As a review, consider the first question this course aims to answer: What is computation? It involves three main steps:

1. Receive an input: a finite string over a finite alphabet \( \Sigma \), such as \( \Sigma = \{0, 1\} \)
2. Perform some steps
3. Produce an output: accept or reject

In this lecture, we look at one model of computation, the deterministic finite automaton.

**Topics Covered**

1. Deterministic Finite Automata
2. Regular Languages
3. Regular Operations

### 1 Deterministic Finite Automata

**Even/Odd Machine** As an example, consider the computational process of determining whether an input string over the alphabet \( \Sigma = \{0, 1\} \) has an even or odd number of zeroes. In particular, it will accept strings with an even number of zeroes, and reject strings with an odd number of zeroes. A machine that computes this is:

The *even* state represents the state in which we’ve seen an even number of zeroes so far. Similarly, the *odd* state is the case in which we’ve seen an odd number of zeroes so far. There’s some new notation here. The arrowhead pointing at the *even* state indicates that it’s the start state, and the double circles denote an accepting state. A single circle denotes a rejecting state. This machine is an example of a DFA.
• Formally, a **deterministic finite automaton (DFA)** consists of:

1. A finite set of states \( Q \)
2. A finite alphabet \( \Sigma \)
3. A transition function \( \delta : Q \times \Sigma \rightarrow Q \)
4. A unique start state \( q_0 \in Q \)
5. A set of accepting states \( F \subseteq Q \)

In our even/odd machine, these DFA components are:

1. \( Q = \{ \text{even, odd} \} \)
2. \( \Sigma = \{ 0, 1 \} \)
3. \( \delta = \{ \text{even} \times 1 \rightarrow \text{even}, \text{even} \times 0 \rightarrow \text{odd}, \text{odd} \times 1 \rightarrow \text{odd}, \text{odd} \times 0 \rightarrow \text{even} \} \)
4. \( q_0 = \text{even} \)
5. \( F = \{ \text{even} \} \)

How does our DFA operate on specific inputs? On input the empty string, \( \epsilon \), it starts in the \( \text{even} \) state and immediately accepts. On input 010, it starts in the \( \text{even} \) state, then moves to \( \text{odd} \), then \( \text{odd} \) again, and finally \( \text{even} \). Because it ends on an accepting state, the DFA accepts. This leads us to a more formal definition of what it means to accept an input.

• A **computation history (CH)** of a DFA \( M = (Q, \Sigma, \delta, q_0, F) \) on input \( w = w_1w_2 \cdots w_n \) is a sequence of states \( r_0, r_1, \ldots, r_n \) such that:

1. \( r_0 = q_0 \)
2. For \( 0 \leq i < n \), \( r_{i+1} = \delta(r_i, w_{i+1}) \)

• An **accepting CH** of \( M \) on input \( w \) is one where \( r_n \in F \).

• A DFA \( M = (Q, \Sigma, \delta, q_0, F) \) **accepts** a string \( w \) over \( \Sigma \) if its CH on input \( w \) is accepting. Else, it rejects \( w \).

## 2 Regular Languages

Languages are a way to denote the strings that a certain DFA or machine accepts.

• The **language** of a DFA \( M \) is \( L(M) = \{ w \mid M \text{ accepts } w \} \). We also say that \( M \text{ recognizes} \) the language \( L(M) \). Note that a language over \( \Sigma \) is any finite or infinite set of finite strings over \( \Sigma \).
Modulo Three Machine For another example of a DFA, consider the machine whose language $A$ over $\Sigma = \{0, 1, 2\}$ is $A = \{w \mid \text{the sum of digits of } w \text{ is a multiple of three}\}$. A DFA satisfying this spec is as follows:

In this DFA, $M$, the states correspond to the sum of digits seen so far, modulo three. That is, $q_0$ corresponds to a multiple of three, $q_1$ corresponds to one more than a multiple of three, and $q_2$ corresponds to two more than a multiple of three. In this example, we can again identify the five components of a DFA:

1. $Q = \{q_0, q_1, q_2\}$
2. $\Sigma = \{0, 1, 2\}$
3. $\delta(q_i, a) = q_{i+a \mod 3}$ for $0 \leq i \leq 2$
4. $q_0$
5. $F = \{q_0\}$

Proof of Correctness We can prove the correctness of $M$ by induction. Specifically, we can prove the lemma: $M$ recognizes the language $A$, or $L(M) = A$. It is sufficient to prove the claim: For all $j \geq 0$, after reading $j$ symbols of $w$, $M$ is in a state $q_u$ where $u = \Sigma_{i=1}^j w_i \mod 3$.

Proof of Claim For the base case of induction, let $j = 0$. At this point, we haven’t read anything and are still in the start state. The claim is true because after reading 0 symbols, $M$ is in state $q_0$. Our inductive hypothesis is that for $j > 0$, the claim is true for $j - 1$. For the inductive step, we assume by the inductive hypothesis that after reading the first $j - 1$ symbols of $w$, $M$ is in a state $q_u$ where $u = \Sigma_{i=1}^{j-1} w_i \mod 3$. Next, read the
symbol $w_j$, and by the transition function of $M$, we get to $\delta(q_u, w_j) = q_{u+w_j} = q_v$ where $v = \sum_{i=1}^{j-1} w_i + w_j \mod 3 = \sum_{i=1}^{j} w_i \mod 3$. In conjunction with the base case, this shows by induction on $j$ that the claim holds for all $j \geq 0$.

What kinds of languages are accepted by DFAs? We’ve seen two examples here: the even/odd language and the modulo-three language.

- A language $A$ is **regular** if there exists a DFA $M$ such that $A = L(M)$.

### 3 Regular Operations

What properties do regular languages have, aside from being recognized by DFAs? First, a few definitions.

- The **complement** of a language $L$ over $\Sigma$ is $L^C = \{ w \mid w \in \Sigma, w \notin L \}$.
- The **union** of two languages $A$ and $B$ is $A \cup B = \{ w \mid w \in A \text{ or } w \in B \}$.
- The **concatenation** of two languages $A$ and $B$ is $A \circ B = \{ w \mid w = x \circ y, x \in A, y \in B \}$.
- The **Kleene star** of a language $A$ is $A^* = \{ w \mid w = x_1 x_2 \cdots x_n \text{ where } n \geq 0 \text{ and every } x_i \in A \}$.

These last three operations—union, concatenation, and Kleene star—are known as **regular operations** on languages. The reason for this, as we will prove, is that regular languages are closed under the regular operations.

To see examples of these operations, consider the regular language $A = \{anna\}$ over the alphabet $\Sigma = \{a, n\}$. An *anna* machine that accepts this language $A$ is as follows:

![Diagram](image-url)
In addition, let the language $B = \{ w \mid w \text{ begins and ends with } n \}$. $B$ is a regular language, as we can construct a recognizing DFA:

![](image)

Given these regular languages $A$ and $B$, we can consider what happens when we perform regular operations on them. For example, $A \cup B = \{ w \mid w = \text{anna} \text{ or } w \text{ begins and ends with } n \}$. The concatenation $A \circ B = \{ \text{anna} \circ w \mid w \text{ begins and ends with } n \}$. On the language $A$, the Kleene star $A^* = \{ \epsilon, \text{anna,annaanna,annaannaanna,} \ldots \}$.

**Max’s Observation** For this language $B$, $B^* = \{ \epsilon \} \cup B$.

As an aside, note that $\emptyset$ is the empty language, and $\emptyset^* = \{ \epsilon \}$. Now that we have seen definitions and examples of complements and regular operations, we will begin to prove the closure of regular languages under these operations.

**Theorem** If $L$ is regular then so is $L^C$.

**Peter’s Proof** To show that a language is regular, we can construct a DFA that recognizes the language. Take $M$, a DFA recognizing $L$, where $M = (Q, \Sigma, \delta, q_0, F)$. Let $M' = (Q, \Sigma, \delta, q_0, F' = Q \setminus F)$. We can show that $M'$ is a DFA accepting $L^C$. Why does $M'$ accept this language? If $w$ is accepted by $M$, then there is a CH by $M$ on $w$ which ends on a state in $F$. When $M'$ reads the same input $w$, the CH ends in exactly the same state, which is now a rejecting state in $M'$. The opposite case follows by the same reasoning.

In particular, if $w$ is accepted by $M$, this implies that $w$ is rejected by $M'$, and if $w$ is rejected by $M$, this implies that $w$ is accepted by $M'$. Thus, $M'$ recognizes $L^C$, showing that it is a regular language.
Theorem  Regular languages are closed under union. This is Theorem 1.25 in Sipser’s book: If \( A \) and \( B \) are regular languages over the same alphabet, then so is \( C = A \cup B \). This can be generalized to languages over different alphabets, but we will assume the same alphabet for simplicity in the proof.

Proof Idea  Let \( M_A \) be a DFA recognizing \( A \), and let \( M_B \) be a DFA recognizing \( B \). Define \( M_A = (Q_A, \Sigma, \delta_A, q_{0A}, F_A) \) and \( M_B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B) \). From these, we will design a DFA \( M_C \) that runs \( M_A \) and \( M_B \) in parallel. In particular, \( M_C = (Q_C, \Sigma, \delta_C, q_{0C}, F_C) \) has the components:

1. \( Q_C = Q_A \times Q_B \)
2. \( \Sigma \) remains the same
3. \( \delta_C((q_A, q_B), a) = (\delta_A(q_A, a), \delta_B(q_B, b)) \)
4. \( q_{0C} = (q_{0A}, q_{0B}) \)
5. \( F_C = \{(q_A, q_B) \mid q_A \in F_A \text{ or } q_B \in F_B \} \)

For an example of a union DFA, we can consider the DFA that accepts the union of the even/odd and modulo-three languages seen earlier in class. We’ll add the element 2 to the even/odd alphabet to maintain consistency.