Part 1: The "AND" Problems

These DFAs accept strings only if they meet **both** the starting and ending conditions.

1. Starts with "aba" AND ends with "ab"

- **Regular Expression**: aba(a|b)*ab

- **Solution :** The DFA first follows a strict path for the prefix "aba". Any other starting sequence goes to a non-accepting dead state (qD). After the prefix is confirmed, the machine enters a loop of states (q_3, q_4, q_5) that track the suffix. The machine only enters the final state q_5 if the string's last two characters are "ab".
- **Transition Table**:

State	On 'a'	On 'b'	Description
-> q_0	q1	qD	Start
q_1	qD	q2	Prefix "a"
q_2	q3	qD	Prefix "ab"
q_3	q3	q5	Starts "aba", ends "a"
q_4	q3	q4	Starts "aba", ends "b"
*** q_5**	q3	q4	Starts "aba", ends "ab" (Final)
qD	qD	qD	Dead State

---

2. Starts with "aba" AND ends with "aba"

- **Regular Expression**: aba | aba(a|b)*aba

- **Solution :** This DFA confirms the "aba" prefix, immediately landing in the final state q_3 (since "aba" is a valid string). From there, it enters a loop of states that track the current suffix. It will only return to the final state q_3 if the string once again ends in "aba", correctly handling overlaps like in "ababa".
- **Transition Table**:

State	On 'a'	On 'b'	Description
-> q_0	q1	qD	Start
q_1	qD	q2	Prefix "a"
q_2	q3	qD	Prefix "ab"
*** q_3**	q4	q5	Starts "aba", ends "aba" (Final)
q_4	q4	q5	Starts "aba", ends "a"
q_5	q3	q6	Starts "aba", ends "ab"
q_6	q4	q6	Starts "aba", ends "b"
q_D	qD	qD	Dead State

---

3. Starts with "aba" AND ends with "ba"

- **Regular Expression**: aba(a|b)*ba

- **Solution** : First, the machine verifies the "aba" prefix. Then, it uses a three-state group (q_3, q_4, q_5) to check the ending. State q_3 represents ending in "a", q_4 represents ending in "b", and the final state q_5 represents ending in "ba". The transitions move between these states based on the input to track the suffix correctly.
- **Transition Table**:

State	On 'a'	On 'b'	Description
-> q_0	q1	qD	Start
q_1	qD	q2	Prefix "a"
q_2	q3	qD	Prefix "ab"
q_3	q3	q4	Starts "aba", ends "a"
q_4	q5	q4	Starts "aba", ends "b"
*** q_5**	q3	q4	Starts "aba", ends "ba" (Final)
qD	qD	qD	Dead State

---

4. Starts with "aba" AND ends with "aa"

- **Regular Expression**: aba(a|b)*aa

- **Solution :** Like the others, this DFA first locks in the "aba" prefix. The subsequent states are designed to track the suffix, moving to the final state q_5 only when two consecutive 'a's are read after the initial prefix part is complete.
- **Transition Table**:

State	On 'a'	On 'b'	Description
-> q_0	q1	qD	Start

q_1	qD	q2	Prefix "a"
q_2	q3	qD	Prefix "ab"
q_3	q5	q4	Starts "aba", ends "a"
q_4	q3	q4	Starts "aba", ends "b"
*** q_5**	q5	q4	Starts "aba", ends "aa" (Final)
q_D	qD	qD	Dead State

---

5. Starts with "aba" AND ends with "bb"

- **Regular Expression**: aba(a|b)*bb

- **Solution :** This machine forces the "aba" prefix, then enters a small loop of states. The only purpose of this loop is to detect if the string ends with "bb", transitioning to the final state q_5 only when that specific condition is met.
- **Transition Table**:

State	On 'a'	On 'b'	Description
-> q_0	q1	qD	Start
q_1	qD	q2	Prefix "a"
q_2	q3	qD	Prefix "ab"
q_3	q3	q4	Starts "aba", ends "a"
q_4	q3	q5	Starts "aba", ends

			"b"
*** q_5**	q3	q5	Starts "aba", ends "bb" (Final)
qD	qD	qD	Dead State

Part 2: The "OR" Problems

These DFAs accept strings if they meet **at least one** of the two conditions.

6. Starts with "aba" OR ends with "ab"

- **Regular Expression**: aba(a|b)* | (a|b)*ab

- **Solution :** This DFA has two paths to victory. 🏆 First, it checks for the "aba" prefix. If found, it enters a final "sink" state qS and accepts the string no matter what follows. If the prefix is not "aba", the machine uses its other states to perform a second check: does the string end in "ab"? If so, it lands in a different final state (q_2 or qAB).
- **Transition Table**:

State	On 'a'	On 'b'	Description
-> q_0	q1	qB	Start
q_1	qA	q2	Prefix "a"
*** q_2**	qS	qB	Prefix "ab" (Final, ends "ab")
*** q**	qS	qS	Starts "aba" (Final Sink)

q_A	qA	qAB	Other, ends "a"
q_B	qA	qB	Other, ends "b"
*** q_{aB}**	qA	qB	Other, ends "ab" (Final)

---

7. Starts with "aba" OR ends with "aba"

- **Regular Expression**: aba(a|b)* | (a|b)*aba

- **Solution** : Because "aba" satisfies both conditions, the logic is unique. The DFA starts with a set of non-final states (q_0, q_1, q_2) that check if a string ends in "aba". As soon as the "aba" sequence is confirmed (either as a prefix or suffix), the machine jumps to a set of states that are *all* final (q_3, q_4, q_5, q_6). This guarantees that any string starting with "aba" is always accepted.
- **Transition Table**:

State	On 'a'	On 'b'	Description
-> q_0	q1	q0	Suffix "", not started "aba"
q_1	q1	q2	Suffix "a", not started "aba"
q_2	q3	q0	Suffix "ab", not started "aba"
*** q_3**	q4	q5	Ends "aba" (Final)
*** q_4**	q4	q5	Starts "aba", ends "a" (Final)
*** q_5**	q3	q6	Starts "aba", ends "ab" (Final)

*** q_6**	q4	q6	Starts "aba", ends "b" (Final)

---

8. Starts with "aba" OR ends with "ba"

- **Regular Expression**: aba(a|b)* | (a|b)*ba
-
- **Solution** : This DFA's strategy is to first check for the "aba" prefix. If it finds it, the string is accepted by moving to the final sink state qS. If the string starts differently, the machine uses its other states to check the second condition: does the string end in "ba"? If so, it moves to the other final state, qBA, to accept the string.
- **Transition Table**:

State	On 'a'	On 'b'	Description
-> q_0	q1	qB	Start
q_1	qA	q2	Prefix "a"
q_2	qS	qB	Prefix "ab"
*** q**	qS	qS	Starts "aba" (Final Sink)
q_A	qA	qB	Other, ends "a"
q_B	qBA	qB	Other, ends "b"
*** q_{Ba}**	qA	qB	Other, ends "ba" (Final)

---

9. Starts with "aba" OR ends with "aa"

- **Regular Expression**: aba(a|b)* | (a|b)*aa
-
- **Solution** : The machine has two ways to accept. It looks for the "aba" prefix, and if

successful, transitions to the final sink state qS for an automatic pass. For any other starting sequence, it relies on its backup plan: using the states qA and qAA to check if the string ends with "aa", which is the other winning condition.

- **Transition Table**:

State	On 'a'	On 'b'	Description
-> q_0	q1	qB	Start
q_1	qAA	q2	Prefix "a"
q_2	qS	qB	Prefix "ab"
*** q□**	qS	qS	Starts "aba" (Final Sink)
q_A	qAA	qB	Other, ends "a"
q_B	qA	qB	Other, ends "b"
*** q_{aa}**	qAA	qB	Ends "aa" (Final)

---

10. Starts with "aba" OR ends with "bb"

- **Regular Expression**: aba(a|b)* | (a|b)*bb
-
- **Solution :** This DFA works just like the previous "OR" examples. It has a primary goal of finding the "aba" prefix to enter the qS final state. If that fails, it executes its secondary function: checking if the string ends in "bb". If a 'b' is followed by another 'b', it transitions to the final state qBB to accept the string.
- **Transition Table**:

State	On 'a'	On 'b'	Description
-> q_0	q1	qB	Start
q_1	qA	q2	Prefix "a"

q_2	qS	qB	Prefix "ab"
*** q□**	qS	qS	Starts "aba" (Final Sink)
q_A	qA	qB	Other, ends "a"
q_B	qA	qBB	Other, ends "b"
*** q_{BB}**	qA	qBB	Ends "bb" (Final)