFG can be converted to a pushdown automaton.

The PDA to the right recognizes the language described by the following grammar:

$$S \rightarrow 0S1 \mid \varepsilon$$



Context-free closure

Context-free languages are closed under union, concatenation, and Kleene star.

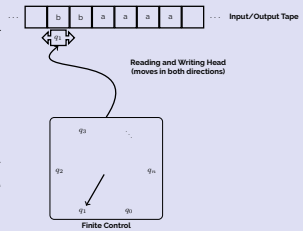
They are **not** closed under intersection or complement.

5 Recursively enumerable languages

Turing Machines

Turing machine is the simplest model of computation.

- Input written on (infinite) one sided tape.
- Special blank characters.
- Finite state control (similar to DFA).
- Every step: Read character under head, write character out, move the head right or left (or stay).
- Every TM M can be encoded as a string $\langle M \rangle$



Transition Function: $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow, \square\}$

$$\delta(q, c) = (p, d, \leftarrow)$$

- q : current state.
- c : character under tape head.
- p : new state.
- d : character to write under tape head
- \leftarrow : Move tape head left.

