These notes are intended mainly as a supplement to the lectures and textbooks; they will be useful for reminders about notation and terminology.

**Some basic notation and terminology**

An **alphabet** is a finite set of symbols. We use $A$ to denote an alphabet; often $\Sigma$ is used. A **word** over $A$ is defined to be a string of 0 or more members of $A$. The string consisting of exactly 0 members of $A$ is called the **empty string** and will be denoted by $\epsilon$. The empty string is also commonly denoted by the symbol 1 or $\lambda$.

The set of all words over $A$ will be denoted by $A^*$. It is sometimes convenient to exclude the empty string and define $A^+$ to be the set given by

$$A^+ = A^* \setminus \{\epsilon\}.$$  

**Example.** If $A = \{a\}$ then

$$A^* = \{\epsilon, a, aa, aaa, \ldots\}$$  

and $A^+ = \{a, aa, aaa, \ldots\}$

**Example.** If $A = \{0, 1\}$ then

$$A^* = \{\epsilon, 0, 00, 000, 0000, 00000, 000000, \ldots\}.$$  

Note that although the set $A$ is finite the set $A^*$ is infinite, as long as $A \neq \emptyset$.

If $w_1 = a_1 a_2 \ldots a_n$ (where $a_1, a_2, \ldots, a_n \in A$) and $w_2 = b_1 b_2 \ldots b_n$ (where $b_1, b_2, \ldots, b_n \in A$) are words over an alphabet $A$ then their **product** or **concatenation** is the word $w_1 w_2$ defined by

$$w_1 w_2 = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_n.$$  

We also define $\epsilon w = w = w\epsilon$ for all $w \in A^*$.

The product of $n$ copies of the word $w$ will be denoted by $w^n$, so for example $w^0 = \epsilon$, $w^1 = w$, $w^3 = www$.

It is easy to verify that this product is associative, i.e. $(uv)w = u(vw)$ for all $u, v, w \in A^*$.

The length of a word $w \in A^*$ is defined to be the number of letters it contains and will be denoted by $|w|$. In particular we have:

$$|\epsilon| = 0, \quad |a| = 1 \text{ if } a \in A, \quad |a_1 a_2 \ldots a_n| = n \text{ if } a_1, a_2, \ldots, a_n \in A.$$  

$|w_1 w_2| = |w_1| + |w_2|$, for all $w_1, w_2 \in A^*$.

Let $u, v, w$ be words over $A$. If $w = uv$ then we say that $u$ is a **prefix** of $w$ and $v$ is a **suffix** of $w$. Note that the empty word is a prefix and a suffix of every word. Also each word is a prefix and a suffix of itself. A prefix or suffix of $w$ will be called **proper** if it is neither $\epsilon$ nor $w$.

A **language** (or **formal language**) over an alphabet $A$ is defined to be a subset of $A^*$.

Let $A$ be an alphabet. If $L_1, L_2 \subseteq A^*$ then the **product** $L_1 L_2$ is defined by

$$L_1 L_2 = \{w_1 w_2 : w_1 \in L_1 \text{ and } w_2 \in L_2\}.$$  

If $L \subseteq A^*$ then the **Kleene closure** of $L$ is the set $L^*$ defined by

$$L^* = \{w_1 w_2 \ldots w_n : n \geq 0 \text{ and } w_1, w_2, \ldots, w_n \in L\}.$$  

If we define $L^n$ by $L^0 = \{\epsilon\}$, $L^{n+1} = L^* L$ for $n \geq 0$, then $L^* = \cup_{n=0}^{\infty} L^n$.

**Examples.** Let $A = \{a, b, c, d\}$. Then

$$\{a\} \{b\} = \{ab\}$$  

$$\{a, b\} \{c, d\} = \{ac, ad, bc, bd\}$$  

$$\{a\}^* = \{\epsilon, a, a^2, a^3, \ldots\}$$  

$$\{c, d\}^* = \{\epsilon, c, d, c^2, d^2, cd, dc, c^3, d^3, \ldots\}$$  

$$\{a\}^* \cup \{b\}^* = \{\epsilon, a, b, aa, bb, a^3, b^3, \ldots\}.$$  

A **regular expression** is an expression constructed from union, product and closure, for example

$$\{a\}^* \cup \{b\}^* = \{\epsilon, a, b, aa, bb, a^3, b^3, \ldots\}.$$  

A **regular language** is a language that can be represented using a regular expression.

**Deterministic Finite Automata**

A deterministic finite automaton is a mechanism for recognising words in a language; a simplified model of computation in which an input word is scanned left-to-right. The reading of each letter causes some change to take place in the machine. When all letters have been read the final state is used to determine whether the string has been accepted or not. This can be viewed as a simple form of output with just two possibilities “acceptance” or “rejection.”
A deterministic finite automaton (or DFA) is a quintuple $A = (Q, A, \phi, i, T)$ where

1. $Q$ is a finite nonempty set whose members are called states of the automaton;
2. $A$ is a finite nonempty set called the alphabet of the automaton;
3. $\phi$ is a map from $Q \times A$ to $Q$ called the transition function of the automaton;
4. $i$ is a member of $Q$ and is called the initial state;
5. $T$ is a nonempty subset of $Q$ whose members are called terminal states or accepting states.

The map $\phi$ describes the change of state when a single letter is read in — if the automaton is in state $q$ reads the letter $a$ then its state changes to $\phi(q, a)$. Initially the state is $i$ and if the input word is $w = a_1a_2 \ldots a_n$ then, as each letter is read, the state changes and we get $q_1, q_2, \ldots, q_n$ defined by

$$q_1 = \phi(i, a_1)$$
$$q_2 = \phi(q_1, a_2)$$
$$q_3 = \phi(q_2, a_3)$$
$$\vdots$$
$$q_n = \phi(q_{n-1}, a_n)$$

It is useful to extend the definition of $\phi$ so that its second argument can be a word instead of just a letter and we can write $q_n = \phi(i, w)$. This is done as follows. For a deterministic finite automaton defined using the above notation, extend the map $\phi : Q \times A \to Q$ to $\phi : Q \times A^* \to Q$ by defining:

$$\phi(q, \epsilon) = q \quad \text{for all } q \in Q$$
$$\phi(q, wa) = \phi(\phi(q, w), a) \quad \text{for all } q \in Q, w \in A^*; \ a \in A.$$

It is easy to show by induction that this extended map satisfies

$$\phi(q, vw) = \phi(\phi(q, v), w) \quad \text{for all } q \in Q; \ v, w \in A^*.$$

We say that a word $w \in A^*$ is accepted (or recognised) by the automaton $A$ if $\phi(i, w) \in T$, otherwise it is said to be rejected. The set of all words accepted by $A$ is called the language accepted by $A$ and will be denoted by $L(A)$. Thus

$$L(A) = \{ w \in A^* : \phi(i, w) \in T \}.$$

If $w = a_1a_2 \ldots a_n \in A^*$ and $\phi(p, w) = q$ then there are states $r_1, r_2, \ldots, r_{n-1}$ with

$$r_1 = \phi(p, a_1), \ r_2 = \phi(r_1, a_2), \ r_3 = \phi(r_2, a_3), \ldots, \ q = \phi(r_{n-1}, a_n).$$

We say that the states $p$ and $q$ are connected by a path with label $w$ and write

$$p \xrightarrow{a_1} r_1 \xrightarrow{a_2} r_2 \xrightarrow{a_3} \ldots \xrightarrow{a_{n-2}} r_{n-1} \xrightarrow{a_{n-1}} q \quad \text{or} \quad p \xrightarrow{w} q.$$

In particular a word $w$ is accepted by an automaton if and only if there is a path from the initial state to a terminal state with label $w$ (which we call an accepting path).

Example. Let $A$ be the automaton $(Q, A, \phi, i, T)$ where $Q = \{ i, t, r \}$, $A = \{ 0, 1 \}$, $T = \{ t \}$ and the transition function $\phi$ is given by

$$\phi(i, 0) = r, \ \phi(i, 1) = t,$$
$$\phi(t, 0) = t, \ \phi(t, 1) = t,$$
$$\phi(r, 0) = r, \ \phi(r, 1) = r.$$

It is simpler to describe a transition function by a table of values. In this example we have:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$0$</th>
<th>$1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
</tr>
<tr>
<td>$r$</td>
<td>$r$</td>
<td>$r$</td>
</tr>
</tbody>
</table>

If for example the word 0011 is input to this automaton then we obtain the following path:

$$i \xrightarrow{0} r \xrightarrow{0} r \xrightarrow{1} r \xrightarrow{1} t$$

so this word is rejected. The word 100 produces the following path

$$i \xrightarrow{1} t \xrightarrow{0} t \xrightarrow{0} t$$

and so is accepted.

An automaton can also be defined by giving its state diagram (which is a directed graph with labelled edges). The vertices of the graph are the states of the automaton. Two states $p$ and $q$ are joined by an edge labelled with a letter $a$ if and only if $\phi(p, a) = q$ where $\phi$ is the transition function. Multiple edges are usually combined and loops are possible.

The initial state is indicated by an arrow pointing to it, terminal states by arrows pointing away (or sometimes other symbols).

Example. The automaton of the last example has the following state diagram:
Problem. Find the language accepted by the automaton with the above state diagram.

Solution. After the first letter of an input word is read the automaton enters one of the states \( r, t \) and remains there no matter what the other letters are. Thus the single accepting state \( t \) can be entered if and only if the first letter read is 1. So the language accepted by \( A \) is the set of bit strings which begin with 1, i.e.

\[
L(A) = \{1w : w \in \{0,1\}^*\}.
\]

Examples of Deterministic Finite Automata

Problem. Construct a deterministic finite automaton accepting the language of all bit strings with three consecutive 0’s.

Solution. A sequence of states can be used to count the number of consecutive zeroes read in. If three are read we should reach an accepting state and remain there no matter what letters follow. If a 1 is encountered before the accepting state is reached then we return to the initial state and start counting zeroes again. A suitable automaton can therefore be described by the following state diagram.

Problem. Find the language accepted by the deterministic finite automaton defined by the following state diagram:

Solution. Note that the initial state is also a terminal state — this implied that the empty string \( \epsilon \) will be accepted. Any input string consisting of only \( a \)'s will leave the automaton in the initial state and so be accepted. Also from the initial state any string consisting of only \( b \)'s will lead to an accepting state. This suggests that the language accepted is

\[
L = \{a^mb^n : m, n \geq 0\}.
\]

Clearly any word in \( L \) is accepted. We need to show that there are no more words the automaton accepts. A word which is not in \( L \) will have \( ba \) as a substring and will therefore be rejected since in each of the three states the input \( ba \) leads to the right-hand state.

The language \( L \) can be written more compactly using Kleene closures since \( \{w^n : n \geq 0\} = \{w\}^* \) for any \( w \). We have

\[
L = \{a\}^*\{b\}^*.
\]

Problem. Give a state diagram for a deterministic finite automaton with alphabet \( \{a, b\} \) which accepts the language of all words with an odd number of \( a \)'s and an even number of \( b \)'s.

Solution. A suitable machine can be constructed using states which record whether odd or even numbers of each letter have been read at each step. Let the set of states of the automaton be

\[
\{\text{odd/odd}, \text{odd/even}, \text{even/odd}, \text{even/even}\},
\]

where each is interpreted as giving the parity of the number of \( a \)'s followed by the parity of the number of \( b \)'s read so far. It is now easy to give a table of values for the transition function:

\[
\begin{array}{ccc}
\text{odd/odd} & \text{even/even} & \text{odd/odd} \\
\text{odd/even} & \text{even/even} & \text{odd/odd} \\
\text{even/odd} & \text{even/odd} & \text{odd/odd} \\
\text{even/even} & \text{odd/even} & \text{even/even} \\
\end{array}
\]

For the set of terminal (i.e. accepting) states we take \( \{\text{odd/even}\} \). Initially no \( a \)'s and no \( b \)'s have been read so we have an even number of each and can take even/even as the initial state. A suitable state diagram is therefore:
Incomplete transition functions

We would like to be able to say that the following diagram (left hand side) shows a DFA that just accepts words “cat” and “dog”. But so far, our notion of “valid DFA” requires it to show what happens for every state/symbol combination. Using alphabet \( \{a, c, d, g, o, t\} \) we have to “complete” it to get the right-hand diagram.

We start out by extending our definition of DFAs in a way that makes no difference to the kinds of language you can describe, but sometimes simplifies the description of the DFA. We define an “incomplete” DFA \((Q, A, φ, i, T)\) by allowing \(φ\) to be a partial function from \((Q \times A)\) to \(Q\). Hence \(φ(q, a)\) may be undefined for some \((q, a)\) pairs, and if that happens during a computation we regard the entire string as having been rejected.

The reason why this does not change things is that we may always “complete” an incomplete DFA by adding a new rejecting state which the completed machine enters when no transition is defined for the original machine. An incomplete DFA is sometimes easier to define or depict.

Nondeterministic Finite Automata

The machines we have looked at so far are called deterministic because when a letter is read in, the next state is determined by the transition function. Suppose we widen the definition to allow more than one possible new state when a letter is read in. So \(φ(q, a)\) is a subset of \(Q\), and incompleteness can be included by allowing that \(φ(q, a)\) may be the empty set \(∅\).

The rule for accepting a string is that where there is a choice of new state, there should exist a new state that leads to acceptance. (In general, there should exist a sequence of choices.) This “non-determinism” can simplify the task of constructing complex machines.

Let \(P(S)\) denote the set of all subsets of a set \(S\). A nondeterministic finite automaton (or NFA) is a quintuple \(A = (Q, A, φ, i, T)\) where

1. \(Q\) is a finite nonempty set whose members are called states of the automaton;
2. \(A\) is a finite nonempty set called the alphabet of the automaton;
3. \(φ\) is a map from \(Q \times A\) to \(P(Q)\) called the transition function of the automaton;
4. \(i\) is a member of \(Q\) and is called the initial state;
5. \(T\) is a nonempty subset of \(Q\) whose members are called terminal states or accepting states.

The map \(φ\) can be extended to a map \(Q \times A^* \rightarrow P(Q)\) by defining

\[
φ(q, ϵ) = \{q\} \quad \text{for all } q ∈ Q
\]

\[
φ(q, wa) = \bigcup_{p ∈ φ(q, w)} φ(p, a) \quad \text{for all } q ∈ Q; w ∈ A^*; a ∈ A.
\]

Thus \(φ(q, w)\) is the set of all possible states that can arise when the input \(w\) is received in the state \(q\). As before we then have

\[
φ(q, vw) = φ(φ(q, v), w) \quad \text{for all } q ∈ Q; v, w ∈ A^*.
\]
We say that \( w \in A^* \) is accepted by \( A \) if \( \phi(i, w) \cap T \neq \emptyset \), i.e. if the set \( \phi(i, w) \) of all possible states arising from the input \( w \) contains at least one terminal state.

**Problem.** Write down the table of values of the transition function of the nondeterministic finite automaton with the following state diagram. Give the paths labelled by the input words \( ab \) and \( aba \) and decide if these are accepted or rejected. Find the language accepted by the automaton.

\[
\begin{array}{c|cc}
\phi & a & b \\
\hline
i & \{p, q\} & \{i, s\} \\
p & \emptyset & \{i, s\} \\
q & \{i\} & \emptyset \\
s & \emptyset & \emptyset \\
t & \emptyset & \emptyset \\
\end{array}
\]

**Solution.** The input word \( ab \) produces the following path
\[
\{i\} \xrightarrow{a} \{p, q\} \xrightarrow{b} \{i, s\}
\]
and is accepted since the final set contains a terminal state. The input word \( aba \) produces the following path
\[
\{i\} \xrightarrow{a} \{p, q\} \xrightarrow{b} \{i, s\} \xrightarrow{a} \{p, q\}
\]
and is rejected since the final set contains no terminal state.

The terminal state \( t \) can only be reached if the input word is one of \( aa, abaa, ababaa, \ldots \), while the terminal state \( s \) can only be reached if the input is one of \( ab, abab, ababab, \ldots \). Hence the language accepted by this automaton is
\[
\{ab\}^* \{a^2 \} \cup \{ab\}^+
\]
where \( L^+ \) denotes \( L^* \setminus \epsilon \).

We now show that introducing nondeterminism does not affect the languages which are accepted by our automata.

**Theorem.** A language which is accepted by a nondeterministic finite automaton is also accepted by some deterministic finite automaton.

**Proof.** Let \( A = (Q, A, \phi, i, T) \) be a nondeterministic finite automaton. We define a deterministic finite automaton \( A' \) as follows. The alphabet of \( A' \) is the same as that of \( A \). The set of states of \( A' \) is \( \mathcal{P}(Q) \). The transition function of \( A' \) is the map \( \psi: \mathcal{P}(Q) \times A \to \mathcal{P}(Q) \) defined by
\[
\psi(p, a) = \bigcup_{p \in P} \phi(p, a).
\]

For the initial state of \( A' \) we take \( \{i\} \in \mathcal{P}(Q) \). We now have
\[
w \in L(A) \iff \phi(i, w) \cap T \neq \emptyset \iff \psi(\{i\}, w) \cap T \neq \emptyset.
\]
Hence defining \( T' \) by
\[
T' = \{ P \in \mathcal{P}(Q) : P \cap T \neq \emptyset \}
\]
we have now specified a deterministic automaton \( A' = (\mathcal{P}(Q), A, \psi, \{i\}, T') \) and
\[
w \in L(A') \iff \psi(\{i\}, w) \in T' \iff \psi(\{i\}, w) \cap T \neq \emptyset \iff w \in L(A).
\]
Thus
\[
L(A) = L(A').
\]

Often it is simpler to define a nondeterministic automaton to perform a given task. We look at the example of searching for a pattern in a string of text.

**Problem.** Design a finite state automaton which accepts the language
\[
\{0, 1\}^* \{0101\} \{0, 1\}^*
\]
of all bit strings containing the substring 0101.

**Solution.** The language is accepted by the nondeterministic automaton with the following state diagram: If an input string does contain 0101 then in one possible computation the machine loops around the start state until 0101 is reached when it proceeds to the terminal state. Thus the input is accepted. If the input does not contain 0101 then there is no computation leading from the initial to the terminal state so the input is rejected.

Note that this nondeterministic automaton has a simpler state diagram than the following deterministic one which searched for the same substring.
Limitations of Finite Automata

If \( w \in A^* \) for some alphabet \( A \) and \( w = a_1a_2\ldots a_n \) where \( a_1, a_2, \ldots, a_n \in A \), then the reverse of \( w \) is the word \( w^R \) given by \( w^R = a_n \ldots a_2a_1 \). We also define \( \emptyset^R = \emptyset \).

If \( w = w^R \) then the word \( w \) is called a palindrome.

**Examples.** Let \( A = \{a, b, c, \ldots, x, y, z\} \). Then \( (\text{computer})^R = \text{retupmach} \). The following words are palindromes in \( A^* \): \( \text{bob, deed, rotator} \).

We now show that there are languages that cannot be accepted by any finite-state automaton.

**Example.** There is no deterministic finite automaton which accepts the language \( L = \{a^nb^n : n = 1, 2, 3, \ldots\} \).

This is proved by contradiction. Suppose that there is a DFA \( (Q, \{a, b\}, \delta, i, T) \) which accepts the language \( L \). For each \( n \) let \( q_n \) be the state after the word \( a^n \) is input, i.e. \( q_n = \delta(i, a^n) \). Then \( q_1, q_2, q_3, \ldots \) cannot all be distinct since the set \( Q \) of states is finite. Hence \( q_r = q_s \) for some \( r, s \) with \( r \neq s \).

Now if the word \( a^rb^r \) is input we obtain the final state 
\[
\delta(i, a^rb^r) = \delta(q_r, b^r) = \delta(q_s, b^r) = \delta(i, a^rb^r) \in T.
\]

Hence \( a^rb^r \in L \). But since \( r \neq s \) we have \( a^rb^r \notin L \), a contradiction.

If a word \( z \) belongs to a language accepted by an automaton with \( N \) states and \( |z| > N \) then the accepting path for \( z \) must contain a repeated state and therefore will have the form
\[
i \rightarrow \underbrace{v \rightarrow \cdots \rightarrow q \rightarrow \cdots \rightarrow q \rightarrow \cdots \rightarrow v \rightarrow \cdots \rightarrow t}_w\]

where \( i \) is the initial state, \( t \) is an accepting state and \( z = uvw \), where \( v \neq \emptyset \).

Now we can repeat the path labelled with \( v \) any number of times and obtain an accepting path with label \( uv^n w \). Hence \( uv^n w \) is also accepted by the automaton.

Also if \( q \) is taken to be the first repeated state then the path from \( i \) to the second \( q \) can contain at most \( N + 1 \) states, so its label \( uv \) has length at most \( N \).

The word \( uv^n w \) can be thought of as having been obtained by “pumping” the substring \( v \) of \( z = uvw \). We have now proved the following result which is often used for showing that certain languages cannot be accepted by finite automata.

**The Pumping Lemma.** Let \( L \subseteq A^* \) be a language which is accepted by some deterministic finite automaton with \( N \) states. Then every word \( z \in L \) with \( |z| \geq N \) can be written in the form \( z = uvw \) for some \( u, v, w \in A^* \) with

1. \( v \neq \emptyset \)
2. \( |uv| \leq N \)
3. \( uv^n w \in L \) for \( n = 1, 2, 3, \ldots \)

This just says that any sufficiently long word which is accepted has a substring which can be repeated (“pumped”) to give us more words which are accepted.

As an application of the pumping lemma we show that there is no deterministic finite automaton which can recognise whether its input is a palindrome or not. This also shows the limited power of a finite state automaton as a model of computation.

**Problem.** Let \( A \) be an alphabet which contains at least 2 letters and let \( P \) be the set of palindromes over \( A \), i.e. \( P = \{w \in A^* : w = w^R\} \). Show that there is no deterministic finite automaton which accepts \( P \).

**Solution.** Suppose by way of contradiction that there is an automaton that accepts \( P \) and let \( N \) be the number of states that it possesses. Let \( a, b \) be distinct letters in \( A \) and choose some \( m \) with \( m > N \). The word \( a^mb^m \) is a palindrome and has length \( > N \) so by the pumping lemma we can write
\[
\underbrace{a^m b^m}_w = uvw \quad \text{where} \quad v \neq \emptyset \quad \text{and} \quad |uv| \leq N < m
\]
and \( uv^n w \) is a palindrome for every \( n \geq 1 \). But now
\[
a^m b^m = uvw \quad \text{where} \quad \text{uw is shorter than} \quad a^m.
\]

Comparing the letters in these two words we see that \( uv \) must be a string of \( a \)‘s and hence \( v \) must be a string of at least one \( a \).

It now follows that \( u^2 w = a^{m+1} b^m \) for some \( r \geq 1 \). By the pumping lemma this word must belong to the language \( P \) of all palindromes, but is clearly not a palindrome, giving us a contradiction.

Hence the language \( P \) is not accepted by any deterministic finite automaton.