1. Draw the finite state machines, and write regular expressions that denote the strings recognized by the following finite automata:

(a) \( s_1 \) is the start state; \( s_2 \) is the final (accepting) state

\[
\begin{array}{c|cc}
& a & b \\
\hline
s_1 & s_2 \\
s_2 & s_3 \\
s_3 & s_2 \\
\end{array}
\]

(b) \( s_1 \) is the start state; \( s_4 \) is the final state

\[
\begin{array}{c|cccc}
& a & b & \\
\hline
s_1 & s_2 & s_3 & s_4 \\
s_2 & s_3 & s_4 \\
s_3 & s_4 \\
s_4 & s_4 \\
\end{array}
\]

(c) \( s_1 \) is both the start state and the final state

\[
\begin{array}{c|cccc}
& a & b & c \\
\hline
s_1 & s_2 & s_3 & s_4 \\
s_2 & s_1 \\
s_3 & s_1 \\
s_4 & s_1 \\
\end{array}
\]

(d) \( s_1 \) is the start state; \( s_1 \) and \( s_3 \) are final states

\[
\begin{array}{c|ccc}
& a & b & c \\
\hline
s_1 & s_2 \\
s_2 & s_2 & s_3 \\
s_3 & \\
\end{array}
\]

2. Write DFA’s that recognize the following languages. Be careful to note starting and accepting states in your automata.

(a) \( \{ w \in \{a, b\}^* : \text{each } a \text{ in } w \text{ is followed by at least one } b \} \)

(b) \( \{ w \in \{a, b\}^* : \text{w contains precisely three } a's \} \)

(c) \( \{ w \in \{a, b\}^* : \text{w has an odd number of } a's \text{ and an even number of } b's \} \)

(d) \( \{ w \in \{a, b\}^* : \text{w has } bab \text{ as a substring } \} \)
3. Consider the following regular expression:

\[(abc^* \mid a^*bc)\]

(a) As described in class, construct an NFA for this expression.
(b) Using subset construction, convert the NFA into a DFA.
(c) Optimize the DFA to minimize the number of states.

4. Consider the grammar below:

\[
S ::= S ; S \\
    \mid <ID> = E \\
    \mid \text{print} ( L ) \\
E ::= E + E \\
    \mid ( S , E ) \\
    \mid <ID> \\
    \mid <NUM> \\
L ::= E \\
    \mid L , E
\]

(a) Write a grammar that accepts the same language as the grammar above, but that is suitable for LL(1) parsing.
(b) Find **FIRST** and **FOLLOW** sets for your grammar.
(c) Show the LL(1) parsing table for your grammar.
(d) Given the input string, “x = 5; print(x, 6)”, use your parse table to expand the goal into a parse for the expression. Note which grammar production is applied at each step.