Problem 0:

i. Construct an NFA that is equivalent to a regex, (a U b)*
ii. Construct an NFA that is equivalent to a regex, (ab U b)*
iii. Remove a state q_{rip} using the state elimination algorithm. (Fig. 1)
iv. Find a regex for a DFA. (Try to derive a regex by guessing. Also, try to derive one by applying the state elimination algorithm.) (Fig. 2)

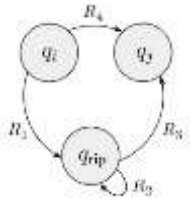

Fig. 1

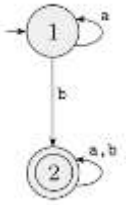

Fig. 2

v. For each of the following regular expressions, give two strings that are members and two strings that are not members—a total of four strings for each part. Assume the alphabet Σ = {a,b} in all parts.

a) a*b*

b) a(ba)*b

c) a*∪b*

vi. Generate a regex for the following languages.

a) L={w| w begins with a 1 and ends with a 0}

b) L={w| w contains at least three 1s}

c) L={w| w contains the substring 0101 (i.e., w = x0101y for some x and y)}

d) L={w| w has length at least 3 and its third symbol is a 0}

Problem 1: Designing Regular Expressions

Below are a list of alphabets and languages over those alphabets. For each language, write a regular expression for that language.

i. Let Σ= {a, b, c, d, e}. Write a regular expression for the language L= {w w $\in$ Σ^* | Σ^* | the letters in w are sorted alphabetically }. For example, abcde, bee, and a, belong to L but cabb is not.

ii. Write a regular expression for the complement of the language defined in (i).

Like NFAs and unlike DFAs, there isn't a simple construction that starts with a regex for L and turns it into a regex for $\bar{L}$.

iii. On Unix-style operating systems like macOS or Linux, files are organized into directories. You can reference a file by giving a path to the file, a series of directory names separated by slashes. For example, the path **/home/username/** might represent a user's home directory, and a path like **/home/username/Documents/PS7.tex** might represent a file. Paths that start with a slash character are called absolute paths and say exactly where the file is on disk. Paths that don't start with a slash are called relative paths and say where, relative to the current folder, a file can be found. For example, if I'm logged into my computer and am in my home folder, I could look up the file **Documents/PS7.tex** to find the file above. The general pattern here is that a file path consists of a series of directory or file names separated by slashes. That path might optionally start with a slash, but isn't required to, and it might optionally end with a slash, but isn't required to. However, you can't have two consecutive slashes.

Let Σ= {a, /}. Write a regular expression for L= { w $\in \Sigma^*$ | w represents the name of a file path on a Unix-style system}. For example, /aaa/a/aa/$\in$ L, /$\in$ L, a$\in$ L, /a/a/a/$\in$ L, and aaa/$\in$ L, but //a//$\notin$ L, a//a$\notin$ L, and $\varepsilon \notin$ L.

Problem 2: Regular expressions are frequently used in software systems to validate user input. For example, if a form on a website requires an email address, the website might use a regular expression to confirm that what the user types in is a valid email address before letting them submit the form. It helps prevent people from making accidental mistakes when signing up or giving contact information, and it can help prevent spam. However, there are risks with validating data too strictly. It's unfortunately common for websites to reject valid inputs because the person designing the website made an unwarranted assumption about what those inputs are supposed to look like.

Let's illustrate this with an example. In "real-world" regexes, the notation [A-Z] means the same thing as a regex (A ∪ B ∪ … ∪ Z), and the notation [A-z] has the same meaning as a regex (A ∪ B ∪ … ∪ Z a ∪ b∪ … ∪ z). Now, suppose a website asks the user to enter their full legal name. They use the following regex to do so: [A-z]+ [A-z]+ [A-z]+ For example, this would match the strings **Kenneth Lane Thompson** and **Alan Mathison Turing**.

Give an example of the full name of a real person – ideally someone you know, but it could also be a celebrity or other famous person – that does not match this regular expression. Briefly explain why their name doesn't match.

The lesson learned from this problem is to be ***careful when validating user data***. The human experience is rich and varied and many assumptions that you might take as basic and fundamental might be completely different from others' lived experiences. If you make unwarranted assumptions about user data – even as something as basic as "what is your name?" – you might lock people out from using a product or service.

Problem 3: The state elimination algorithm gives a way to transform a DFA or NFA into a regular expression. Let Σ= {a, b} and let L= {w has an even number of a's and an even number of b's}.

i. Draw a finite automaton for L.
ii. Apply the state elimination algorithm to the automaton.

Problem 4: A language is called ***finite*** if contains finitely many strings (that is, |L| is a natural number). Given a finite language , explain how to write a regular expression for . Briefly justify your answer; no formal proof is necessary. This shows that all finite languages are regular.