\(\Sigma\): any alphabet.
* "zero or more" \(\Sigma^*\) = the set of all strings over \(\Sigma\).

**Defn:** A language \(A \subseteq \Sigma^*\) is **regular** if there is a DFA \(M = (Q, \Sigma, \delta, s, F)\) such that \(A = L(M)\).

Examples:
- \(\Sigma^2\) is regular: \(M = \begin{array}{c}
  Q = \{a, b\} \\
  \delta(a, a) = b, \delta(b, b) = a, \delta(a, b) = a, \delta(b, a) = b
\end{array}\)
- \(\Sigma^*\) is regular: \(M = \begin{array}{c}
  Q = \{a, b\} \\
  \delta(a, a) = a, \delta(b, b) = b, \delta(a, b) = b, \delta(b, a) = a
\end{array}\)
- \(\Sigma^0 = \{\varepsilon\}\) is regular: \(M = \begin{array}{c}
  Q = \{a, b\} \\
  \delta(a, a) = \varepsilon, \delta(b, b) = \varepsilon, \delta(a, b) = \varepsilon, \delta(b, a) = \varepsilon
\end{array}\)

**Theorem:** If \(A\) is regular then so is its **complement** \(\overline{A} = \Sigma^* \setminus A = \{x \in \Sigma^* : x \notin A\}\).

**Proof:** Take \(M = (Q, \Sigma, \delta, s, F)\) from the defn "\(A\) is regular.
Now build \(M' = (Q, \Sigma, \delta, s, Q \setminus F)\), This is a DFA, \(L(M') = \{x : M\text{ does not accept } x\} = \{x : x \notin A\} = \overline{A}\). \(\Box\)
Theorem: If $A$ and $B$ are regular languages, then $A \cap B$ is also a regular language.

Proof: Take DFAs $M_A$, $M_B$ s.t. $L(M_A) = A$ and $L(M_B) = B$. We have $M_A = (Q_A, \Sigma, \delta_A, s_A, F_A)$ (with $\Sigma$ the same set as $A$ and $B$) and $M_B = (Q_B, \Sigma, \delta_B, s_B, F_B)$ (where the alphabet is the same alphabet). Let $C = A \cap B$. Define $M_C = (Q_C, \Sigma, \delta_C, s_C, F_C, L_M_C)$.

Goal: Build $M_C = (Q_C, \Sigma, \delta_C, s_C, F_C)$ s.t. $L(M_C) = C$.

Define $Q_C = Q_A \times Q_B = \{ (p, q) : p \in Q_A \text{ and } q \in Q_B \}$ and $s_C = (s_A, s_B)$.

If we view $\delta$ as a function:

$$\delta_C((p, q), a) = \left( \delta_A(p, a), \delta_B(q, a) \right)$$

Another way: If $(p, q, r)$ is an instruction of $\delta_A$ viewed as a set of triples in $M_A$, and $(q, a, r')$ is similar in $M_B$, then $M_C$ has the same instruction $(p, q, a, r, r')$. Finally, build $F_C = \{ (p, q) : p \in F_A \text{ AND } q \in F_B \}$.

Support $D = A \cup B$. Then $D = \sim (A \cap B)$, but direct:

Let $M_D$ have $F_D = \{ (p, q) : p \in F_A \text{ OR } q \in F_B \}$.

What about $E = A \Delta \{ (A \cap B) \cup \Delta \}$? Use XYZ.
Added: Example. \( \Sigma = \{0, 1\} \)

\( A = \{ x \in \Sigma^* : \text{the number of 1s in } x \text{ is odd} \} \)

\( B = \{ x \in \Sigma^* : \text{the number of 0s in } x \text{ is a multiple of 3} \} \)

For \( A \cap B \) \( \text{odd}_0 \) is the only accepting state: \( F_c = \{ \text{odd}_0 \} \)

For \( A \cup B \) \( F_0 = \{ \text{odd}_0, \text{odd}_1, \text{odd}_2, \text{EOF} \} \)

For \( E = \text{ALB} \) \( F_E = F_c \Delta F_0 = \{ \text{odd}_1, \text{odd}_2, \text{EOF} \} \).

Usually the tandem machine doesn't come out so "geometrically nice."
The "Turing K's" program was intended to have menu choices for taking
out two user-designed DFAs, \( M_A \) and \( M_B \) and computing the machine \( M_C \).
The sticking point was: How to draw \( M_C \) in a nice way? This leads in to the
major area of Graph Drawing. We've never found Java code for it.

To answer a question from class, suppose we intersect three or more machines. Hey, for
any (prime) number \( p \) it is easy to build \( M_p \) with \( p \) states so that \( L(M_p) = \{ x \in \Sigma : \text{the number of 2s in } x \text{ is a multiple of } p \} \).
Consider \( L(M_2) \cap L(M_3) \cap L(M_5) \cap L(M_7) = L(M) \). A DFA \( M_E \) for the whole must have \( 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 = 2,310 \) states! This gets big quickly...