Deterministic Finite Automata

DFAs are computational machines which
- have finite memory,
- compute in "real-time" in that they fully process an input upon receiving it,
- and accept/reject their input string.

Visually, we can draw a DFA $M$ as

A DFA accepts a string $\omega$ if after processing all of its input it is in an accepting state. It rejects $\omega$ if it is in any other state.

Visually, we can draw a DFA $M$

$\Sigma = \{a, b, c\}$

What does this DFA do?

$L(M) = \{\omega \mid \omega \in \Sigma^* \text{ and starts with } a \}$

Give a DFA for the following languages with $\Sigma = \{a, b\}$
- $\emptyset$, $\Sigma$
- $\{xyzb \mid x, y, z \in \Sigma^*\}$
- $\{xabbay \mid x, y \in \Sigma^*\}$
- $\{w \in \Sigma^* \mid w \text{ has equal } a's \text{ and } b's\}$
- $\not{\text{not regular!}}$
5) No DFA can decide this. To know \( \#(a) = \#(b) \), you must track their difference. Since \( \#(a) - \#(b) \) is unbounded, this cannot be done with finite memory.
A *deterministic finite automaton* $M$ is a 5-tuple

\[ M = (Q, \Sigma, \delta, q_0, F) \]

where $Q$, $\Sigma$, $\delta$, $q_0$, and $F$ represent *states*, *alphabet*, *transition function*, *start state*, and *accepting states*, respectively.

Consider the DFA

\[ \begin{array}{c}
| q_1 \rightarrow & 0 & q_2 \rightarrow & 1 \rightarrow q_1 \\
| 0 & q_2 \rightarrow & a_1 \leftarrow & 1 \rightarrow q_2 \\
| q_1 \leftarrow & a_1 \rightarrow & 1 & q_1 \rightarrow \qquad \text{(cycle)} \\
\end{array} \]

What does it do?

- $Q = \{q_1, q_2\}$
- $\Sigma = \{0, 1\}$
- $\delta = \begin{array}{c|c}
0 & q_2 \\
a_1 & a_1 \\
1 & q_1 \\
\end{array}$
- $q_0 = q_1$
- $F = \{q_1, q_2\}$

**Def:** A language $A$ is **regular** if there is some DFA $M$ with $L(M) = A$. 

---

It appears that the handwritten notes for DFA 3A1 might be a mixture of lecture notes and possible homework concepts, but the main focus is on the formal definition and example of a DFA.
What is the DFA for $A \cap B = \{ \varepsilon \omega | \omega \in \Sigma^* \}$?