1 Characteristics of Nondeterministic Finite State Automata

- Every state of a deterministic finite state automaton (DFA) always has exactly one exiting transition arrow for each symbol in the alphabet. That is, in a DFA, the transition function is a true function. This means that there was never any choice about what to do at any step in the computation.
  
e.g., \( \delta(q_0, a) = q_1 \)

But in a nondeterministic finite state automaton (NFA), a state may have zero, one, or many exiting arrows for each alphabet symbol. That is, we can think of the transition function of NFA as returning not a unique state, but a set of states, including the empty set.
  
e.g., \( \delta(q_0, a) = \{q_0, q_1\} \)

- In a DFA, labels on the transition arrows are symbols from the alphabet.
  
An NFA can have arrows labeled with members of the alphabet, or the empty string \( (\epsilon) \).

2 Computation of NFA

Consider the machine below called \( N1 \).

```
q0 ----> 1 ----> q1 -----> 0, \epsilon ----> q2 -----> 1 ----> q3
```

2.1 Things to note

- Because the transition arrow from \( q1 \) to \( q2 \) is labeled with the empty string and a 0, the machine can move from state \( q1 \) to state \( q2 \) either by scanning 0 or simply by ‘jumping’ from \( q1 \) to \( q2 \).

- If the machine read a 1 in state \( q0 \), it can either remain in \( q0 \) or it can move to state \( q2 \).

- In state \( q1 \), there is no exiting arrow for 1, and in state \( q2 \), there is no exiting arrow for 0.
2.2 Example computation

Does the above machine $N_1$ accept 010110?

1. We start at $q_0$ and we read the first symbol in the string:
   
   \[ 0 \triangleleft 10110 \]

2. We remain in state $q_0$ and move to the next symbol:
   
   \[ 01 \triangleleft 0110 \]

3. We are now face with two options. We ‘copy’ the machine and let each copy carry
   on the computation.
   
   - copy-1: stays in $q_0$;
   - copy-2: moves to $q_1$.

4. An empty string exits state $q_1$, so the machine makes another copy.
   
   - copy-1: stays in $q_0$;
   - copy-2: stays in $q_1$;
   - copy-3: moves to $q_2$.

5. We move to the next symbol in the input string:
   
   \[ 010 \triangleleft 110 \]
   
   - copy-1: stays in $q_0$;
   - copy-2: moves to $q_2$;
   - copy-3: dies.

6. We now scan the fourth symbol in the string:
   
   \[ 0101 \triangleleft 10 \]
   
   This is ambiguous for copy-1, so it generates a new copy:
   
   - copy-1: stays in $q_0$;
   - copy-4: moves to $q_1$;
   - copy-2: moves to $q_3$.

7. An empty string exits state $q_1$, so copy-4 makes another copy.
   
   - copy-1: stays in $q_0$;
   - copy-4: stays in $q_1$;
   - copy-5: moves to $q_2$;
   - copy-2: stays in $q_3$.

8. Next we look at the fifth symbol in the string:
   
   \[ 01011 \triangleleft 0 \]
   
   This is ambiguous for copy-1, so it generates a new copy:
   
   - copy-1: stays in $q_0$;
   - copy-6: moves to $q_1$;
copy-4: dies;
copy-5: moves to $q_3$;
copy-2: stays in $q_3$.

9. An empty string exits state $q_1$, so copy-6 makes another copy.

  copy-1: stays in $q_0$;
copy-6: stays to $q_1$;
copy-7: moves to $q_2$;
copy-5: stays in $q_3$;
copy-2: stays in $q_3$.

10. Since copy-5 and copy-2 are in the same state, the two copies collapse into one.

  copy-1: stays in $q_0$;
copy-6: stays in $q_1$;
copy-7: stays in $q_2$;
copy-2: stays in $q_3$.

11. Now we scan the last symbol in the string:

    010110 $\langle$

  copy-1: stays in $q_0$;
copy-6: moves to $q_2$;
copy-7: dies;
copy-2: stays in $q_3$.

  We’ve reached the end of the string, so we check to see if any of the copies are in an accepting state. Since copy-2 is in $q_3$, which is an accepting state, we can conclude that the machine $N_1$ accepts the string.

Question: Which of the following strings does the above machine $N_1$ accept?

a. 1011  
b. 101010  
c. 000001  
d. 1100101

Question: What strings does the following NFA accept? Use set notation to describe the language it accepts.
3 Formal Definition of NFA

- Definition 1.17
  A nondeterministic finite state automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where:
  
  1. \(Q\) is a finite set of states;
  2. \(\Sigma\) is a finite set called the alphabet;
  3. \(\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q\) is the transition function;
  4. \(q_0 \in Q\) is the start state;
  5. \(F \subseteq Q\) is the set of accept states.

- We can describe \(N_1\) formally by writing \(N_1 = (Q, \Sigma, \delta, q_0, F)\), where
  
  1. \(Q = \{q_0, q_1, q_2, q_3\}\);
  2. \(\Sigma = \{0, 1\}\);
  3. \(\delta\) is defined as

      \[
      \begin{array}{c|ccc}
      \delta & 0 & 1 & \varepsilon \\
      \hline
      q_0 & \{q_0\} & \{q_0, q_1\} & \emptyset \\
      q_1 & \{q_2\} & \emptyset & \{q_2\} \\
      q_2 & \emptyset & \{q_3\} & \emptyset \\
      q_3 & \{q_3\} & \{q_3\} & \emptyset \\
      \end{array}
      \]

  4. \(q_0\) is the start state;
  5. \(F = \{q_3\}\).

- Question
  Give a formal description of the following machine \(N_2\).

      \[
      \begin{array}{c}
      q_0 \quad \xymatrix{ 0,1 & 1 & 0,1,\varepsilon & 0,1,\varepsilon & q_3 } \\
      \end{array}
      \]

  What strings does the machine \(N_2\) accept?

- Question
  Give a formal description of the following machine \(N_3\).
What strings does the machine $N3$ accept?

- Question
  
  Given the formal description of finite state automaton $N4$ below, draw a corresponding state diagram for $N$.

  $N4 = (Q, \Sigma, \delta, q0, F)$, where

  1. $Q = \{q0, q1, q2\}$;
  2. $\Sigma = \{a, b\}$;
  3. $\delta$ is defined as

        \[
        \begin{array}{c|ccc}
        \delta & a & b & \epsilon \\
        \hline
        q0 & \emptyset & \{q1\} & \{q2\} \\
        q1 & \{q1, q2\} & \{q2\} & \emptyset \\
        q2 & \{q0\} & \emptyset & \emptyset \\
        \end{array}
        \]

  4. $q0$ is the start state;
  5. $F = \{q0\}$.
4 Designing NFAs

- The language \( \{w | w \text{ contains the substring 0101} \} \) with five states. \( \Sigma = \{0, 1\} \).

- Question: The language \( \{w : \text{ the length of } w \text{ is at most 4} \} \). \( \Sigma = \{0, 1\} \).

- Question: The language \( \{w : w \text{ contains an odd number of 1’s} \} \). \( \Sigma = \{0, 1\} \).

- Question: The language \( \{\varepsilon, 101, 10001\} \). \( \Sigma = \{0, 1\} \).
• Notation: for any symbol $\sigma$ of the alphabet, the notation $\sigma^*$ means that we can have any number $n$ of consecutive $\sigma$, where $n \geq 0$ (i.e., we can also have no $\sigma$).

• Question: The language $0^*$. Diagram with one state.

• Question: The language $0^*1^*$ is. Diagram with two states.

• For recitation: Give a non-deterministic FSA diagram with the specified number of states for each of the following languages. All these languages have alphabet $\{0, 1\}$.
  
  a. The language $\{\epsilon\}$ with one state.
  b. The language $\{0\}$ with two states.
  c. The language $\{w : w \text{ ends with } 00\}$ with three states.
  d. The language $001^*$ with three states.
  e. The language $\{w : w \text{ starts with } 010\}$, with four states.
  f. The language $0^*1^*0$ with three states.
5 Equivalence of NFAs and DFAs

We say that two machines are equivalent if they recognize the same language.

Do NFAs accept different sets of languages from DFAs? That is, do NFAs which allow the transition function to return a set of states have more computational power than DFAs which require the transition function to return a single state?

No. NFAs and DFAs are equivalent.

5.1 Any DFA is also an NFA.

- We can convert any DFA \((Q, \Sigma, \delta, q_0, F)\) into a NFA \((Q, \Sigma, \delta', q_0, F)\) by changing every element of \(\delta\) of the form:
  \[\delta(a, a) = q_j\]
  into an element of \(\delta'\) of the form:
  \[\delta'(q_i, a) = \{q_j\}\].

- Let \(L(DFA)\) represent the class of languages recognized by the DFAs and let \(L(NFA)\) be the set of languages recognized by NFAs. Since any DFA is also an NFA, the following is clearly true.
  \[L(DFA) \subseteq L(NFA)\].

5.2 Any NFA has an equivalent DFA.

| Theorem 1.19 |
| Every nondeterministic finite state automaton has an equivalent deterministic finite state automaton. |

- If we prove Theorem 1.19, then we will have shown that the following holds:
  \[L(NFA) \subseteq L(DFA)\].

  Since we already know that
  \[L(DFA) \subseteq L(NFA)\]

  then we can conclude that
  \[L(DFA) = L(NFA)\]

- We will prove Theorem 1.19 by showing how to convert an NFA into an equivalent DFA that simulates the NFA.

  Recall that when you were doing the computation for a given string on an NFA, you kept track of a set of states from all the copies generated by the NFA. In other words, we had to keep track of a set of states in the NFA.
A state in the DFA that simulates an NFA is just a set of states corresponding to the state the copy machine would be in at any given moment during the computation. If \( k \) is the number of states of the NFA, it has \( 2^k \) subsets of states. Each subset corresponds to one of the possibilities that DFA must remember, so the DFA simulating the NFA will have \( 2^k \) states. That is, a state in our DFA should be an element of the power set of the set of states in the NFA.

- Here is how to convert an NFA into an equivalent DFA.

Let \( N = (Q, \Sigma, \delta, q_0, F) \) be the NFA recognizing some language \( A \). We construct a DFA, \( M = (Q', \Sigma, \delta', q_0', F') \), recognizing \( A \).

1. \( Q' = \varphi(Q) \).
   Every state of \( M \) is a set of states of \( N \).
2. For \( R \in Q' \) and \( a \in \Sigma \), let \( \delta'(R, a) = \{ q \in Q | q \in \delta(r, a) \text{ for some } r \in R \} \).
   In a more fancy way:
   \[
   \delta'(R, a) = \bigcup_{r \in R} \delta(r, a)
   \]
3. But wait! NFAs can make a transition on the empty string \( \epsilon \). For the proof to go through, we need to say what to do with \( \delta(q, \epsilon) \). To do so, we set up an extra bit of notation. For any state \( R \in M \), we define \( E(R) \) to be the collection of states that can be reached from \( R \) by going along \( \epsilon \) arrows, including the members of \( R \) themselves. Formally, for \( R \subseteq Q \) let
   \[
   E(R) = \{ q | q \text{ can be reached from } R \text{ by traveling along } 0 \text{ or more } \epsilon \text{ arrows} \}.
   \]
   Now we modify the definition of transition function of \( M \) as follows.
   \[
   \delta'(R, a) = \{ q \in Q | q \in E(\delta(r, a)) \text{ for some } r \in R \}.
   \]
4. \( q_0' = E(\{q_0\}) \).
5. \( F' = \{ R \in Q' | R \text{ contains an accept state of } N \} \).

- Example

Let \( N_4 = (Q, \Sigma, \delta, q_0, F) \) be defined as follows, which is a NFA.

1. \( Q = \{ q_0, q_1, q_2 \} \);
2. \( \Sigma = \{ a, b \} \);
3. \( \delta \) is defined as

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>a</th>
<th>b</th>
<th>( \epsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0</td>
<td>\emptyset</td>
<td>{q_1}</td>
<td>{q_2}</td>
</tr>
<tr>
<td>q_1</td>
<td>{q_1,q_2}</td>
<td>{q_2}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>q_2</td>
<td>{q_0}</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
4. \( q_0 \) is the start state;
5. \( F = \{ q_0 \} \).
We will construct an equivalent DFA, $M' = (Q', \Sigma, \delta', q_0', F')$.

1. $Q' = \varphi(Q) = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0,q_1\}, \{q_1,q_2\}, \{q_0,q_2\}, \{q_0,q_1,q_2\}\}$

2. $q_0' = \{q_0, q_2\}$

3. $\delta'$ is defined as:

   \[
   \begin{array}{|c|c|c|}
   \hline
   \delta' & a & b \\
   \hline
   \emptyset & \emptyset & \emptyset \\
   \{q_0\} & \emptyset & \{q_1\} \\
   \{q_1\} & \{q_1,q_2\} & \{q_2\} \\
   \{q_2\} & \{q_0,q_2\} & \emptyset \\
   \{q_0,q_1\} & \{q_1,q_2\} & \{q_1,q_2\} \\
   \{q_1,q_2\} & \{q_0,q_1,q_2\} & \{q_2\} \\
   \{q_0,q_2\} & \{q_0,q_2\} & \{q_1\} \\
   \{q_0,q_1,q_2\} & \{q_1,q_2,q_0\} & \{q_1,q_2\} \\
   \hline
   \end{array}
   \]

4. $F' = \{ \{q_0\}, \{q_0,q_1\}, \{q_0,q_2\}, \{q_0,q_1,q_2\} \}$

We can simplify this machine by observing that no arrows point at states $\{q_0\}$, and $\{q_0,q_1\}$, so they may be removed without affecting the performance of the machine.
• Corollary 1.20

Corollary 1.20
A language is regular if and only if some nondeterministic finite state automaton recognizes it.

One direction: A language is regular if some NFA recognize it.

Theorem 1.19 shows that any NFA can be converted into an equivalent DFA, so if an NFA recognizes some language, so does some DFA, and hence the language is regular.

The other direction: If a language is regular, some NFA recognizes it.

If a language is regular, then some DFA recognizes it. And any DFA is also an NFA.

• Corollary 1.20 is very convenient for two reasons:

1. It is often easier to design an NFA than a DFA, since the former usually requires fewer states. Once we find the right NFA, we can always build the corresponding DFA if we need to by simply following the proof for Theorem 1.20.

2. Some of our proofs will be simplified since we only need to show how to construct appropriate NFA rather than the more complex DFA.
6 Proof of Closure under Union

The class of regular languages are closed under the regular operations (union, intersection, and star operation).

Here we will prove this only for union, using nondeterministic finite state automata.

**Theorem 1.22**
The class of regular languages are closed under the union operation.

- Recall that this theorem means that if \( A \) and \( B \) are regular languages, then \( A \cup B \) are also regular languages.
- If \( A \) is a regular language, there is a NFA, call it \( N_A \) that accepts \( A \) and if \( B \) is a regular language, there is a NFA, call it \( N_B \) that accepts \( B \). Also, if \( A \cup B \) is a regular language, then there is a NFA, \( N_{A\cup B} \) that accepts \( A \cup B \).
- We will prove Theorem 1.22 by constructing a new NFA \( N_{A\cup B} \) by combining \( N_A \) and \( N_B \).

All we need to do is create a new start state and have two \( \epsilon \) transitions from this state, where one goes to \( N_A \) and the other goes to \( N_B \).

- Here is how to construct such a machine more formally.

Let \( N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) \) recognize \( A \), and \( N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) \) recognize \( B \).

Construct \( N = (Q, \Sigma, \delta, q_0, F) \) to recognize \( A \cup B \).

1. \( Q = \{q_0\} \cup Q_1 \cup Q_2 \).
2. \( q_0 \) is the start state of \( N \).
3. The accept states \( F = F_1 \cup F_2 \).
4. Define \( \delta \) such that for any \( q \in Q \) and any \( a \in \Sigma \),

\[
\delta(q, a) = \begin{cases} 
\delta_1(q, a) & \text{if } q \in Q_1 \\
\delta_2(q, a) & \text{if } q \in Q_2 \\
\{q_1, q_2\} & \text{if } q = q_0 \text{ and } a = \epsilon \\
\emptyset & \text{if } q = q_0 \text{ and } a \neq \epsilon 
\end{cases}
\]

- Example

\( L(N_1) = \{w \mid \text{The third symbol from the last symbol in } w \text{ is } 1 \} \)
\( L(N2) = \{ w \mid w \text{ contains } 1 \} \)

Construct a machine \( N3 \) that accepts \( L(N1) \cup L(N2) \).

Question: Give formal descriptions of \( N1 \) and \( N2 \). Then give a formal description of \( N3 \) using the proof for Theorem 1.22.