Chapter 3

DFA's, NFA's, Regular Languages

The family of regular languages is the simplest, yet interesting family of languages.

We give six definitions of the regular languages.

1. Using *deterministic finite automata (DFAs)*.
2. Using *nondeterministic finite automata (NFAs)*.
3. Using a *closure definition* involving, union, concatenation, and Kleene $*$.
4. Using *regular expressions*.
5. Using *right-invariant equivalence relations of finite index* (the Myhill-Nerode characterization).
6. Using *right-linear context-free grammars*.

We prove the equivalence of these definitions, often by providing an *algorithm* for converting one formulation into another.

We find that the introduction of NFA's is motivated by the conversion of regular expressions into DFA's.

To finish this conversion, we also show that every NFA can be converted into a DFA (using the *subset construction*).

So, although NFA's often allow for more concise descriptions, they do not have more expressive power than DFA's.

NFA's operate according to the paradigm: *guess a successful path, and check it in polynomial time*.

This is the essence of an important class of hard problems known as $\mathcal{NP}$, which will be investigated later.

We will also discuss methods for proving that certain languages are not regular (Myhill-Nerode, pumping lemma).

We present algorithms to convert a DFA to an equivalent one with a minimal number of states.

3.1 Deterministic Finite Automata (DFA's)

First we define what DFA's are, and then we explain how they are used to accept or reject strings. Roughly speaking, a DFA is a finite transition graph whose edges are labeled with letters from an alphabet Σ.

The graph also satisfies certain properties that make it deterministic. Basically, this means that given any string w, starting from any node, *there is a unique path in the graph "parsing" the string* w.

Example 1. A DFA for the language

$$L_1 = \{ab\}^+ = \{ab\}^*\{ab\},$$

i.e.,

$$L_1 = \{ab, abab, ababab, \ldots, (ab)^n, \ldots\}.$$

Input alphabet: $\Sigma = \{a, b\}$.

State set $Q_1 = \{0, 1, 2, 3\}$.

Start state: 0.

Set of accepting states: $F_1 = \{2\}$.

Transition table (function) δ_1:

	a	b
0	1	3
1	3	2
2	1	3
3	3	3

Note that state 3 is a *trap state* or *dead state*.

Here is a graph representation of the DFA specified by the transition function shown above:

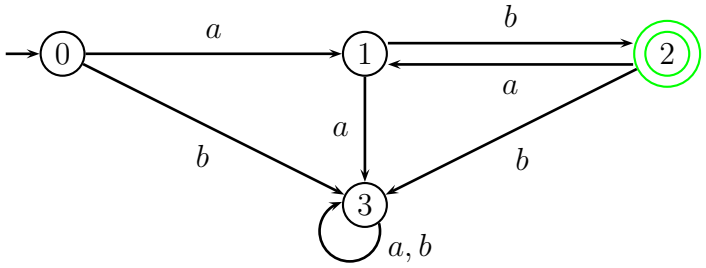

Figure 3.1: DFA for $\{ab\}^+$

Example 2. A DFA for the language

$$L_2 = \{ab\}^* = L_1 \cup \{\epsilon\}$$

i.e.,

$$L_2 = \{\epsilon, ab, abab, ababab, \ldots, (ab)^n, \ldots\}.$$

Input alphabet: $\Sigma = \{a, b\}$.

State set $Q_2 = \{0, 1, 2\}$.

Start state: 0.

Set of accepting states: $F_2 = \{0\}$.

Transition table (function) δ_2:

	a	b
0	1	2
1	2	0
2	2	2

State 2 is a *trap state* or *dead state*.

Here is a graph representation of the DFA specified by the transition function shown above:

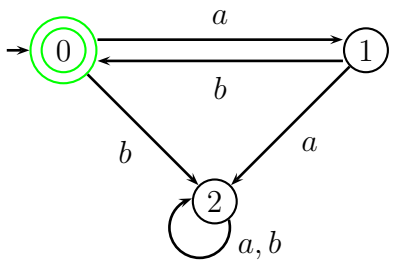

Figure 3.2: DFA for $\{ab\}^*$

Example 3. A DFA for the language

$$L_3 = \{a, b\}^*\{abb\}.$$

Note that L_3 consists of all strings of a's and b's ending in abb.

Input alphabet: $\Sigma = \{a, b\}$.

State set $Q_3 = \{0, 1, 2, 3\}$.

Start state: 0.

Set of accepting states: $F_3 = \{3\}$.

Transition table (function) δ_3:

	a	b
0	1	0
1	1	2
2	1	3
3	1	0

Here is a graph representation of the DFA specified by the transition function shown above:

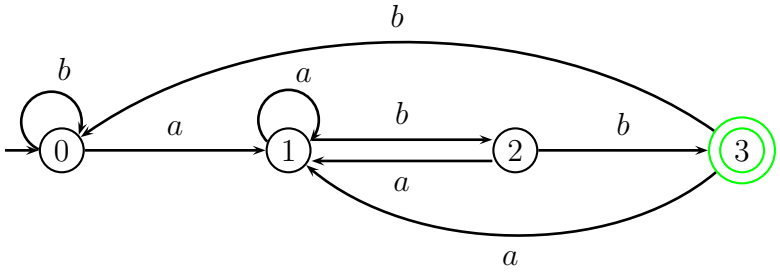

Figure 3.3: DFA for $\{a, b\}^*\{abb\}$

Is this a minimal DFA?

Definition 3.1. A *deterministic finite automaton (or DFA)* is a quintuple $D = (Q, \Sigma, \delta, q_0, F)$, where

- Σ is a finite *input alphabet*;
- Q is a finite set of *states*;
- F is a subset of Q of *final (or accepting) states*;
- $q_0 \in Q$ is the *start state (or initial state)*;
- δ is the *transition function*, a function

$$\delta \colon Q \times \Sigma \to Q.$$

For any state $p \in Q$ and any input $a \in \Sigma$, the state $q = \delta(p, a)$ is uniquely determined.

Thus, it is possible to define the state reached from a given state $p \in Q$ on input $w \in \Sigma^*$, following the path specified by w.

Technically, this is done by defining the extended transition function $\delta^*\colon Q \times \Sigma^* \to Q$.

Definition 3.2. Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$, the *extended transition function* $\delta^*\colon Q \times \Sigma^* \to Q$ is defined as follows:

$$\begin{aligned} \delta^*(p, \epsilon) &= p, \\ \delta^*(p, ua) &= \delta(\delta^*(p, u), a), \end{aligned}$$

where $a \in \Sigma$ and $u \in \Sigma^*$.

It is immediate that $\delta^*(p, a) = \delta(p, a)$ for $a \in \Sigma$.

The meaning of $\delta^*(p, w)$ is that *it is the state reached from state p following the path from p specified by w.*

We can show (by induction on the length of v) that

$$\delta^*(p, uv) = \delta^*(\delta^*(p, u), v) \quad \text{for all } p \in Q \text{ and all } u, v \in \Sigma^*$$

For the induction step, for $u \in \Sigma^*$, and all $v = ya$ with $y \in \Sigma^*$ and $a \in \Sigma$,

$$\begin{aligned} \delta^*(p, uya) &= \delta(\delta^*(p, uy), a) && \text{by definition of } \delta^* \\ &= \delta(\delta^*(\delta^*(p, u), y), a) && \text{by induction} \\ &= \delta^*(\delta^*(p, u), ya) && \text{by definition of } \delta^*. \end{aligned}$$

We can now define how a DFA accepts or rejects a string.

Definition 3.3. Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$, the *language $L(D)$ accepted (or recognized) by D* is the language

$$L(D) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}.$$

Thus, a string $w \in \Sigma^*$ is accepted iff the path from q_0 on input w ends in a final state.

The definition of a DFA does not prevent the possibility that a DFA may have states that are not reachable from the start state q_0, which means that *there is no path from q_0 to such states*.

For example, in the DFA D_1 defined by the transition table below and the set of final states $F = \{1, 2, 3\}$, the states in the set $\{0, 1\}$ are reachable from the start state 0, but the states in the set $\{2, 3, 4\}$ are not (even though there are transitions from $2, 3, 4$ to 0, but they go in the wrong direction).

	a	b
0	1	0
1	0	1
2	3	0
3	4	0
4	2	0

Since there is no path from the start state 0 to any of the states in $\{2, 3, 4\}$, *the states $2, 3, 4$ are useless as far as acceptance of strings*, so they should be deleted as well as the transitions from them.

Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$, the above suggests defining the set Q_r of *reachable* (or *accessible*) states as

$$Q_r = \{p \in Q \mid (\exists u \in \Sigma^*)(p = \delta^*(q_0, u))\}.$$

The set Q_r consists of those states $p \in Q$ such that there is some path from q_0 to p (along some string u).

Computing the set Q_r is a *reachability problem in a directed graph*. There are various algorithms to solve this problem, including breadth-first search or depth-first search.

Once the set Q_r has been computed, we can clean up the DFA D by deleting all redundant states in $Q - Q_r$ and all transitions from these states.

More precisely, we form the DFA
$D_r = (Q_r, \Sigma, \delta_r, q_0, Q_r \cap F)$, where $\delta_r \colon Q_r \times \Sigma \to Q_r$ is the restriction of $\delta \colon Q \times \Sigma \to Q$ to Q_r.

If D_1 is the DFA of the previous example, then the DFA $(D_1)_r$ is obtained by deleting the states $2, 3, 4$:

	a	b
0	1	0
1	0	1

It can be shown that $L(D_r) = L(D)$ (see the homework problems).

A DFA D such that $Q = Q_r$ is said to be *trim* (or *reduced*).

Observe that the DFA D_r is trim. *A minimal DFA must be trim*.

Computing Q_r gives us a method to test whether a DFA D accepts a nonempty language. Indeed

$$L(D) \neq \emptyset \quad \text{iff} \quad Q_r \cap F \neq \emptyset. \qquad (*_{\text{emptyness}})$$

We now come to the first of several equivalent definitions of the regular languages.

Regular Languages, Version 1

Definition 3.4. A language L is a *regular language* if it is accepted by some DFA.

Note that a regular language may be accepted by many different DFAs. Later on, we will investigate how to find minimal DFA's.

For a given regular language L, *a minimal DFA for L is a DFA with the smallest number of states among all DFA's accepting L* .

A minimal DFA for L must exist since every nonempty subset of natural numbers has a smallest element.

In order to understand how complex the regular languages are, we will investigate the closure properties of the regular languages under union, intersection, complementation, concatenation, and Kleene $*$.

It turns out that the family of regular languages is closed under all these operations. For union, intersection, and complementation, we can use the cross-product construction which preserves determinism.

However, for concatenation and Kleene $*$, there does not appear to be any method involving DFA's only. The way to do it is to introduce nondeterministic finite automata (NFA's), which we do a little later.

3.2 The "Cross-product" Construction

Let $\Sigma = \{a_1, \ldots, a_m\}$ be an alphabet.

Given any two DFA's $D_1 = (Q_1, \Sigma, \delta_1, q_{0,1}, F_1)$ and $D_2 = (Q_2, \Sigma, \delta_2, q_{0,2}, F_2)$, there is a very useful construction for showing that the union, the intersection, or the relative complement of regular languages, is a regular language.

Given any two languages L_1, L_2 over Σ, recall that

$$\begin{aligned} L_1 \cup L_2 &= \{w \in \Sigma^* \mid w \in L_1 \quad \text{or} \quad w \in L_2\}, \\ L_1 \cap L_2 &= \{w \in \Sigma^* \mid w \in L_1 \quad \text{and} \quad w \in L_2\}, \\ L_1 - L_2 &= \{w \in \Sigma^* \mid w \in L_1 \quad \text{and} \quad w \notin L_2\}. \end{aligned}$$

Let us first explain how to constuct a DFA accepting the intersection $L_1 \cap L_2$. Let D_1 and D_2 be DFA's such that $L_1 = L(D_1)$ and $L_2 = L(D_2)$.

The idea is to construct a DFA *simulating D_1 and D_2 in parallel*. This can be done by using states which are pairs $(p_1, p_2) \in Q_1 \times Q_2$.

Thus, we define the DFA D as follows:

$$D = (Q_1 \times Q_2, \Sigma, \delta, (q_{0,1}, q_{0,2}), F_1 \times F_2),$$

where the transition function $\delta\colon (Q_1 \times Q_2) \times \Sigma \to Q_1 \times Q_2$ is defined as follows:

$$\delta((p_1, p_2),\ a) = (\delta_1(p_1, a),\ \delta_2(p_2, a)),$$

for all $p_1 \in Q_1$, $p_2 \in Q_2$, and $a \in \Sigma$.

Clearly, D is a DFA, since D_1 and D_2 are. Also, by the definition of δ, we have

$$\delta^*((p_1, p_2),\, w) = (\delta_1^*(p_1, w),\, \delta_2^*(p_2, w)),$$

for all $p_1 \in Q_1$, $p_2 \in Q_2$, and $w \in \Sigma^*$.

Now, we have $w \in L(D_1) \cap L(D_2)$

$$\begin{aligned}
&\text{iff} \quad w \in L(D_1) \text{ and } w \in L(D_2),\\
&\text{iff} \quad \delta_1^*(q_{0,1}, w) \in F_1 \text{ and } \delta_2^*(q_{0,2}, w) \in F_2,\\
&\text{iff} \quad (\delta_1^*(q_{0,1}, w), \delta_2^*(q_{0,2}, w)) \in F_1 \times F_2,\\
&\text{iff} \quad \delta^*((q_{0,1}, q_{0,2}),\, w) \in F_1 \times F_2,\\
&\text{iff} \quad w \in L(D).
\end{aligned}$$

Thus, $L(D) = L(D_1) \cap L(D_2)$.

We can now modify D very easily to accept $L(D_1) \cup L(D_2)$.

We change the set of final states so that it becomes $(F_1 \times Q_2) \cup (Q_1 \times F_2)$.

Indeed, $w \in L(D_1) \cup L(D_2)$

iff $w \in L(D_1)$ or $w \in L(D_2)$,
iff $\delta_1^*(q_{0,1}, w) \in F_1$ or $\delta_2^*(q_{0,2}, w) \in F_2$,
iff $(\delta_1^*(q_{0,1}, w), \delta_2^*(q_{0,2}, w)) \in (F_1 \times Q_2) \cup (Q_1 \times F_2)$,
iff $\delta^*((q_{0,1}, q_{0,2}),\ w) \in (F_1 \times Q_2) \cup (Q_1 \times F_2)$,
iff $w \in L(D)$.

Thus, $L(D) = L(D_1) \cup L(D_2)$.

We can also modify D very easily to accept $L(D_1) - L(D_2)$.

We change the set of final states so that it becomes $F_1 \times (Q_2 - F_2)$.

Indeed, $w \in L(D_1) - L(D_2)$

$$\begin{aligned}
&\text{iff} \quad w \in L(D_1) \text{ and } w \notin L(D_2),\\
&\text{iff} \quad \delta_1^*(q_{0,1}, w) \in F_1 \text{ and } \delta_2^*(q_{0,2}, w) \notin F_2,\\
&\text{iff} \quad (\delta_1^*(q_{0,1}, w), \delta_2^*(q_{0,2}, w)) \in F_1 \times (Q_2 - F_2),\\
&\text{iff} \quad \delta^*((q_{0,1}, q_{0,2}),\ w) \in F_1 \times (Q_2 - F_2),\\
&\text{iff} \quad w \in L(D).
\end{aligned}$$

Thus, $L(D) = L(D_1) - L(D_2)$.

In all cases, if D_1 has n_1 states and D_2 has n_2 states, the DFA D has $n_1 n_2$ states.

Definition 3.5. The *equivalence problem for DFA's* is the following problem: given some alphabet Σ, is there an algorithm which takes as input any two DFA's D_1 and D_2 and decides whether $L(D_1) = L(D_2)$.

The cross-product construction yields an algorithm for deciding the equivalence problem for DFA's; see the course notes.

3.3 Nondeteterministic Finite Automata (NFA's)

NFA's are obtained from DFA's by allowing multiple transitions from a given state on a given input.

This can be done by defining $\delta(p, a)$ as a **subset** of Q rather than a single state. It will also be convenient to allow transitions on input ϵ.

We let 2^Q denote the set of all subsets of Q, including the empty set. The set 2^Q is the *power set* of Q.

Example 4. A NFA for the language

$$L_3 = \{a, b\}^*\{abb\}.$$

Input alphabet: $\Sigma = \{a, b\}$.

State set $Q_4 = \{0, 1, 2, 3\}$.

Start state: 0.

Set of accepting states: $F_4 = \{3\}$.

Transition table δ_4:

	a	b
0	$\{0, 1\}$	$\{0\}$
1	$\emptyset$	$\{2\}$
2	$\emptyset$	$\{3\}$
3	$\emptyset$	$\emptyset$

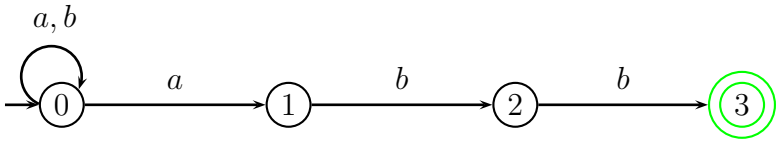

Figure 3.4: NFA for $\{a, b\}^*\{abb\}$

Example 5. Let $\Sigma = \{a_1, \ldots, a_n\}$, with $n \geq 2$, let

$$L_n^i = \{w \in \Sigma^* \mid w \text{ contains an odd number of } a_i\text{'s}\},$$

and let

$$L_n = L_n^1 \cup L_n^2 \cup \cdots \cup L_n^n.$$

The language L_n consists of those strings in Σ^* that contain an odd number of some letter $a_i \in \Sigma$.

Equivalently $\Sigma^* - L_n$ consists of those strings in Σ^* with an even number of *every* letter $a_i \in \Sigma$.

It can be shown that every DFA accepting L_n has at least 2^n states.

However, there is an NFA with $2n + 1$ states accepting L_n.

We define NFA's as follows.

Definition 3.6. A *nondeterministic finite automaton (or NFA)* is a quintuple $N = (Q, \Sigma, \delta, q_0, F)$, where

- Σ is a finite *input alphabet*;
- Q is a finite set of *states*;
- F is a subset of Q of *final (or accepting) states*;
- $q_0 \in Q$ is the *start state (or initial state)*;
- δ is the *transition function*, a function

$$\delta\colon Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q.$$

For any state $p \in Q$ and any input $a \in \Sigma \cup \{\epsilon\}$, the set of states $\delta(p, a)$ is uniquely determined. We write $q \in \delta(p, a)$.

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we would like to define the language accepted by N.

However, given an NFA N, unlike the situation for DFA's, given a state $p \in Q$ and some input $w \in \Sigma^*$, in general *there is no unique path from p on input w, but instead a tree of computation paths*.

For example, given the NFA shown below,

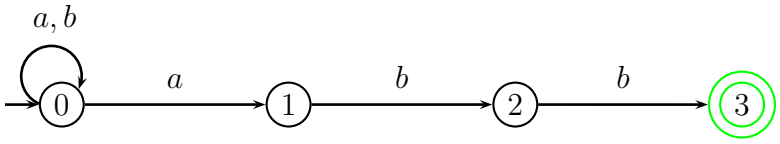

Figure 3.5: NFA for $\{a, b\}^*\{abb\}$

from state 0 on input $w = ababb$ we obtain the following tree of computation paths:

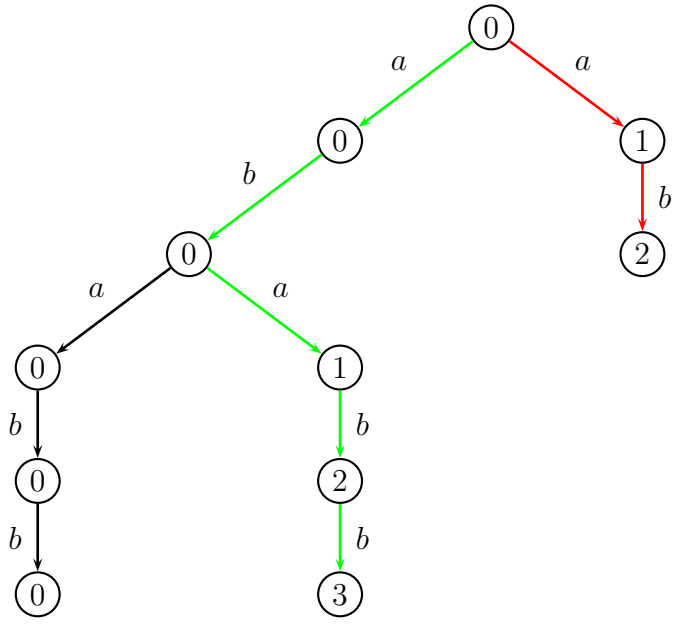

Figure 3.6: A tree of computation paths on input *ababb*

Observe that there are three kinds of computation paths:

1. A path on input w ending in a *rejecting state* (for example, the lefmost path).
2. A path on some proper prefix of w, along which *the computation gets stuck* (for example, the rightmost path).
3. A path on input w ending in an *accepting state* (such as the path ending in state 3).

The acceptance criterion for NFA is *very lenient*: a string w is accepted iff *the tree of computation paths contains some accepting path* (of type (3)).

Thus, all failed paths of type (1) and (2) are ignored. Furthermore, there is *no charge* for failed paths.

A string w is rejected iff *all computation paths are failed paths of type (1) or (2)*.

The "philosophy" of nondeterminism is that an NFA "guesses" an accepting path and then checks it in polynomial time by following this path. We are only charged for one accepting path (even if there are several accepting paths).

A way to capture this acceptance policy if to extend the transition function $\delta\colon Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$ to a function

$$\delta^*\colon Q \times \Sigma^* \to 2^Q.$$

The presence of ϵ-transitions (i.e., when $q \in \delta(p, \epsilon)$) causes technical problems, and to overcome these problems, we introduce the notion of ϵ-closure.

3.4 ϵ-Closure

Definition 3.7. Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$ (with ϵ-transitions) for every state $p \in Q$, the *ϵ-closure of p* is set ϵ-closure(p) consisting of all states q such that there is a path from p to q whose spelling is ϵ (an *ϵ-path*).

This means that either $q = p$, or that *all the edges on the path from p to q have the label ϵ*.

We can compute ϵ-closure(p) using a sequence of approximations as follows. Define the sequence of sets of states $(\epsilon\text{-clo}_i(p))_{i \geq 0}$ as follows:

$$\begin{aligned} \epsilon\text{-clo}_0(p) &= \{p\}, \\ \epsilon\text{-clo}_{i+1}(p) &= \epsilon\text{-clo}_i(p) \cup \\ &\qquad \{q \in Q \mid \exists s \in \epsilon\text{-clo}_i(p),\ q \in \delta(s, \epsilon)\}. \end{aligned}$$

Since $\epsilon\text{-clo}_i(p) \subseteq \epsilon\text{-clo}_{i+1}(p)$, $\epsilon\text{-clo}_i(p) \subseteq Q$, for all $i \geq 0$, and Q is finite, it can be shown that there is a smallest i, say i_0, such that

$$\epsilon\text{-clo}_{i_0}(p) = \epsilon\text{-clo}_{i_0+1}(p).$$

It suffices to show that there is some $i \geq 0$ such that $\epsilon\text{-clo}_i(p) = \epsilon\text{-clo}_{i+1}(p)$, because then there is a smallest such i (since every nonempty subset of $\mathbb{N}$ has a smallest element).

Assume by contradiction that

$$\epsilon\text{-clo}_i(p) \subset \epsilon\text{-clo}_{i+1}(p) \quad \text{for all } i \geq 0.$$

Then, I claim that $|\epsilon\text{-clo}_i(p)| \geq i + 1$ for all $i \geq 0$.

This is true for $i = 0$ since $\epsilon\text{-clo}_0(p) = \{p\}$.

Since $\epsilon\text{-clo}_i(p) \subset \epsilon\text{-clo}_{i+1}(p)$, there is some $q \in \epsilon\text{-clo}_{i+1}(p)$ that does not belong to $\epsilon\text{-clo}_i(p)$, and since by induction $|\epsilon\text{-clo}_i(p)| \geq i + 1$, we get

$$|\epsilon\text{-clo}_{i+1}(p)| \geq |\epsilon\text{-clo}_i(p)| + 1 \geq i + 1 + 1 = i + 2,$$

establishing the induction hypothesis.

If $n = |Q|$, then $|\epsilon\text{-clo}_n(p)| \geq n + 1$, a contradiction.

Therefore, there is indeed some $i \geq 0$ such that $\epsilon\text{-clo}_i(p) = \epsilon\text{-clo}_{i+1}(p)$, and for the least such $i = i_0$, we have $i_0 \leq n - 1$.

It can also be shown that

$$\epsilon\text{-closure}(p) = \epsilon\text{-clo}_{i_0}(p),$$

by proving that

1. $\epsilon\text{-clo}_i(p) \subseteq \epsilon\text{-closure}(p)$, for all $i \geq 0$.
2. $\epsilon\text{-closure}(p)_i \subseteq \epsilon\text{-clo}_{i_0}(p)$, for all $i \geq 0$.

where $\epsilon\text{-closure}(p)_i$ is the set of states reachable from p by an ϵ-path of length $\leq i$.

When N has no ϵ-transitions, i.e., when $\delta(p, \epsilon) = \emptyset$ for all $p \in Q$ (which means that δ can be viewed as a function $\delta\colon Q \times \Sigma \to 2^Q$), we have

$$\epsilon\text{-closure}(p) = \{p\}.$$

It should be noted that there are more efficient ways of computing ϵ-closure(p), for example, using a stack (basically, a kind of depth-first search).

We present such an algorithm below. It is assumed that the types *NFA* and *stack* are defined. If n is the number of states of an NFA N, we let

eclotype = **array**$[1..n]$ **of boolean**

```
function eclosure[N: NFA, p: integer]: eclotype;
    begin
      var eclo: eclotype, q, s: integer, st: stack;
      for each q ∈ setstates(N) do
        eclo[q] := false;
      endfor
      eclo[p] := true; st := empty;
      trans := deltatable(N);
      st := push(st, p);
      while st ≠ emptystack do
        q = pop(st);
        for each s ∈ trans(q, ε) do
          if eclo[s] = false then
            eclo[s] := true; st := push(st, s)
          endif
        endfor
      endwhile;
      eclosure := eclo
    end
```

This algorithm can be easily adapted to compute the set of states reachable from a given state p (in a DFA or an NFA).

Given a subset S of Q, we define ϵ-closure(S) as

$$\epsilon\text{-closure}(S) = \bigcup_{s \in S} \epsilon\text{-closure}(s),$$

with

$$\epsilon\text{-closure}(\emptyset) = \emptyset.$$

When N has no ϵ-transitions, we have

$$\epsilon\text{-closure}(S) = S.$$

We are now ready to define the extension $\delta^* \colon Q \times \Sigma^* \to 2^Q$ of the transition function $\delta \colon Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$.

3.5 Converting an NFA into a DFA

The intuition behind the definition of the extended transition function is that *$\delta^*(p, w)$ is the set of all states reachable from p by a path whose spelling is w*.

Definition 3.8. Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$ (with ϵ-transitions), the *extended transition function* $\delta^*\colon Q \times \Sigma^* \to 2^Q$ is defined as follows: for every $p \in Q$, every $u \in \Sigma^*$, and every $a \in \Sigma$,

$$\begin{aligned} \delta^*(p, \epsilon) &= \epsilon\text{-closure}(\{p\}), \\ \delta^*(p, ua) &= \epsilon\text{-closure}\Big(\bigcup_{s \in \delta^*(p,u)} \delta(s, a) \Big). \end{aligned}$$

In the second equation, if $\delta^*(p, u) = \emptyset$ then

$$\delta^*(p, ua) = \emptyset.$$

The *language $L(N)$ accepted by an NFA N* is the set

$$L(N) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \emptyset\}.$$

Observe that the definition of $L(N)$ conforms to the lenient acceptance policy: a string w is accepted iff $\delta^*(q_0, w)$ contains *some final state*.

In order to convert an NFA into a DFA we also extend $\delta^* : Q \times \Sigma^* \to 2^Q$ to a function

$$\widehat{\delta} : 2^Q \times \Sigma^* \to 2^Q$$

defined as follows: for every subset S of Q, for every $w \in \Sigma^*$,

$$\widehat{\delta}(S, w) = \bigcup_{s \in S} \delta^*(s, w),$$

with

$$\widehat{\delta}(\emptyset, w) = \emptyset.$$

Let $\mathcal{Q}$ be the subset of 2^Q consisting of those subsets S of Q that are ϵ-closed, i.e., such that

$$S = \epsilon\text{-closure}(S).$$

If we consider the restriction

$$\Delta\colon \mathcal{Q} \times \Sigma \to \mathcal{Q}$$

of $\widehat{\delta}\colon 2^Q \times \Sigma^* \to 2^Q$ to $\mathcal{Q}$ and Σ, we observe that Δ is the transition function of a DFA.

Indeed, this is the transition function of a DFA accepting $L(N)$. It is easy to show that Δ is defined directly as follows (on subsets S in $\mathcal{Q}$):

$$\Delta(S, a) = \epsilon\text{-closure}\left(\bigcup_{s \in S} \delta(s, a)\right),$$

with

$$\Delta(\emptyset, a) = \emptyset.$$

Then, the DFA D is defined as follows:

$$D = (\mathcal{Q}, \Sigma, \Delta, \epsilon\text{-closure}(\{q_0\}), \mathcal{F}),$$

where $\mathcal{F} = \{S \in \mathcal{Q} \mid S \cap F \neq \emptyset\}$.

It is not difficult to show that $L(D) = L(N)$, that is, D is a DFA accepting $L(N)$. For this, we show that

$$\Delta^*(S, w) = \widehat{\delta}(S, w).$$

Thus, *we have converted the NFA N into a DFA D (and gotten rid of ϵ-transitions)*.

Since DFA's are special NFA's, the subset construction shows that DFA's and NFA's accept *the same* family of languages, the *regular languages, version 1* (although not with the same complexity).

The states of the DFA D equivalent to N are ϵ-closed subsets of Q. For this reason, the above construction is often called the *subset construction*. This construction is due to Rabin and Scott.

Michael Rabin and Dana Scott were awarded the prestigious *Turing Award* in 1976 for this important contribution and many others.

Although theoretically fine, the method may construct useless sets S that are not reachable from the start state ϵ-closure$(\{q_0\})$. A more economical construction is given next.

An Algorithm to convert an NFA into a DFA: The "subset construction"

Given an input NFA $N = (Q, \Sigma, \delta, q_0, F)$, a DFA $D = (K, \Sigma, \Delta, S_0, \mathcal{F})$ is constructed. It is assumed that K is a linear array of sets of states $S \subseteq Q$, and Δ is a 2-dimensional array, where $\Delta[i, a]$ is the index of the target state of the transition from $K[i] = S$ on input a, with $S \in K$, and $a \in \Sigma$.

$S_0 :=$ ϵ-closure$(\{q_0\})$; $total := 1$; $K[1] := S_0$;
$marked := 0$;
while $marked < total$ **do**;
  $marked := marked + 1$; $S := K[marked]$;
  for each $a \in \Sigma$ **do**
    $U := \bigcup_{s \in S} \delta(s, a)$; $T :=$ ϵ-closure(U);
    if $T \notin K$ **then**
      $total := total + 1$; $K[total] := T$
    endif;
    $\Delta[marked, a] :=$ index(T)
  endfor
endwhile;
$\mathcal{F} := \{S \in K \mid S \cap F \neq \emptyset\}$

Let us illustrate the subset construction on the NFA of Example 4.

A NFA for the language

$$L_3 = \{a, b\}^*\{abb\}.$$

Transition table δ_4:

	a	b
0	$\{0, 1\}$	$\{0\}$
1	$\emptyset$	$\{2\}$
2	$\emptyset$	$\{3\}$
3	$\emptyset$	$\emptyset$

Set of accepting states: $F_4 = \{3\}$.

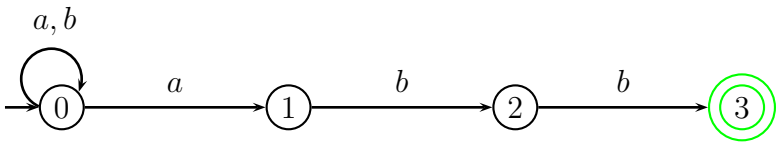

Figure 3.7: NFA for $\{a, b\}^*\{abb\}$

The pointer $\Rightarrow$ corresponds to *marked* and the pointer $\rightarrow$ to *total*.

Initial transition table Δ.

	index	states	a	b
$\Rightarrow$				
$\rightarrow$	A	$\{0\}$		

Just after entering the while loop

	index	states	a	b
$\Rightarrow\rightarrow$	A	$\{0\}$		

After the first round through the while loop.

	index	states	a	b
$\Rightarrow$	A	$\{0\}$	B	A
$\rightarrow$	B	$\{0,1\}$		

After just reentering the while loop.

	index	states	a	b
	A	$\{0\}$	B	A
$\Rightarrow\rightarrow$	B	$\{0,1\}$		

After the second round through the while loop.

	index	states	a	b
	A	$\{0\}$	B	A
$\Rightarrow$	B	$\{0,1\}$	B	C
$\rightarrow$	C	$\{0,2\}$		

After the third round through the while loop.

	index	states	a	b
	A	$\{0\}$	B	A
	B	$\{0,1\}$	B	C
$\Rightarrow$	C	$\{0,2\}$	B	D
$\rightarrow$	D	$\{0,3\}$		

After the fourth round through the while loop.

	index	states	a	b
	A	$\{0\}$	B	A
	B	$\{0,1\}$	B	C
	C	$\{0,2\}$	B	D
$\Rightarrow\rightarrow$	D	$\{0,3\}$	B	A

This is the DFA of Figure 3.3, except that in that example A, B, C, D are renamed $0, 1, 2, 3$.

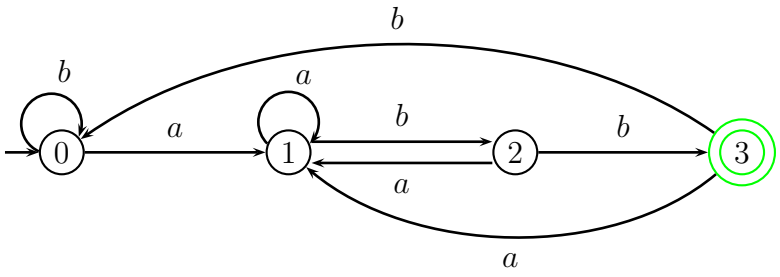

Figure 3.8: DFA for $\{a, b\}^*\{abb\}$