We first review the key points about regular languages. A language \( L \) is regular if \( L = L(M) \) for a DFA \( M \). Because any NFA \( N \) can be converted to an equivalent DFA, it follows that for any NFA \( N \), \( L(N) \) is regular. We also have a closure theorem, stating that if \( A \) and \( B \) are regular languages, then so are the languages \( A \cup B \), \( A \circ B \), and \( A^* \).

**Topics Covered**

1. Regular Expressions
2. Languages of Regexes
3. DFA-to-Regex Conversion

### 1 Regular Expressions

Consider a few small examples of regular languages: \( \{a\} \) for \( a \in \Sigma \), \( \{\epsilon\} \), and \( \emptyset \). We can clearly construct DFAs recognizing these languages, which implies that these languages are regular. Now consider applying several regular operations to the language \( \{1\} \), which we abbreviate as \( 1 \). For example, \((1 \cup 0)^*111(1 \cup 0)^*\) is regular by the closure of regular languages under union, concatenation, and Kleene star. Note that concatenation is implied in expressions like \( 111 = 1 \circ 1 \circ 1 \). This is an example of a regular expression, which we define inductively.

- A **regular expression** (**regex**) of size 1 over \( \Sigma \) is one of:
  1. \( a \), where \( a \in \Sigma \)
  2. \( \epsilon \), the empty string
  3. \( \emptyset \), the empty set

- A **regular expression** of size \( n > 1 \) is one of:
  4. \( R_1 \cup R_2 \), where \( R_1 \) and \( R_2 \) are regexes of size \( n_1 \) and \( n_2 \) such that \( n = n_1 + n_2 \)
  5. \( R_1 \circ R_2 \), where \( R_1 \) and \( R_2 \) are regexes of size \( n_1 \) and \( n_2 \) such that \( n = n_1 + n_2 \)
  6. \( R_1^* \), where \( R_1 \) is a regex of size \( n - 1 \)
In our example, \( R = (1 \cup 0)^*111(1 \cup 0)^* \) can be written as \( R = R_1 \circ R_2 \), where \( R_1 = (1 \cup 0)^* \) and \( R_2 = 111(1 \cup 0)^* \). Similarly, \( R_1 = R_3^* \) where \( R_3 = 1 \cup 0 \), and \( R_2 = 111 \circ R_3^* \). We can continue breaking down the regular expressions until we reach regexes of size 1.

In general, a regular expression defines a language. Two different regexes may describe the same language. For example, \( R, R \cup \emptyset \), and \( R \circ \epsilon \) all describe the same language. This is similar to the fact that different DFAs may recognize the same language.

- Let \( R \) be a regex over \( \Sigma \) and let \( a \in \Sigma \). The language of a regex \( L(R) \) is defined as follows:
  1. If \( R = a \), then \( L(R) = \{a\} \)
  2. If \( R = \epsilon \), then \( L(R) = \{\epsilon\} \)
  3. If \( R = \emptyset \), then \( L(R) = \emptyset \)
  4. If \( R = R_1 \cup R_2 \), then \( L(R) = L(R_1) \cup L(R_2) \)
  5. If \( R = R_1 \circ R_2 \), then \( L(R) = L(R_1) \circ L(R_2) \)
  6. If \( R = R_1^* \), then \( L(R) = (L(R_1))^* \)

There are a few edge cases to consider. If \( A \) is a language, then \( A \cup \{\epsilon\} \) is either \( A \) or \( \emptyset \) with the addition of the element \( \epsilon \). The language \( A \circ \{\epsilon\} \) is \( \emptyset \). The union \( A \cup \emptyset \) is \( A \), while the concatenation \( A \circ \emptyset \) is \( \emptyset \).

**Order of Operations** The order of operations on regular expressions is also important. The order of precedence is Kleene star, concatenation, then union. For example, we would write \( 01^* \cup 10^* \cup 00 \) as \( 0(1^*) \cup 1(0^*) \cup 00 \), which is equivalent to \( (0(1^*)) \cup (1(0^*)) \cup (00) \), and finally, \( (((0(1^*)) \cup (1(0^*)) \cup (00)) \).

**2 Languages of Regexes**

**Nathan’s As-Good-As-Planted Question** Is it a coincidence that our definition of the language of a regex looks a lot like the language of a DFA? Nope! We explain the relationship in the following theorem.

**Theorem** If \( R \) is a regex then \( L(R) \) is regular. That is, there is a DFA that recognizes \( L(R) \).

**Proof** We prove this theorem by induction. The base case is when \( R \) is a regex of size 1. Then \( R \) falls into one of three cases:
1. $R = a$ for $a \in \Sigma$. Then $L(R) = \{a\}$. Here is an NFA that recognizes $\{a\}$, which implies by the existence of an equivalent DFA that there exists a DFA that recognizes $\{a\}$:

![NFA diagram for $\{a\}$]

2. $R = \epsilon$. Then $L(R) = \{\epsilon\}$. An NFA accepting this language is as follows, implying that some DFA also accepts $\{\epsilon\}$:

![NFA diagram for $\{\epsilon\}$]

3. $R = \emptyset$. Then $L(R) = \emptyset$. An NFA accepting $\emptyset$ is the following, and implies the existence of a DFA recognizing $\emptyset$:

![NFA diagram for $\emptyset$]

Our (strong) inductive hypothesis is that for all $0 < k < n$ and a regex of size $k$, there exists a DFA that recognizes $L(R)$. For the inductive step, let $R$ be a regex of size $n$. There are three cases to consider:

4. $R = R_1 \cup R_2$. By the inductive hypothesis, there exist DFAs $M_1$ and $M_2$ that recognize $L(R_1)$ and $L(R_2)$, respectively. Because union is a regular operation, $L(R_1) \cup L(R_2)$, which is $L(R)$, is regular.

5. $R = R_1 \circ R_2$. By the inductive hypothesis, there exist DFAs that accept $L(R_1)$ and $L(R_2)$. Concatenation is a regular operation, so $L(R_1) \circ L(R_2)$, which is $L(R)$, is also regular.

6. $R = R_1^*$. As in the previous two cases, the inductive hypothesis implies that there exists a DFA $M_1$ accepting $L(R_1)$. Kleene star is a regular operation, so $(L(R_1))^*$ is regular. This is the language $L(R)$. $\blacksquare$

**Notation** There are a few notational shortcuts used to express regexes. In general, concatenation is implied, so $RR = R \circ R$. The notation $R^+$ denotes $RR^*$, and $R^k = R \circ R \circ \ldots \circ R \circ R$ ($k$ times). An alphabet $\Sigma$ is $0 \cup 1$ for binary strings, or more generally, $a_1 \cup a_2 \cup \ldots \cup a_k$ for $\Sigma = \{a_1, a_2, \ldots, a_k\}$. Finally, $\Sigma^*$ represents all finite strings over the alphabet $\Sigma$. 

Lecture 4: Regular Expressions
Max’s As-Good-As-Planted Question  Are there languages that are not regular? This is the topic of the next lecture, but the short answer is yes. As we will see, the language \( L = \{ a^k b^k \mid k > 1 \} \) is not regular.

3 DFA-to-Regex Conversion

Kalinda’s As-Good-As-Planted Question  Does every regular language have a regex? The answer is yes, and is considered in the following theorem.

Theorem  Let \( L \) be a regular language. Then there is a regex \( R \) such that \( L = L(R) \).

Proof  The proof of this theorem is constructive. We first consider a concrete example of a DFA, which we can convert to a regex. Suppose we have a DFA accepting binary strings over the alphabet \( \Sigma = \{0, 1\} \) such that the number of zeroes is even. One such DFA is:

\[
\begin{array}{c}
q_1 \\
1 \\
0 \\
q_2 \\
1
\end{array}
\]

Let’s convert this DFA to a regex. Our goal is to transform it into a DFA of the form:

\[
\begin{array}{c}
q_s \\
\text{regex } R \\
q_a
\end{array}
\]

Our first step is to add a new start state, \( q_s \), and a new accept state, \( q_a \). We add transitions from \( q_s \) and to \( q_a \) such that the input string is a regex:
To convert this to our final DFA, we need to reduce our machine down to two states. We’ll start by “ripping” out the state $q_1$, adjusting transitions between the other states to account for $q_1$’s transition behavior:

We can simplify the regexes by observing that taking the union of a regex $R$ with the empty set simply yields $R$, as does concatenating $R$ with the empty string:

The final step is to rip out $q_2$:
Thus, we have found a regex whose language consists of binary strings with an even number of zeroes: $1^* \cup 1^*0(1 \cup 01^*)^*01^*$. Along the way, we converted our original DFA into something not quite a DFA or NFA. In particular, we constructed a machine—called a generalized NFA—whose transitions are defined by regexes rather than single characters of the alphabet.

- A **generalized NFA (GNFA)** is a 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ where:
  1. $Q$ is a set of states
  2. $\Sigma$ is the input alphabet
  3. $\delta$ is defined for every pair of non-accepting states such that $\delta : (Q - q_{\text{accept}}) \times (Q - q_{\text{start}}) \rightarrow R$ where $R$ is a regular expression over $\Sigma$
  4. $q_{\text{start}} \in Q$ is a unique start state
  5. $q_{\text{accept}} \in Q$ is a unique accept state

- An **accepting computation history (CH)** of a GNFA on input $w$ consists of $r_0, \ldots, r_m$ and $y_1, \ldots, y_m$ such that:
  1. $r_0 = q_{\text{start}}$
  2. $y_i$ is in $L(\delta(r_{i-1}, r_i))$
  3. $r_m = q_{\text{accept}}$

In our example, if we let $w = 110011$, we start at $r_0 = q_s$, read in $y_1 = 110$, transition to $r_1 = q_2$, read in $y_2 = 011$, and transition to $r_2 = q_a$. To generalize our proof, we need to consider this procedure for any regular language and its recognizing DFA.

**General Proof** Let $M$ be a DFA recognizing the language $L$. We will construct an equivalent regex via the following steps:

1. First, convert $M$ to a GNFA $G$ by adding states $q_{\text{start}}$ and $q_{\text{accept}}$. Add a transition with input $\emptyset$ between all pairs of states without transitions in the DFA $M$. Note that the old accept states become regular states, although not indicated in the following sketch of this step. The old accept states transition to $q_{\text{accept}}$ on input $\epsilon$, and $q_{\text{start}}$ transitions to the old start state on input $\epsilon$. 

2. One by one, “rip” the states of $M$ out of $G$. To remove the state $q_{rip}$ from $G$, construct $G'$ such that:

(a) $Q' = Q - q_{rip}$

(b) $\Sigma$ remains the same

(c) $q_{accept}$ and $q_{start}$ remain the same

(d) $\delta'$ is defined such that for all $q_a, q_b$ that are different from $q_{rip}$, we update $\delta'(q_a, q_b) = \delta(q_a, q_b) \cup R_1 R_2^* R_3$, where $R_1 = \delta(q_a, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$, and $R_3 = \delta(q_{rip}, q_b)$.

3. For the final step, we output the one remaining regex, $\delta(q_{start}, q_{accept})$.

To prove correctness, we need to show that even as we rip out states, we preserve the language of the original DFA $M$. This proof is included in Section 1.3 of Sipser’s book.