Strings and Languages:

Strings of characters are fundamental building blocks in CS. The alphabet over which the strings are defined may vary with the application. For our purpose, we define an alphabet to be any nonempty finite set. The members of the alphabet are the symbols of the alphabet. We usually use capital greek letters $\Sigma$ and $\Gamma$ to designate alphabets.

$\Sigma_1 = \{0, 1\}$

$\Sigma_2 = \{a, b, c, d, \ldots, x, y, z\}$

$\Gamma = \{0, 1, x, y, z\}$

A string over an alphabet is a finite sequence of symbols from that alphabet, usually written one next to the other and NOT separated by commas.

if $\Sigma_1 = \{0, 1\}$ then $01001$ is a string over $\Sigma_1$

if $\Sigma_2 = \{a, b, c, d, \ldots, x, y, z\}$ then $abracadabra$ is a string over $\Sigma_2$.

**Length of a string:**

If $w$ is a string over $\Sigma$, the length of $w$, written $|w|$, is the number of symbols that it contains.

The string of length zero is called the *empty string* and is written $\varepsilon$.

The empty string plays the role of 0 in a number system.
**Reverse of a String:**

If a string \( w \) has length \( n \), we can write \( w = w_1w_2\ldots w_n \), where each \( w_i \in \Sigma \). The reverse of \( w \), written \( w^R \), is the string obtained by writing \( w \) in opposite order, i.e. \( w^R = w_nw_{n-1}\ldots w_2w_1 \).

**Substring:**

String \( z \) is a substring of \( w \) if \( z \) appears consequentively within \( w \). For example, \( cad \) is a substring of abracadbra.

**Concatenation of strings:**

If we have a string \( x \) of length \( m \) and string \( y \) of length \( n \), the concatenation of \( x \) and \( y \), written \( xy \), is the string obtained by appending \( y \) to the end of \( x \), as in \( x_1x_2\ldots x_m y_1y_2\ldots y_n \).

To concatenate a string to itself many times, we use the superscript notation:

\[
\underbrace{x \ldots x}^k = x^k
\]

**Lexicographic Ordering of Strings:**

The lexicographic ordering of strings is an ordering in which shorter strings precede longer ones. Thus the lexicographic ordering of all strings over the alphabet
\{0, 1\} is
\(\varepsilon, 0, 1, 00, 01, 10, 11, 000,\ldots\).

**Computational Models:**

An idealized mathematical model of a computer. A computational model may be accurate in some ways but not in others. We shall be defining increasingly powerful models of computation.

We begin with the simplest model of computation called the **finite state machine** or **finite automaton**.

In the children’s game called Scissors-Paper-Stone, the two players select a member of the set \{SCISSORS, PAPER, STONE\} and indicate their selection with hand signals. If the two selections are the same, the game starts overs. If the selections differ, one player wins, according to the relation beats:
The directed graph to represent the relation *beats* is:

![Directed Graph](image)

**Finite Automata:**

The Finite State automaton (FSA):

- has a central processing unit of fixed, finite capacity.
- has no or an extremely limited amount of memory.
- receives its input as a string, it delivered to it on an input tape.
- delivers no output, except an indication of whether or not the input is acceptable.
Various electromechanical devices are built on the FSA models. In addition, FSA are applicable to the design of several types of algorithms and programs.

Case of automatic door controller:

Automatic doors swing open when sensing that a person is approaching. An automatic door has a pad in front to detect the presence of a person about to walk through the doorway. Another pad is located to the rear of the doorway so that the controller can hold the door open long enough for the person to pass all the way through.

The controller is in either of two states: “OPEN” or “CLOSED” representing the corresponding condition of the door.

<table>
<thead>
<tr>
<th>Input signal</th>
<th>States</th>
<th>Neither</th>
<th>Front</th>
<th>Rear</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Closed</td>
<td>Closed</td>
<td>Open</td>
<td>Closed</td>
<td>Closed</td>
</tr>
<tr>
<td></td>
<td>Open</td>
<td>Closed</td>
<td>Open</td>
<td>Open</td>
<td>Open</td>
</tr>
</tbody>
</table>
Other common devices have controllers with somewhat larger memories. In an elevator controller a state may represent the floor the elevator is on and the input is the signals it receives from the buttons. This computer might need several bits to keep track of this information.

Finite automata and their probabilistic counterpart Markov chains are useful tools when attempting to recognize patterns in data.
Let the following picture depict a finite automaton called M1:

![Finite Automaton Diagram]

1,01,11,0101,11001,00110

The figure above is called the state diagram of M1:
- It has three states, labeled q1, q2, and q3.
- The start state q1 is indicated by the arrow pointing at it from nowhere.
- The accept state q2 is represented with a double circle.
- The arrows going from one state to another are called transitions.

When this automaton receives an input string such as 1101, it processes that string and produces an output. The output is either accept or reject.

We will consider only this yes/no type of output. The processing begins in M1’s start state. The automaton receives the symbols from the input string one by one from left to right. After reading each symbol, M1 moves from one state to another along the transition that has that symbol as its label. When it reads the last symbol, M1 produces its output. The output is accept if M1 is now in an accept state and rejects if it is not.

What strings are accepted by the automaton?
Formal definition of a finite automaton:

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:
1. $Q$ is a finite set called the states
2. $\Sigma$ is a finite set called the alphabet
3. $\delta : Q \times \Sigma \rightarrow Q$ is the transition function
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept or final states

We can describe $M_1$ formally by writing $M_1= (Q, \Sigma, \delta, q_1, F)$, where
1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. $\delta$ is described as:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

4. $q_1$ is the start state, and
5. $F= \{q_2\}$

$\delta(q_1, 0) = q_1$ \hspace{1cm} $\delta(q_1, 1) = q_2$

$\delta(q_2, 0) = q_3$ \hspace{1cm} $\delta(q_2, 1) = q_2$

$\delta(q_3, 0) = q_2$ \hspace{1cm} $\delta(q_3, 1) = q_2$

$M= \{\{q_1, q_2, q_3\}, \{0, 1\}, \delta, q_1, \{q_2\}\}$
If \( A \) is the set of all strings that machine \( M \) accepts, we say that \( A \) is the **language of machine** \( M \) and write \( L(M) = A \) that \( M \) **recognizes** \( A \) or that \( M \) **accepts** \( A \).

A machine may accept several strings, but it always recognizes only one language. If a machine accepts no strings, it still recognizes only one language, namely the empty language \( \phi \).

\[ A = \{ w \mid w \text{ contains at least one } 1 \text{ and an even number of } 0\text{'s follow the last } 1 \} \]

**Notations for DFA’s**

There are two preferred notations for describing automata:

1. A transition diagram, which is a graph.
2. A transition table, which is a tabular listing of the \( \delta \) function.

**Transition Diagrams:**

A transition diagram for a DFA \( M = (Q, \Sigma, \delta, q_0, F) \) is a graph defined as follows:

1. For each state in \( Q \) there is a node.
2. For each state \( q \) in \( Q \) and each input symbol \( a \) in \( \Sigma \), let \( \delta(q,a) = p \) then the transition diagram has an arc from node \( q \) to node \( p \), labeled \( a \). If there are several input symbols that cause transitions from \( q \) to \( p \), then the transition diagram can have one arc, labeled by the list of these symbols.
3. There is an arrow into the start q0, labeled Start. This arrow does not originate at any node.
4. Nodes corresponding to accepting states (those in F) are marked by a double circle.
5. States not in F have a single circle.

**Transition Tables:**
1. A transition table is a conventional, tabular representation of a function like $\delta$ that takes two arguments and returns a value.
2. The rows of the table correspond to the states and the columns correspond to the inputs.
3. The entry for the row corresponding to state q and the column corresponding to input a is the state $\delta(q,a)$.

**Language of a DFA**

If A is the set of all strings that machine M accepts, we say that A is the **language of machine M** and write $L(M) = A$ that M recognizes A or that M accepts A.

A machine may accept several strings, but it always recognizes only one language.
If a machine accepts no strings, it still recognizes only one language, namely the empty language $\emptyset$.
The language A defined by the Machine M1 is:

$$A = \{w \mid w \text{ contains at least one } 1 \text{ and an even number of 0’s follow the last } 1\}$$

**Formal Definition of Computation**
Let $M= (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w=w_1w_2\ldots w_n$ be a string where each $w_i$ is a member of the alphabet $\Sigma$. Then $M$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ in $Q$ exists with three conditions:

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$
3. $r_n \in F$

**Extending the transition function to Strings:**

The DFA defines a language which is the set of all strings that result in a sequence of state transitions from the start state to an accepting state.

**Extended Transition function:**

If $\delta$ be the transition function, then the extended transition function is a function that takes a state $q$ and a string $w$ and returns the state $p$ -

The state that the automaton reaches when starting in state $q$ and processing the sequence of input $w$.

We define $\hat{\delta}$ by induction on the length of the input string, as follows:

**BASIS:** $\hat{\delta}(q, \varepsilon) = q$. That is, if we are in state $q$ and read no inputs, then we are still in state $q$.

**Induction:** Suppose $w$ is a string of the form $xa$; that is $a$ is the last symbol of $w$ and $x$ is the string consisting of all but the last symbol.

Then $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$
**Definition:**
A language is called a *regular language* if some finite automaton recognizes it.

The regular operations:

Let A and B be languages. We define the regular operation union, concatenation and star as follows:

- **Union**: \( A \cup B = \{x \mid x \in A \text{ or } x \in B\} \).
- **Concatenation**: \( A \circ B = \{xy \mid x \in A \text{ and } y \in B\} \)
- **Star**: \( A^* = \{x_1x_2...x_k \mid k>0 \text{ and each } x_i \in A\} \)

**Theorem**

The class of regular languages is closed under the union operation. That is if \( A_1 \) and \( A_2 \) are regular languages then \( A_1 \cup A_2 \) is a regular language.

Proof: See book page 46

Example 1.24 on worksheet:

**Theorem:**
The class of regular languages is closed under the concatenation operation.
Non determinism:

Deterministic computation:

So far, we have seen automata in which every step of a computation follows in a unique way from the preceding step. When the machine is in a given state and reads the next symbol, we know the next state it’ll be in, it is determined.
In a nondeterministic machine several choices exist for the next state at any point.

No determinism is a generalization of determinism, so every deterministic finite automaton is automatically a nondeterministic finite automaton.

Notation:
A deterministic finite state automaton is called DFA and a nondeterministic finite state automaton is called NFA.
Differences between a DFA and a NFA:

- Every state of a DFA has exactly one exiting transition arrow for each symbol of the alphabet.
- An NFA violates this rule.
- in a DFA, labels on the transition arrows are symbols from the alphabet.
- An NFA may have arrows labeled with members of the alphabet or \( \varepsilon \)
• Zero, one or many arrows may exit from each state with the label \( \varepsilon \)

Computation in an NFA:

Suppose we are running an NFA on an input string and come to a state with multiple ways to proceed. For example, say that we are in state \( q_1 \) in NFA \( N_1 \), and the next input symbol is 1. After reading that symbol, the machine splits into multiple copies of itself and follows \( \textit{all} \) the possibilities in parallel. Each copy of the machine takes one of the possible ways to proceed and continues as before. If there are subsequent choices, the machine splits again. If the next input symbol doesn’t appear on any of the arrows exiting the state occupied by a copy of the machine, that copy along with the branch of computation associated with it, die. Finally, if any one of these copies of the machine is in an accept state at the end of the input, the NFA accepts the input string. Nondeterminism can be viewed as a kind of parallel computation wherein multiple independent processes a or threads can be running concurrently. If at least one of these processes accepts, then the entire computation accepts.
Formal definition of a nondeterministic finite automaton:

The formal definition of a NFA differs from that of a DFA in one essential way: the type of transition function. In a DFA the transition function takes a state and an input symbol and produces the next state. In an NFA the transition function takes a state and an input symbol or the empty string and produces a set of possible states.

Definition:
We define the power set $P(Q)$ of set $Q$ to be the collection of all subsets of $Q$.

Definition:
For any alphabet $\sum$ we write $\sum_{\varepsilon} = \sum \cup \varepsilon$

Definition:
A nondeterministic finite automaton is a 5-tuple $(Q, \sum, \delta, q_0, F)$ where:
1. $Q$ is a finite set of states
2. $\sum$ is a finite alphabet
3. $\delta: Q \times \sum_{\varepsilon} \rightarrow P(Q)$ is the transition function
4. $q_0 \in Q$ is the start state and
5. $F \subseteq Q$ is the set of accept states
Equivalence of NFAs and DFAs:

Deterministic and nondeterministic finite automata recognize the same class of languages. We say that two machines are equivalent if they recognize the same language.

Theorem:
Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

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 the DFA must remember, so the DFA simulating the NFA will have $2^k$ states.

Proof:
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 $A$.

Case 1: N has no $\varepsilon$ arrows into account.

1. $Q' = P(Q)$. Every state of M is a set of states of N.
   (Recall that $P(Q)$ is the set of subsets of $Q$).

2. For $R \in Q'$ and $a \in \Sigma$ let $\delta'(R, a) = \{q \in Q | q \in \delta(r, a)$ for some $r \in R\}$
   if $R$ is a state of M, it is also a set of states of N.
When M reads a symbol a in a state R, it shows where a takes each state in R. Because each state may go to a set of states, we take the union of all these sets. Another way to write this expression is: $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$

3. $q_0' = \{q_0\}$ M starts in the state corresponding to the collection containing just the start state of N.

4. $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$. The machine M accepts if one of the possible states that N could be in at this point is an accept state.

Case 2: Now we consider the $\varepsilon$ arrows:
For any state R in M, Let $E(R)$ be the collection of states that can be reached from R by going only along $\varepsilon$ arrow, including the members of R themselves.
Formally, for $R \subseteq Q$ let $E(R) = \{q | q \text{ can be reached from } R \text{ by traveling along } 0 \text{ or more } \varepsilon \text{ arrows}\}$.
Thus we modify the transition function of M by placing additional states that can be reached by going along $\varepsilon$ arrows. Replacing $\delta'(r, a)$ by $E(\delta'(r, a))$ achieves this effect.
$\delta'(R, a) = \{q \in Q | q \in E(\delta'(r, a)) \text{ for some } r \in R\}$
The start state is modified by adding all possible states that can be reached from the start state of N along the $\varepsilon$ arrow
So $q_0' = E(\{q_0\})$
Example:
Let us illustrate the procedure we gave for converting an NFA into a DFA by using the machine N4 depicted below:

\[
N4=\left(\{1,2,3\}, \{a, b\}, \delta, 1, \{1\}\right)
\]

To construct a DFA that is equivalent to N4, we first determine D’s states, D’s states are \(P(Q)\) or power set of Q. N4 has three states, so we construct D with \(2^3 = 8\) states, one of each subset of N4’s states.

\[D’s=\{\emptyset, \{1\}, \{2\}, \{3\}, \{1,3\}, \{1,2\}, \{2,3\}, \{1,2,3\}\}\]

Next, we determine the start and accept states of D.

- The start state of N4 is \(E(\{1\})\), the set of states of that are reachable from 1 by traveling along \(\varepsilon\) arrows, plus 1 itself, so \(E(\{1\})=\{1,3\}\).
- The new accept states are those containing N4’s accept states; thus \(\{\{1\}, \{1,2\}, \{1,3\}, \{1,2,3\}\}\).
- Finally, we determine D’s transition function. Each of D’s states goes to one place on input a and one place on input b.
- In D,
- state {2} goes to {2,3} on input a, and to {3} on input b
- state {1} goes to $\emptyset$ on a, because no arrows exit it. It goes to {2} on b.
- state {3} goes to {1,3} on a, because in N4,
- state {3} goes to 1 on a and 1 in turn goes to 3 with an $\epsilon$ arrow.
- state {3} on b goes to $\emptyset$
- state {1,2} on a goes to {2,3} because 1 points at no states with a arrows and 2 points at states {2, 3}
- state {1,2} on b goes to {2,3}
- state {1,3} on a goes to {1,3}
- state {1,3} on b goes to {2}
- state {1,2,3} on a goes to {1,2,3}
- state {1,2,3} on b goes to {2,3}

We may simplify this machine by removing unreachable states such as {1} and {1, 2}.

Closure under the regular operations:

Theorem:
The class of regular languages is closed under the union operation

Theorem:
The class of regular languages is closed under the concatenation operation

Theorem: The class of regular languages is closed under the star operation.