1) (10 pts) Design a DFA that accepts the following language over the alphabet \{a, b\}: \( L = \{ w \mid w \text{ has exactly one occurrence of the substring } "aa". \} \) Express your DFA as a state diagram drawing. Make sure to label the start state, all accept states and all transitions clearly. Note: the number of substrings of length 2 in a string of length \( n \) is \( n-1 \). Thus, two different substrings in a string may overlap and share characters in common.

\[ q_0 : 2a's \text{ have yet to be seen and our "current streak" is } 0a's. \]

\[ q_1 : 2a's \text{ have yet to be seen and our "current streak" is } 1a. \]

\[ q_2 : 2a's \text{ have been seen and our "current streak" is } 1a. \]

\[ q_3 : 2a's \text{ have been seen and our "current streak" is } 0a's. \]

\[ q_4 : \text{Already seen } "aa" \text{ at least twice.} \]

Grading

1 pt label start
1 pt label accept
2 pts all valid transitions shown
2 pts - rejecting all strings w/o "aa"
2 pts - accepting string w/one "aa"
2 pts - rejecting strings w/two or more "aa"'s.
2) (15 pts) Using the algorithm shown in the text to convert the NFA below to an equivalent DFA that accepts the same language. Remember to only create the "reachable" states in the DFA to save time.

```
Grading
1 pt label start
3 pts labeling subsets of \{q_0, q_1, q_2, q_3\}
3 pts having all valid transitions (DFA rule)
2 pts labeling accept states
6 pts - 1 pt each for transitions out of the 6 states (correctness)
```
3) (15 pts) Consider the process of converting a DFA to a regular expression that expresses the same language. The initial DFA has states q0, q1 and q2. In the beginning of the process states S and F are added, with an epsilon transition from S to the start state of the DFA, q0, and epsilon transitions from the accept states q1 and q2 to F. Finally, F is denoted as the only accept state. After ripping out q1, the 4 state GNFA looks like this:

\[ e = \text{epsilon}, U = \text{union} \]

![Diagram of DFA with states S, q0, q2, and F connected by transitions labeled 0, 1, 0 U 10*1, and 10* U e.]

Complete the process of determining the equivalent regular expression by first ripping out q0, followed by ripping out q2. (Note: credit will be taken off for ripping the states in the opposite order, so please follow these directions.)

Rip q0, need to calculate \( S \rightarrow q2, q2 \rightarrow q2 = \)

\[ (0 U 10*)0^*1 \]

Rip q2, need to calculate \( S \rightarrow F = \)

\[ 0^*1[(0 U 10*)0^*1]^*(10^* U E) \]

Equivalent R.E. = \( 0^*1[(0 U 10*)0^*1]^*(10^* U E) \)

Grading - 5 pts for each of the 3 transition calculations.
4) (10 pts) In class we proved that the language \( L_1 = \{0^n | n \in \text{Primes}\} \) over the alphabet \{0\} was not regular. Utilizing this result, show that the language \( L_2 = \{0^n | n \in \text{Composite}\} \) is not regular. (Note: for the purposes of this question, composite numbers are positive integers greater than 1 that can be expressed as a product of two non-necessarily distinct integers, neither of which is the number itself.) Do not use the pumping lemma in your proof. Try to use an indirect method that is easier. For a bit of extra credit, explain why using the pumping lemma to show this language isn’t regular directly is problematic.

Note that \( L_1 \cap L_2 = \emptyset \). Let \( L_3 = L_1 \cup L_2 = \{0^n | n \in \mathbb{Z}, n \geq 2\} \).

1 pt) \( L_3 \) is regular since we can create a 3 state DFA to recognize it. \( (>q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_0) \).

Note that \( L_2 = L_3 - L_1 = L_3 \cap \overline{L_1} \).
Also, \( L_1 = L_3 - L_2 = L_3 \cap \overline{L_2} \). (2 pts)

2 pts) Let’s assume to the contrary of what we are trying to prove, that \( L_2 \) is regular. It follows that \( \overline{L_2} \) is regular as well. (We proved this previously. Given a DFA \( D \) that accepts \( L_2 \), we create \( D' = \{Q, \Sigma, \delta, q_0, F\} \) the DFA \( D' = \{Q, \Sigma, \delta, q_0, F\} \) accepts \( \overline{L_2} \). All accepted strings in \( D \) are rejected in \( D' \); all rejected strings in \( D \) are accepted by \( D' \), as desired.)

Finally, we proved in class that if \( L_3 \) is regular and \( \overline{L_2} \) is regular, \( L_3 \cap \overline{L_2} \) must be as well. Since \( L_1 = L_3 \cap \overline{L_2} \), it follows that \( L_1 \) is regular. But, this contradicts that we previously (in class) showed that \( L_1 \) is \underline{not} regular. It follows that our initial assumption is wrong. Thus, \( L_2 \) is \underline{not} regular, as desired.

1 pt)
5) (15 pts) The context free grammar below is in the process of being converted to Chomsky Normal Form. In the form below, a separate start state has been added, but epsilon rules haven't been removed. Show the result of removing these epsilon rules.

\[ S_0 \rightarrow S \]
\[ S \rightarrow ABC \mid a \mid BB \mid C \]
\[ A \rightarrow ab \mid cC \mid \varepsilon \]
\[ B \rightarrow BC \mid BCB \mid ACA \mid \varepsilon \]
\[ C \rightarrow c \mid CA \mid CB \]

Remove \( B \rightarrow \varepsilon \), added rules:
\[ B \rightarrow C \mid CB \]
\[ S \rightarrow AC \mid B \mid \varepsilon \]  
\(5\) pts – 1 per rule

Remove \( A \rightarrow \varepsilon \), added rules:
\[ B \rightarrow AC \mid CA \]
\[ S \rightarrow BC \mid C \]  
\(6\) pts – 2 per rule

Remove \( S \rightarrow \varepsilon \), added rules:
\[ S_0 \rightarrow \varepsilon \]  
\(1\) pt

Grammar after removing epsilon rules:
\[ S_0 \rightarrow S \mid \varepsilon \]
\[ S \rightarrow ABC \mid a \mid BB \mid C \mid AC \mid B \mid BC \]
\[ A \rightarrow ab \mid cC \mid \varepsilon \]
\[ B \rightarrow BC \mid BCB \mid ACA \mid C \mid CB \mid AC \mid CA \]
\[ C \rightarrow c \mid CA \mid CB \]

Final ans – 3 pts
6) (15 pts) Consider the language of valid matching parentheses taken from the alphabet \{\(\langle\), \rangle\}, \{\[,\]\}. An example of a string in the language is \(\langle\langle00\rangle\rangle\). Design a PDA to accept this language. Note: your PDA should accept the empty string. Show the state diagram of your PDA clearly illustrating all six parts in the formal definition of all PDAs. Note: The only character that can close an \(\langle\) is \(\rangle\) and the only character that can close an \{ is \}. Thus, \(\{}\) is NOT a valid string in this language.

---

**Grading**

Start state  - 1 pt
Accept states - 2 pts
Correct form for transitions - 2 pts
Using \$ or equivalent to mark bottom - 2 pts
4 rules for pushing/popping chars - 2 pts each (8 pts total)
7) (15 pts) Let $L = \{a^ib^jc^k | i \geq j \geq k\}$ over the alphabet \{a, b, c\}. Using the Pumping Lemma for Context Free Languages, prove that $L$ is not context-free.

Assume $L$ is context free. This it satisfies the Pumping Lemma for context free languages. Let $p$ be the pumping length for $L$. Consider $s = a^pb^pc^p$, $|s| \geq p$ and $s \in L$. According to the pumping lemma, there exists a way to split $s = uvxyz$ such that $|vy| \geq 0$, $|vxy| \leq p$ and $uv^ixy^iz \in L$ for all $i \geq 0$.

The following are the 5 cases for $vxy$:

1. $vxy = a^i$, $i > 0$. In this case $uxz = a^{p-i}b^pc^p$, where $j = |vy| > 0$, and $uxz \notin L$ since $p-j < p$ (the number of $a$'s is less than the # of $b$'s.)

2. $vxy = a^ib^j$. In both cases $uxz$ either contains fewer $a$'s or fewer $b$'s than $c$'s because $vy$ must contain at least $1a$ or $1b$ since $|vy| > 0$. Thus $uxz \notin L$.

3. $vxy = b^i$. In both cases $u^2v^2xz^2 \notin L$ must contain either more $b$'s than $a$'s or more $c$'s than $a$'s since $|vy| > 0$ and $|vxy| \geq 2$. Thus $uv^2xy^2z \notin L$.

In all 5 cases, no matter how we try to split $s$, we find that the pumping lemma isn't satisfied since $uv^ixy^iz \notin L$ for all $i \geq 0$.

String choice: 3pts

8) (5 pts) What is the first ingredient listed on the list of ingredients for a granola bar?

Granola (5pts to all)