Defn: A deterministic finite automaton (DFA) is a 5-tuple $M = (Q, \Sigma, s, s', F)$ where:

- $Q$ is a finite set of states
- $\Sigma$ is an alphabet—that is, a finite set of char.
- $s$, a member of $Q$, is the start state ($q_0$ in text)
- $F$, a subset of $Q$, is the set of (desired) final states, also called the set of accepting states.
- $\delta$ is the transition function: $\delta: Q \times \Sigma \rightarrow Q$.

Class DFA {
    set<State> Q;
    set<char> Sigma;
    State s;
    set<State> F; //REQ: c is in Sigma
    State delta (State p, char c); //REQ: c is in Sigma
    State (*delta) (State p, char c);
    set<Triple<State, char, State>> delta;
}

This is a class method. We need a member method.

In C++ — (WR prefers: — ?);

This makes delta a member rather than a class-wide method so clearly it depends on an instance $M$. Triples are instructions
Visualization:
\( \Delta \) is a set of nodes.
\( \Sigma \) is a set of edges with labels in \( \Sigma \).

Example: Tell whether a given string \( X \) over \( \Sigma = \{0, 1\} \) has an odd number of 1s.
\( \Delta = \{ \text{even}, \text{odd} \} \)

Interpreta / INVariant:
Current state reflects the number of 1s seen so far.
Start state \( \delta = \text{even} \) since initially we have seen zero 1s and 0 is an even number.

If a DFA, the set \( \delta \) has exactly one state for each possible state of \( \{0, 1\} \).

The language \( L(\Delta) \) of this DFA \( \Delta \) equals \( \{ x \in \{0, 1\}^* : \#1(x) \text{ is odd} \} \).

Define: A computation by a DFA \( \Delta = (\Delta, \Sigma, \delta, \delta, F) \) is a sequence:
\[ \gamma = (q_0, X_1, q_1, X_2, \ldots, X_{n-1}, q_{n-1}, X_n, q_n) \]
where:
- \( n = |X| \) (the length of \( X \))
- \( X = X_1 \ldots X_n \)
- \( \delta \) is \( i \)th bit.
- \( q_0 = \delta \)
- each \( q_i \in \Delta \)
- for all \( i, \delta(q_i, X_i) = q_{i+1} \)

For all \( j, 1 \leq j \leq n \), \( (q_{j-1}, X_j, q_j) \in \delta \)

\( L(\Delta) = \{ x \in \{0, 1\}^* : \Delta \text{ has an accepting computation on input } x \} \)
OK to define DFA by their diagrams. Some simple cases.

\[ \Sigma = \{0, 1\} \]

\[ M_0 = \]

\[ \alpha = \{0, 1, 2\} \]

\[ F = \emptyset \]

\[ L(M_0) = \emptyset \]

\[ L(M_1) = \Sigma^* = \{0, 1\}^* \]

\[ F = \{0, 1, 2\} \]

Hence a DFA need not be "in lowest terms".

One More Example:

How about \#1(x) Mod 3?

\[ \#1(x) = 0 \equiv 0 \mod 3 \]

Start \(F \iff 0\) should be in \(L\).

\[ \Sigma = \{0, 1, 2\} \]

\[ F = \{0, 1, 2\} \]

\[ L(M) = \{x \in \{0, 1, 2\}^* : \text{sum of digits is a multiple of 3}\} \]

\[ M_2 = \]

\[ \text{Read } c = 0: \text{congruence stays same.} \]

\[ \text{Read } c = 1: \text{congruence up by 1 and 3 cycle clockwise.} \]

\[ \text{Read } c = 2: \text{cycle counterclockwise} \]

\[ 0, 3, 6, 9 \]

The way I defined \(L_3\) looks circular, but that's a problem only when \(x\) is a single digit, so we declare that 0, 3, 6, 9 \(L_3\) as the bases.

For other \(x\), it's well-founded.

Indeed, \(M_3\) is kind of the same as \(M\), but with a bigger alphabet.