As a recap of last lecture, recall that a deterministic finite automaton (DFA) consists of \((Q, \Sigma, \delta, q_0, F)\) where \(Q\) is a finite set of states, \(\Sigma\) is a finite alphabet, \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function, \(q_0 \in Q\) is the start state, and \(F \subseteq Q\) is the set of accepting states.

We also defined a computation history (CH) of a DFA \(M\) on input \(w = w_1 \ldots w_n\) as a sequence \(r_0, r_1, \ldots, r_n\) such that (1) \(r_0 = q_0\) of \(M\) and (2) for \(1 \leq i \leq n\), \(r_i = \delta(r_{i-1}, w_i)\) where \(\delta\) is \(M\)’s transition function. \(M\) accepts \(w\) if the CH ends in \(r_n \in F\); we also say that the language of \(M\) \(L(M) = \{w \mid M\) accepts \(w)\).

**Topics Covered**

1. Regular Operations
2. Nondeterministic Finite Automata
3. Equivalence of NFAs and DFAs

**1 Regular Operations**

Let \(A\) and \(B\) be regular languages. The following three operations are regular operations:

- The **union** of \(A\) and \(B\) is \(C = A \cup B = \{w \mid w \in A \text{ or } w \in B\}\)
- The **concatenation** is \(C = A \circ B = \{w \mid w = x \circ y \text{ such that } x \in A, y \in B\}\)
- The **Kleene star** of \(A\) is \(A^* = \{w \mid w = x_1 \ldots x_k \text{ such that } k \geq 0 \text{ and for all } 0 \leq i \leq k, x_i \in A\}\)

**The Closure Theorem** Regular languages are closed under union, concatenation, and Kleene star. That is, if \(A\) and \(B\) are regular languages, then so are \(A \cup B\), \(A \circ B\), \(A^*\), and \(B^*\).

Last time, we saw a proof idea for proving closure under union. What about concatenation? Here is what we wish to show: Let \(M_A\) be a DFA that recognizes language \(A\), and let \(M_B\) be a DFA that recognizes language \(B\). Then we can construct a DFA for \(C = A \circ B\).
For our first idea, we might try “jumping” between $M_A$ and $M_B$, so we read the first part of the input in $M_A$, reach an accept state, and then jump to the start state in $M_B$. The problem with this is that jumping from an accept state of $M_A$ to a start state of $M_B$ requires consuming one of the characters of the input string.

Alternatively, what if we take all transitions to accept states of $M_A$, and instead make them go directly to the start state of $M_B$? This is also problematic. For example, consider the languages $A = \{w \mid w \text{ consists of one or more 0s only} \}$ and $B = \{w \mid w \text{ consists of one or more 1s only} \}$. If we replace transitions to the accept state of $M_A$ with transitions to the start state of $M_B$ we get the following automaton $M_C$:

The language $L(M_C) = \{w \mid w \text{ has one 0, followed by one or more 1s} \}$. This machine would not accept the string “0011”, which is in $A \circ B$. For another idea, we might relax the notion of a finite automaton to allow for empty transitions. This is helpful because we can add empty transitions from the accept states of $M_A$ to the start state of $M_B$. Then we can jump between the machines without reading one of the input characters. With this addition, the automaton $M_C$ from the previous example would become:
Note that this new machine $M_C$ does accept “0011”. At state $a_1$, you can either read in a 0 and stay at $a_1$, or you can jump to $b_0$ without reading any new characters of the input string. If there exists some choice that leads to an accepting state, then we consider a string to be accepted. This leads us to the definition of a nondeterministic finite automaton.

2 Nondeterministic Finite Automata

- A **nondeterministic finite automaton** (NFA) consists of $(Q, \Sigma, \delta, q_0, F)$ where:

  1. $Q$ is a finite set of states
  2. $\Sigma$ is a finite alphabet
  3. $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$. For brevity of notation, we use $\Sigma_\epsilon$ to denote $\Sigma \cup \{\epsilon\}$. The term $\mathcal{P}(Q) = \{S \mid S \subseteq Q\}$ refers to the powerset of $Q$. This means that a state and input symbol can transition to some subset of the states of $Q$, rather than only a single state of $Q$.
  4. $q_0$ is a start state
  5. $F \subseteq Q$

Note that there are many similarities between the definitions of a DFA and NFA. While a DFA acts deterministically in response to a specific state and input symbol, an NFA may behave in one of many ways — nondeterministically. Nonetheless, there are other parallels between DFAs and NFAs, such as the notion of a computation history.

- A **computation history** (CH) of an NFA $N$ on input $w$ consists of $w_1 \cdots w_m$ and $r_0, r_1, \ldots, r_m$ such that:
1. \( w_i \in \Sigma \) for all \( i \) and \( w = w_1 \ldots w_m \)
2. \( r_0 = q_0 \)
3. for all \( 1 \leq i \leq m, \ r_i \in \delta(r_{i-1}, w_i) \)

For an example of an NFA, recall the language \( A = \{anna\} \) from the previous class. This is a regular language, as we can build a DFA to recognize \( A \). We can also build a machine to recognize \( B = \{w \mid w \text{ contains } anna\} \). Note that a string such as \( aannaa \) is in \( B \) but not in \( A \). Consider the following NFA, which recognizes \( B \):

This NFA doesn’t need a “doom” state (usually indicated in these notes with a sadface). In an NFA, it’s possible for a computation to simply become “stuck” and have no possible moves out of its current state. In this case, it will never reach an accept state.

As another example, let the language \( C = \{w \mid |w| \geq 3, \text{ and } w \text{ has a } 1 \text{ somewhere in the last three positions}\} \). An NFA for \( C \) looks like:

3 Equivalence of DFAs and NFAs

**Theorem** Let \( N \) be an NFA. Then there exists a DFA \( M \) such that \( L(M) = L(N) \).

**Corollary** The language \( L(N) \) is regular if \( N \) is an NFA.

Let’s consider as an example the NFA for the language \( C \), pictured above, to see how we might determine the strings that are accepted by an NFA. In this case, suppose the NFA runs on input “0110”. We can determine the possible states at each time step; given previous time steps, we then consider all possibilities for the next step. For this input, the possible states at each time step are:

1. Time 0: \( \{q_0\} \)
2. Time 1: \( \{q_0\} \)
3. Time 2: \( \{q_0, q_1, q_2, q_3\} \)
4. Time 3: \( \{q_0, q_1, q_2, q_3\} \)
5. Time 4: \( \{q_0, q_2, q_3\} \)

Observe that at Time 4, when the NFA is done reading the input string, an accept state \( q_3 \) is included in the set of possible states. This means that there exists an accepting computation history for “0110”, so it is in the language of the NFA.

**Construction of an Equivalent DFA** Suppose we have an NFA \( N = (Q_N, \Sigma, \delta_N, q_0N, F_N) \) and would like to construct a DFA \( M = (Q_M, \Sigma, \delta_M, q_0M, F_M) \) that recognizes the same language. We can construct \( M \) with the following components:

1. \( Q_M = \mathcal{P}(Q_N) \). In other words, the states of \( M \) are sets of states in \( Q_N \).
2. \( \Sigma \) remains the same
3. \( \delta_M(q, a) = \{ \text{the set of states of } N \text{ that are reachable from any state in the subset } q \in Q_M \subseteq Q_N \text{ when reading } a \in \Sigma \} \)
4. \( q_0M = \{ \text{the set of all states reachable from } q_0N \text{ by empty transitions} \} \)
5. \( F_M = \{ q \mid q \cap F_N \neq \emptyset \} \)

Is the start state of an NFA the same as the start state in the equivalent DFA? Not necessarily; as a counterexample, consider the following NFA \( N \):

Here is Anna’s “Anna Lysys”. The start state of \( N \) is \( r_1 \). However, the start state of an equivalent DFA \( M \) is the set of all states in \( N \) reachable from \( r_1 \) by an empty transition. This set includes \( r_1 \) and \( r_3 \). Thus, the start state of \( M \) is the state \( \{r_1, r_3\} \).
**The Closure Theorem**  Regular languages are closed under union, concatenation, and Kleene star. That is, if \( A \) and \( B \) are regular languages, then so are \( A \cup B \), \( A \circ B \), \( A^* \), and \( B^* \).

While we could prove this by constructing DFAs for the union, concatenation, and Kleene star of regular languages, we saw that we ran into some problems when trying to do this for concatenation. In particular, we introduced the notion of empty transitions to construct an automaton that recognized the concatenation of two regular languages. It turns out that we can instead construct NFAs to recognize these languages. Because any NFA has an equivalent DFA, a language recognized by an NFA is regular.

**Proof Idea for Union**  To build an NFA to recognize \( A \cup B \) for regular languages \( A \) and \( B \), we can construct components:

1. \( Q = \{q_0\} \cup Q_A \cup Q_B \)
2. \( \Sigma \) remains the same
3. \( \delta(q, a) = \{\delta_A(q, a)\} \) if \( q \in Q_A \) and \( a \in \Sigma \), or \( \{\delta_B(q, a)\} \) if \( q \in Q_B \) and \( a \in \Sigma \). We also add two transitions \( \delta(q_0, \epsilon) = \{q_0A\} \) and \( \delta(q_0, \epsilon) = \{q_0B\} \).
4. \( q_0 \) is the new start state
5. \( F = F_A \cup F_B \)
Proof Idea for Concatenation  To construct an NFA recognizing $A \circ B$, we use the components:

1. $Q = Q_A \cup Q_B$
2. $\Sigma$ remains the same
3. $\delta(q, a) = \{ \delta_A(q, a) \}$ if $q \in Q_A$ and $a \in \Sigma$, or $\{ \delta_B(q, a) \}$ if $q \in Q_B$ and $a \in \Sigma$. We also add empty transitions between every accept state in $A$ to $q_0B$. That is, $\delta(q, \epsilon) = \{ q_{0B} \}$ for every $q \in F_A$.
4. $q_0 = q_{0A}$
5. $F = F_B$

Proof Idea for Kleene Star  Given a regular language $A$, we can construct an NFA for $A^*$ with empty transitions from all accept states to a new start state $q_0$. We also add an empty transition from this new start state, which is also an accept state, to the old start state. This accounts for the fact that $\epsilon \in A^*$.