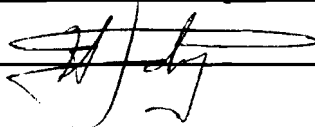


AN ABSTRACT OF THE THESIS OF

Jun Wu for the Master of Science
in Mathematics presented on Dec. 7th, 1992.

Title: Finite State Automata and Regular Languages.

Abstract approved: _____

A handwritten signature in black ink, appearing to be 'Jun Wu', is written over a horizontal line. The signature is stylized and somewhat cursive.

This thesis is intended for an audience familiar with basic formal language theory and finite state automata theory. Chapter 1 is the introduction to regular languages and the operations among them. Chapter 2 is the introduction to deterministic finite state automata and their computation. Chapter 3 is the introduction to nondeterministic finite state automata and shows the equivalence of deterministic finite state automata and nondeterministic finite state automata. Chapter 4 proves the famous Kleen's Theorem and builds up the relationship between regular languages and finite state automata. Chapter 5 discusses the minimization of deterministic finite state automata. Chapter 6 summarizes the thesis.

FINITE STATE AUTOMATA AND REGULAR LANGUAGES

A thesis

Presented to

the Division of Mathematics & Computer Science

EMPORIA STATE UNIVERSITY

in Partial Fulfillment

of the Requirements for the Degree

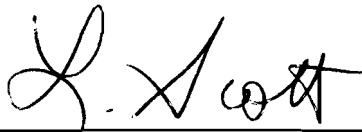
Master of Science

by

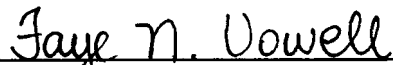
Jun Wu

Dec. 1992

Thesis
1972
W



Approved for the Major Division



Approved for the Graduate Council

CONTENTS

CHAPTER 1: LANGUAGES	1
CHAPTER 2: DETERMINISTIC FINITE STATE AUTOMATA	8
CHAPTER 3: NONDETERMINISTIC FINITE STATE AUTOMATA	15
CHAPTER 4: THE RELATIONSHIP BETWEEN REGULAR LANGUAGES AND FINITE STATE AUTOMATA	32
CHAPTER 5: MINIMIZATION	39
CHAPTER 6: CONCLUSIONS	54
REFERENCES	57

LIST OF TABLES

TABLE 5-1	54
TABLE 5-2	55

LIST OF FIGURES

FIGURE 2-1	9
FIGURE 2-2	9
FIGURE 2-3	11
FIGURE 3-1	17
FIGURE 3-2 (a)	20
FIGURE 3-2 (b)	20
FIGURE 3-3 (a)	21
FIGURE 3-3 (b)	22
FIGURE 3-4	24
FIGURE 3-5	30
FIGURE 4-1	33
FIGURE 4-2	33
FIGURE 4-3	33
FIGURE 4-4	34
FIGURE 4-5	35
FIGURE 4-6	35
FIGURE 4-7	37
FIGURE 4-8	38
FIGURE 5-1	40
FIGURE 5-2	41
FIGURE 5-3	49
FIGURE 5-4	50

Chapter 1 Languages

Section 1-1. Introduction

In this chapter we shall introduce the concept of languages, especially the concept of regular languages. We shall first introduce the necessary terminology and notation about languages and then give the definition of languages and the operations among them. Finally, we shall give the inductive definition of regular languages and illustrate this by an example.

Section 1-2. Strings and Languages

Definition 1-1. An alphabet is a nonempty finite set of symbols.

Thus an example is, naturally, the Roman alphabet { a, b, . . . , z }. In fact, any object can be in an alphabet. It is simply a finite set of any sort.

Definition 1-2. A string over an alphabet is a finite sequence of symbols from the alphabet.

Instead of writing strings with parentheses and commas,

as we denote sequences, we simply juxtapose the symbols. Thus "taxi" is a string over the alphabet $\{ a, b, c, \dots, z \}$, and $\$ab\%$ is a string over $\{ \$, \%, a, b, c \}$. As another example, a binary numeral such as 1011 is a string over $\{ 0, 1 \}$. A string may have no symbols at all. In this case it is called the empty string and is denoted by λ . The set of all strings, including the empty string, over an alphabet Σ is denoted by Σ^* .

The length of a string is its length as a sequence; thus in the examples above the length of the string "taxi" is 5 and the length of the string "\$ab%" is 4. We denote the length of a string w by $|w|$; thus $|10| = 2$ and $|\lambda| = 0$. Alternatively, a string $w \in \Sigma^*$ can be considered as a function $w: \{1, 2, 3, \dots\} \rightarrow \Sigma$; the value of $w(j)$, where $j \in \mathbb{N}$ and $1 \leq j \leq |w|$, is the symbol in the j th position of w . For example if $w = \text{taxi}$, then $w(1) = t$ and $w(5) = i$.

Two strings over the same alphabet can be combined to form a third by the operation of concatenation. The concatenation of string x and string y , written simply as xy , is the string x followed by the string y . Formally,

$w = xy$ if and only if the following 3 conditions hold:

- (1) $|w| = |x| + |y|$
- (2) $w(j) = x(j)$ for $j = 1, \dots, |x|$, and
- (3) $w(|x| + j) = y(j)$ for $j = 1, \dots, |y|$.

For example, the concatenation of 01 and 110 is 01110.

Obviously, $w\lambda = \lambda w = w$ for any string w . And concatenation is associative: $(wx)y = w(xy)$ for any strings w , x , and y .

Section 1-3 Regular Expressions and Regular Languages

Definition 1-3. Any set of strings over an alphabet Σ , that is, any subset of Σ^* will be called a language.

Thus Σ^* , \emptyset , and Σ are languages. Since a language is simply a special kind of set, we can specify a finite language by listing all its strings. For example, $\{aba, czr, d, f\}$ is a language over $\{a, b, \dots, z\}$. However, most languages of interest are infinite. So another way of describing languages is by using what are called regular expressions. This method attempts to mimic the "verbal" description of a language. For example, we may describe a language L by saying that L consists of strings beginning with two 1's, followed by three 0's, and followed by an arbitrary number of either 2's or 3's. Thus 110002332 is in the language, while 11000211 is not. It turns out that it is possible to develop a precise mathematical formalism for such descriptions. We begin with

preliminary concepts and terminology.

Definition 1-4. Let L , L_1 , and L_2 be languages.

1. The language L_1L_2 is defined to be the collection of all strings obtained by concatenating a string from L_1 with a string from L_2 . Formally, $L_1L_2 = \{ x_1x_2 \mid x_1 \in L_1, x_2 \in L_2 \}$.

2. The language L_1+L_2 is the collection of all strings that either belong to L_1 or to L_2 . Thus, L_1+L_2 is just the set union of L_1 and L_2 : $L_1+L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\} = L_1 \cup L_2$.

3. The language L^* is the collection of strings obtained from concatenating an arbitrary but finite number of strings from L . Thus $L^* = \{x_1x_2\dots x_n \mid x_i \in L, i = 1, 2, \dots, n\}$ where n is a nonnegative integer. In this definition of L^* we allow $n = 0$, in which case $x_1x_2\dots x_n$ is to be interpreted as λ , the empty string. Thus for any language L , $\lambda \in L^*$.

We illustrate those by some examples. Let $L_1 = \{01, 1001\}$ and $L_2 = \{11, 00, 1\}$. Then

$$L_1L_2 = \{0111, 0100, 011, 100111, 100100, 10011\} \text{ and}$$

$$L_1+L_2 = \{01, 1001, 11, 00, 1\}.$$

It is obvious that in general, $L_1L_2 \neq L_2L_1$. A string in L_2^* say 1100111: $x_1=11$, $x_2=00$, $x_3=1$, and $x_4=11$ where each $x_i \in L_2$. Such a

representation (partition) is not unique. The same string 1100111 could be represented as $x_1=1, x_2=1, x_3=00, x_4=1, x_5=1,$ and $x_6 = 1.$

We now come to the concept of a regular expression over an alphabet Σ . Regular expressions over an alphabet Σ and the languages denoted by them are defined inductively as follows:

Definition 1-5. Let Σ be an alphabet. A regular expression over Σ is defined inductively by

Base:

1. \emptyset is a regular expression over Σ and \emptyset denotes the empty language, that is the language with no strings at all.

2. λ is a regular expression over Σ and it denotes the language $\{ \lambda \}$, i.e., the language containing the empty string λ .

3. For each $a \in \Sigma$, a is a regular expression and it denotes the language $\{ a \}$, the language consisting of the single string a .

Induction:

4. If r and s are regular expressions over Σ denoting language L_1 and L_2 respectively, then $rs, r + s,$ and r^* are regular expressions, where rs denotes the language $L_1L_2,$ $r + s$ denotes $L_1+L_2,$ and r^* denotes $L_1^*.$

Closure:

5. Any regular expression can be obtained by a finite number of applications of rules 1 through 4.

A comment should be made about points 1 and 2 of the above definition. The regular expression λ is different from the regular expression \emptyset . λ denotes the language consisting of one empty string, while \emptyset denotes the language which contains no strings at all, not even the empty string.

Now we illustrate the definition by giving an example. Let $\Sigma = \{a, b, c\}$. Examples of regular expressions over Σ are a , $a + b$, $ab + c$, and $a^* + bc$. The languages they represent are as follows:

a represents the language $\{ a \}$.

$a + b$ represents the language $\{ a \} + \{ b \} = \{ a, b \}$ consisting of two strings: a and b .

ab represents the language $\{ a \}\{ b \} = \{ ab \}$.

Thus $ab + c$ represents the language $\{ ab \} + \{ c \} = \{ ab, c \}$.

a^* represents the language $\{ \lambda, a, aa, aaa, \dots \}$.

$a^* + bc$ represents the language $\{ bc, \lambda, a, aa, \dots \}$.

Regular expressions may be considerably more complicated than that. Consider, for instance, a regular expression

$X = ab(a + bc)^*(bc + ca)$.

A verbal description of a string x in the language X is as follows: x must begin with ab , followed by some finite number

(or none) of strings each one of which is either a or bc, and ending with either bc or ca. Thus, for instance, ababcaaca is in X, but baabcab is not.

The languages which can be represented by regular expressions are called regular languages. For example, the language X we mentioned above is a regular language because it can be represented by the regular expression $ab(a + ab)^*(bc + ca)$. The languages we shall deal with later will all be regular languages. So we shall simply call them languages although they are only a special kind of language.

Chapter 2 Deterministic Finite State Automata

Section 2-1. Introduction

In this chapter we shall introduce the concept of a deterministic finite state automaton. We shall first motivate the concept through an example, and then, precisely define it. We shall also introduce the necessary terminology and notation concerning finite state automata.

A deterministic finite state automaton will be simply denoted by DFSA.

Section 2-2. Example

Let us consider a simplified version of a soft drink vending machine. Let us assume that

- (1) Each drink is 50 cents,
- (2) The machine will accept only quarters,
- (3) No changes will be returned.

That is, you will get a can of drink after inserting 2 quarters and pressing the drop button. But if you insert 3 or more than 3 quarters what you get after pressing the button is also one single drink. No changes will be returned to you. Now we show how the machine works by a diagram (Fig. 2-1).

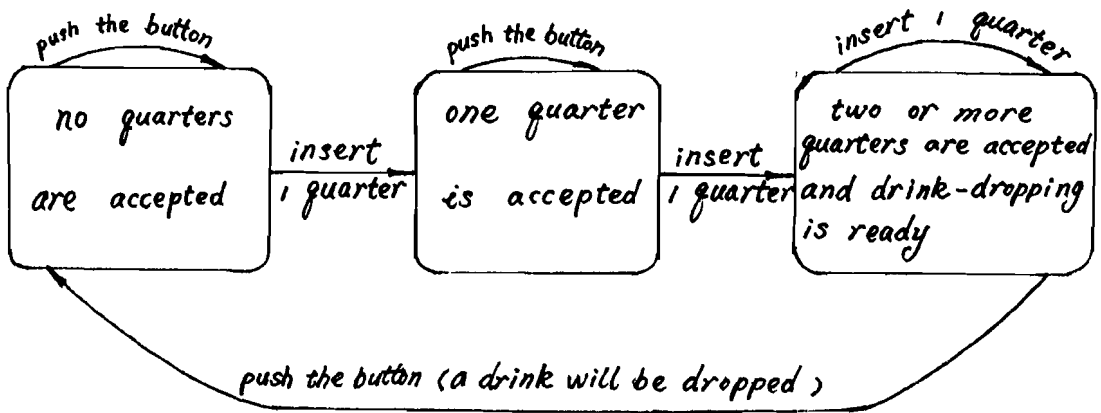


FIGURE 2-1

We can imagine that the machine could be in 3 different states: no quarters are accepted, one quarter is accepted, and two or more quarters are accepted. The machine will stay in the original state or move to another state depending on which input (quarter-inserting or button-pressing) is given and what state it is in. For example, if the machine is in the state that no quarters are accepted, it will stay in that state if the input button-pressing is given, or move to the state that one quarter is accepted if the input quarter-inserting is given. The arrow " $\text{---}\rightarrow$ " means "move to" or "transit to" under the given input.

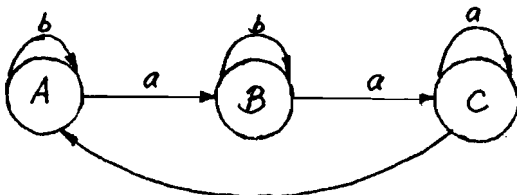


FIGURE 2-2

For convenience, the states no quarters are accepted, one quarter is accepted, and two or more quarters are accepted are denoted by A, B, and C respectively. Similarly, the inputs quarter-inserting and button-pressing are denoted by a and b respectively. So Fig. 2-1 can be redrawn as Fig. 2-2. In fact, this is a DFSA and Fig. 2-2 is usually called a state diagram.

Now we are ready for a rigorous definition.

Section 2-3. Definition

Definition 2-1. A DFSA is a quintuple $M = (S, \Sigma, \delta, s, F)$

where

S is a nonempty finite set of states,

Σ is a nonempty finite set (the input alphabet),

$s \in S$ is the initial state,

$F \subseteq S$ is the set of final states,

and δ , the transition function, is a function from $S \times \Sigma$ to S .

In the example of vending machine $S = \{ A, B, C \}$, $\Sigma = \{ a, b \}$. Also, $s = A$ and $F = \{ C \}$. The transition function δ is

$$\delta(A, a) = B,$$

$$\delta (A, b) = A,$$

$$\delta (B, a) = C,$$

$$\delta (B, b) = B,$$

$$\delta (C, a) = C,$$

$$\delta (C, b) = A.$$

Section 2-4. Movements, Languages, and DFSA

Now we have some basic idea about DFSA. The value $\delta (q, x)$ of the transition function of a DFSA describes the next state. It depends only on the present state q and the present input x , and is completely independent of past moves and inputs. The DFSA stops after it reads all the symbols of the input string. Let us consider the example of the vending machine. On the definition, the machine M could be defined as

$M = (S, \Sigma, \delta, s, F)$, where

$$S = \{ A, B, C \},$$

$$\Sigma = \{ a, b \},$$

$$s = A,$$

$$F = C.$$

The transition function δ is defined as in Fig. 2-2.

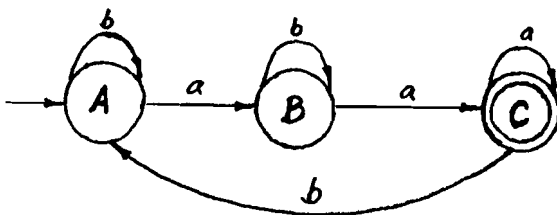


FIGURE 2-3

The state diagram of M can be depicted as in Fig. 2-3. The arrow \rightarrow pointing to A indicates that A is the initial state. The final state C is enclosed in a double circle. We usually indicate the initial state with an arrow, and indicate the final states in double circles in a state diagram. If the input string to the vending machine is baba, then the machine would go through the following sequence of moves:

$$\begin{array}{ccccccc}
 & b & & a & & b & & a \\
 A & \xrightarrow{M} & A & \xrightarrow{M} & B & \xrightarrow{M} & B & \xrightarrow{M} & C.
 \end{array}$$

Here the symbol $q_i \xrightarrow[M]{x} q_j$ means that the machine M changes

state from q_i to q_j upon reading the input x , i.e., $\delta(q_i, x) = q_j$, where $q_i, q_j \in S$ and $x \in \Sigma$. After the entire input string has been read the machine will be in the state C, the final state.

DFSA may be used to "recognize" or "accept" certain kinds of strings and reject others, depending on the state the machine is in after the string has been read. Given a DFSA M, when a string $\sigma = x_1x_2\dots x_n$ is presented to M, it executes the sequence of moves beginning with the initial state and the symbol x_1 . After the symbol x_n has been read we observe the state which the machine has stopped in. If the state is one of the final states, we say that the machine accepts σ ,

otherwise we say that M does not accept σ . In the example described above, the string $\sigma = \text{baba}$ is accepted by the vending machine because it begins with the initial state A and stops in the final state C , after the whole string is read. But the string babb is not accepted by the machine.

To facilitate describing the movements of a machine, we introduce the concept of configuration. A configuration of a machine $M (S, \Sigma, \delta, s, F)$ is a pair (q, σ) , where $q \in S$ and $\sigma = a_k a_{k+1} \dots a_m$ is the unread portion of the input string $a_1 a_2 \dots a_k a_{k+1} \dots a_m$, where $a_1, a_2, \dots, a_m \in \Sigma$. If a machine M is in configuration $(q, a_k a_{k+1} \dots a_m)$ and $\delta (q, a_k) = \hat{q}$, The next configuration of M is $(\hat{q}, a_{k+1} a_{k+2} \dots a_m)$. We denote this by

$$(q, a_k a_{k+1} \dots a_m) \xrightarrow[M]{} (\hat{q}, a_{k+1} a_{k+2} \dots a_m)$$

and call it a move. When the entire string has been read, the configuration of M is (q, λ) , that is, the input string is empty.

If for some sequence of configurations we have the sequence of moves:

$$(q_1, \sigma_1) \xrightarrow[M]{} (q_2, \sigma_2) \xrightarrow[M]{} \dots \xrightarrow[M]{} (q_p, \sigma_p) ,$$

we denote this by

$$(q_1, \sigma_1) \xrightarrow[M]^* (q_p, \sigma_p)$$

and call it a transition. For example, in the example of vending machine we have

$$(A, baba) \xrightarrow[M]{\quad} (A, aba) \xrightarrow[M]{\quad} (B, ba) \xrightarrow[M]{\quad} (B, a) \xrightarrow[M]{\quad} (C, \lambda).$$

So we may denote this by $(A, baba) \xrightarrow[M]{*} (C, \lambda)$.

Now we are ready to relate languages to DFSA by giving a rigorous definition as follows:

Let $M = (S, \Sigma, \delta, s, F)$ be a DFSA. The language $L(M)$ recognized or accepted by M is defined as

$$L(M) = \{ \sigma \in \Sigma^* \mid (s, \sigma) \xrightarrow[M]{*} (q, \lambda) \text{ for some } q \in F \}.$$

Chapter 3 Nondeterministic Finite State Automata

Section 3-1. Introduction and Definition

As we indicated in the previous chapter, a DFSA is deterministic in the sense that for a given input and state, the next state of the machine is completely determined.

In this chapter we add a powerful, though at first not intuitive, feature to DFSA. This feature is called nondeterminism, and is essentially the ability to change states in a way that is only partially determined by the current state and input symbol. That is, we shall now permit several possible "next states" for a given combination of current state and input symbol. The automaton, as it reads an input string, may choose at each step to go into any one of these legal next states. The choice is not determined by anything in our model, and is therefore said to be nondeterministic. On the other hand, the choice is not wholly unlimited either; only those next states that are legal from a given state with a given input symbol can be chosen.

More formally, a nondeterministic finite state automaton, simply denoted by NFSA, is a machine M defined as follows:

Definition 3-1

A NFSA is a quintuple $M = (S, \Sigma, \Delta, s, F)$

where

S is a nonempty finite set of states,

Σ is an alphabet,

$s \in S$ is the initial state,

$F \subseteq S$ is the set of final states,

and Δ , The transition relation, is a finite subset of $S \times \Sigma^* \times S$.

The significance of a triple (q, u, p) being in Δ is that M , when in state q , may consume a string u ($u \in \Sigma^*$) from the input string and enter state p . In other words, $(q, u, p) \in \Delta$ if and only if an arrow $q \xrightarrow{u} p$ appears in the state diagram of

M . Each triple $(q, u, p) \in \Delta$ is called a transition of M . In keeping with the idea that M is a finite device, we have insisted that Δ be a finite set of transitions, even though $S \times \Sigma^* \times S$ is an infinite set.

The formal definition of computations by NFSA are very similar to those for DFSA. A configuration of M is, once again, an element of $S \times \Sigma^*$. The relation $\xrightarrow{\quad}$ between configurations is defined as follows: $(q, w) \xrightarrow{\quad} (q', w')$

if and only if there is a $u \in \Sigma^*$ such that $w = uw'$ and

$(q, u, q') \in \Delta$. Note that \dashrightarrow need not be a function. For some configurations (q, w) , there may be several pairs (q', w') or none at all such that $(q, w) \dashrightarrow (q', w')$. As before, a transition $(q_1, w_1) \dashrightarrow^* (q_p, w_p)$ represents a

sequence of transitions

$$(q_1, w_1) \dashrightarrow (q_2, w_2) \dashrightarrow \dots \dashrightarrow (q_p, w_p)$$

where $q_1, q_2, \dots, q_p \in S$ and $w_1, w_2, \dots, w_p \in \Sigma^*$.

Definition 3-2

A string $w \in \Sigma^*$ is accepted by M if and only if there is a state $q \in F$ such that $(s, w) \dashrightarrow^* (q, \lambda)$. Finally $L(M)$, the

language accepted by M , is the set of all strings accepted by M .

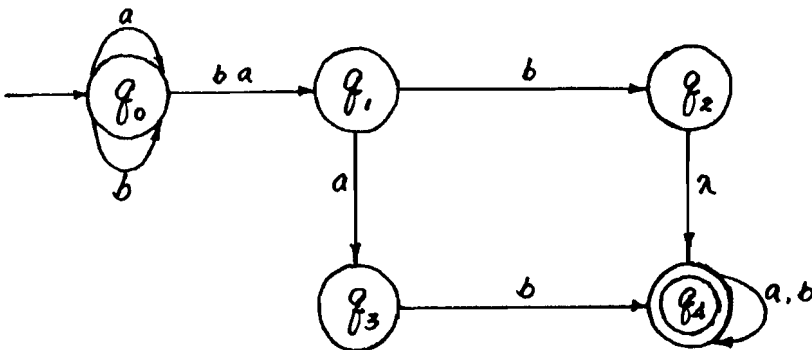


FIGURE 3-1

Example 3-1

Figure 3-1 shows one of several possible NFSA that accept the set of all strings containing an occurrence of the pattern bab or of the pattern baab. Formally, this machine is $(S, \Sigma, \Delta, s, F)$, where

$$S = \{ q_0, q_1, q_2, q_3, q_4 \},$$

$$\Sigma = \{ a, b \},$$

$$s = q_0,$$

$$F = \{ q_4 \},$$

and

$$\Delta = \{ (q_0, a, q_0), (q_0, b, q_0), (q_0, ba, q_1), (q_1, b, q_2), \\ (q_1, a, q_3), (q_2, \lambda, q_4), (q_3, b, q_4), (q_4, a, q_4), \\ (q_4, b, q_4) \}.$$

When M is given the string *baababaab* as input, several different sequences of transitions may ensue. For example, M may wind up in the nonfinal state q_0 in case the only transitions used are (q_0, a, q_0) and (q_0, b, q_0) :

$$(q_0, baababaab) \dashrightarrow (q_0, aababaab)$$

$$\dashrightarrow (q_0, ababaab)$$

$$\dashrightarrow \dots \dashrightarrow (q_0, \lambda).$$

The same input string may drive M from state q_0 to the final state q_4 , and indeed may do so in three different ways. One of the three ways is the following:

$(q_0, baababaab) \dashrightarrow (q_1, aababaab)$

$\dashrightarrow (q_3, babaab)$

$\dashrightarrow (q_4, abaab)$

$\dashrightarrow (q_4, baab)$

$\dashrightarrow (q_4, aab)$

$\dashrightarrow (q_4, ab)$

$\dashrightarrow (q_4, b)$

$\dashrightarrow (q_4, \lambda)$.

Since a string is accepted by a NFSA if and only if there is at least one sequence of transitions leading to a final state, it follows that $baababaab \in L(M)$.

Observe that a DFSA is just a special type of a NFSA: In a DFSA, it happens that the transition relation $\Delta \subseteq S \times \Sigma^* \times S$ is in fact a function from $S \times \Sigma$ to S . In other words, a NFSA = $(S, \Sigma, \Delta, s, F)$ is really a DFSA provided that the following condition is satisfied: If $(q, u, q') \in \Delta$ then $|u| = 1$, and for each $q \in S$ and $\sigma \in \Sigma$, there is a unique $q' \in S$ such that $(q, \sigma, q') \in \Delta$.

Section 3-2. The equivalence of DFSA and NFSA

We next show that although NFSA appear to be more general than DFSA, they are nevertheless no more powerful in terms of the languages they accept: A NFSA can always be converted into a DFSA.

Definition 3-2

Finite automata M_1 and M_2 are said to be equivalent if and only if $L(M_1) = L(M_2)$.

Thus two automata are considered to be equivalent if they accept the same language, even though they may use different methods to do so.

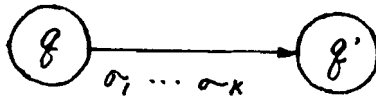


FIGURE 3-2(a)

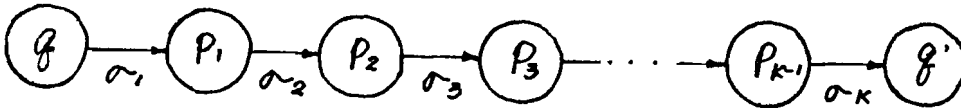


FIGURE 3-2(b)

Theorem 3-1

For each NFSA there is an equivalent DFSA.

Proof: Let $M = (S, \Sigma, \Delta, s, F)$ be a NFSA. In order to transform M into an equivalent DFSA, various possibilities must be eliminated: transitions $(q, u, q') \in \Delta$ such that $u = \lambda$ or such that $|u| > 1$; transitions that are missing or undefined; and multiple transitions that may be applicable to the same configuration. It is relatively easy to eliminate moves (q, u, q') with $|u| > 1$. In essence, we introduce new states such as those in Fig. 3-2 (b) to replace an arrow in the state diagram such as that shown in Fig. 3-2 (a).

Formally, if $(q, \sigma_1\sigma_2\dots\sigma_k, q') \in \Delta$ $\sigma_1, \sigma_2, \dots, \sigma_k \in \Sigma, k \geq 2$, then add new (nonfinal) states p_1, \dots, p_{k-1} to S and new transitions $(q, \sigma_1, p_1), (p_1, \sigma_2, p_2), \dots, (p_{k-1}, \sigma_k, q')$ to Δ .

Let $M' = (S', \Sigma, \Delta', s', F')$ be the NFSA that results from M when this transformation is carried out for each move (q, u, q') of M such that $|u| > 1$. It should be obvious that M' and M are equivalent, and that $|u| \leq 1$ for each move (q, u, q') of M' .

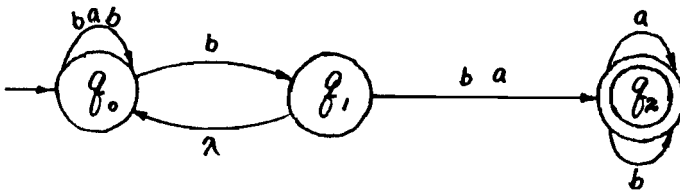


FIGURE 3-3(a)

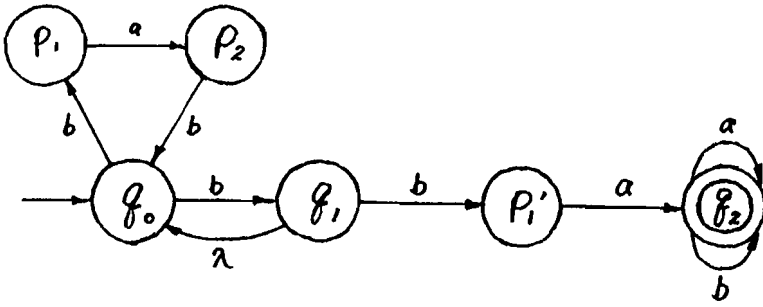


FIGURE 3-3(b)

Example 3-2 when this transformation is carried out on the NFSA of Fig. 3-3 (a), the NFSA of Fig. 3-3 (b) results.

We shall now construct a DFSA $M'' = (S'', \Sigma, \delta'', s'', F'')$ equivalent to M' ; this construction will suffice to establish the theorem. The key idea is to view a NFSA as occupying, at any moment, not a single state but a set of states: namely, all the states that can be reached from the initial state by means of the input consumed thus far. So if M' had five states $\{q_0, \dots, q_4\}$ and, after reading a certain input string, it could be in state $q_0, q_2, \text{ or } q_3$ but not q_1 or q_4 , its state could be considered to be the set $\{q_0, q_2, q_3\}$, rather than an undetermined member of that set. And if the next input symbol could drive M' from q_0 to q_1 or q_2 , from q_2 to q_0 , and from q_3 to q_2 , then the next state of M'' could be considered to be the set $\{q_0, q_1, q_2\}$.

The construction formalizes this idea. The set of states of M'' will be $2^{S'}$, the power set of the set of states of M' . The set of final states of M'' will consist of all those subsets of S' that contain at least one final state of M' . The definition of the transition function of M'' will be slightly more complicated. The basic idea is that a move of M'' on reading an input symbol $\sigma \in \Sigma$ imitates a transition of M' on input symbol σ , followed by some number of transitions of M' on which no input is read. To formalize this idea we need a special definition.

Definition 3-3

For any state $q \in S'$, let $E(q)$ be the set of all states of M' that are reachable from state q without reading any input. That is,

$$E(q) = \{ p \in S' \mid (q, \lambda) \xrightarrow[M']{*} (p, \lambda) \}.$$

If M' moves without consuming any of its input, its operation does not depend on what that input is. So another way to define $E(q)$ would be to pick any string $w \in \Sigma^*$ and write

$$E(q) = \{ p \in S' \mid (q, w) \xrightarrow[M']{*} (p, w) \}.$$

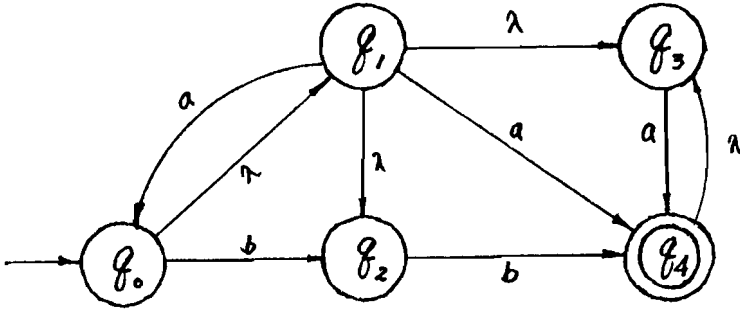


FIGURE 3-4

Example 3-3

In the automaton of Fig. 3-4, $E(q_0) = \{ q_0, q_1, q_2, q_3 \}$,

$E(q_1) = \{ q_1, q_2, q_3 \}$, $E(q_2) = \{ q_2 \}$, $E(q_3) = \{ q_3 \}$,

$E(q_4) = \{ q_3, q_4 \}$.

Now define $M'' = (S'', \Sigma, \delta'', s'', F'')$, where

$$S'' = 2^{S'}$$

$$s'' = E(s')$$

$$F'' = \{ Q \subseteq S' \mid Q \cap F' \neq \emptyset \},$$

and for each $Q \subseteq S'$ and each symbol $\sigma \in \Sigma$,

$$\delta''(Q, \sigma) = \bigcup \{ E(p) \mid p \in S' \text{ and } (q, \sigma, p) \in \Delta' \text{ for some } q \in Q \}.$$

For example, if M' is the automaton of Fig. 3-4, then

$$s'' = E(q_0) = \{ q_0, q_1, q_2, q_3 \}.$$

Since $(q_1, a, q_0) \in \Delta'$ and $(q_1, a, q_4) \in \Delta'$, it follows that

$$\delta'' ((q_1), a) = E(q_0) \cup E(q_4) = \{q_0, q_1, q_2, q_3, q_4\}.$$

Similarly, $(q_0, b, q_2) \in \Delta'$ and $(q_2, b, q_4) \in \Delta'$ so

$$\delta'' ((q_0, q_2), b) = E(q_2) \cup E(q_4) = \{q_2, q_3, q_4\}.$$

It remains to show that M'' is deterministic and equivalent to M' . The demonstration that M'' is deterministic is straightforward since δ'' is single valued and well defined by the way it was constructed. (It is quite possible that $\delta''(Q, \sigma) = \emptyset$ for some $Q \in S'', \sigma \in \Sigma$. Indeed $\delta''(\emptyset, \sigma) = \emptyset$ for each $\sigma \in \Sigma$. But \emptyset is just another member of S'' .)

Lemma 3-1: for any string $w \in \Sigma^*$, and any states $q, p \in S'$,

$$(q, w) \xrightarrow[M']{*} (p, \lambda)$$

if and only if

$$(E(q), w) \xrightarrow[M'']{*} (P, \lambda)$$

for some set P containing p .

From this the theorem will follow easily. To show that M' and M'' are equivalent, consider any string

$w \in \Sigma^*$. Then $w \in L(M')$ iff $(s', w) \xrightarrow[M']{*} (f', \lambda)$ for some $f' \in F'$ (by

definition) and iff $(E(s'), w) \xrightarrow[M'']{*} (Q, \lambda)$ for some Q

containing f' (by the Lemma 3-1); in other words, iff

$$(s'', w) \xrightarrow[M'']{*} (Q, \lambda) \text{ for some } Q \in F''.$$

The last condition is the definition of $w \in L(M'')$.

We prove the lemma by induction on $|w|$.

Basis step:

For $|w| = 0$, that is, for $w = \lambda$, we must show that

$$(q, \lambda) \xrightarrow[M']{*} (p, \lambda)$$

iff $(E(q), \lambda) \xrightarrow[M'']{*} (P, \lambda)$ for some set P containing p .

Suppose $(q, \lambda) \xrightarrow[M']{*} (p, \lambda)$ then $p \in E(q)$. Because

$$(E(q), \lambda) \xrightarrow[M'']{*} (E(q), \lambda) \text{ and } p \in E(q),$$

we have $(E(q), \lambda) \xrightarrow[M'']{*} (P, \lambda)$ for some set P containing p .

Because M'' is deterministic and the transition

$$(E(q), \lambda) \xrightarrow[M'']{*} (P, \lambda)$$

is unique and as we know

$$(E(q), \lambda) \xrightarrow[M'']{*} (E(q), \lambda),$$

we have $P = E(q)$. If $(E(q), \lambda) \xrightarrow[M'']{*} (P, \lambda)$ for some set P

containing p . Since the transition function δ'' of M'' is deterministic and unique and $(E(q), \lambda) \xrightarrow[M'']{*} (E(q), \lambda)$, we

have $P = E(q)$. That is $p \in E(q)$. So by the definition of

$E(q)$, $(q, \lambda) \xrightarrow[M']{*} (p, \lambda)$. This completes the proof of the

basis step.

Induction Hypothesis:

Suppose that the lemma is true for all strings w of length k or less for some $k \geq 0$.

Induction Step:

We prove the lemma for any string w of length $k+1$. Let $w=va$, where $v \in \Sigma^*$ and $a \in \Sigma$.

First suppose that $(q, w) \xrightarrow[M']{*} (p, \lambda)$. Then there are

states

r_1 and r_2 such that

$$(q, va) \xrightarrow[M']{*} (r_1, a) \xrightarrow[M']{} (r_2, \lambda) \xrightarrow[M']{*} (p, \lambda).$$

That is, M' reaches state p from state q by some number of transitions during which input v is read, followed by one transition during which input a is read, followed by some number of transitions during which no input is read. Since

$$(q, va) \xrightarrow[M']{*} (r_1, a), \text{ then } (q, v) \xrightarrow[M']{*} (r_1, \lambda). \text{ Also since } |v|$$

$= k$, by the induction hypothesis

$$(E(q), v) \xrightarrow[M'']{*} (R_1, \lambda) \text{ for some set } R_1 \text{ containing } r_1.$$

Since $(r_1, a) \xrightarrow[M']{} (r_2, \lambda)$, $(r_1, a, r_2) \in \Delta'$, then by the

construction of M'' we get $E(r_2) \subseteq \delta''(R_1, a)$. But since

$$(r_2, \lambda) \xrightarrow[M']{*} (p, \lambda), p \in E(r_2) \text{ we find } p \in \delta''(R_1, a).$$

Therefore $(R_1, a) \xrightarrow[M'']{\dashrightarrow} (P, \lambda)$ for some P containing p and

$$(E(q), va) \xrightarrow[M'']{*} (R_1, a) \xrightarrow[M'']{\dashrightarrow} (P, \lambda).$$

Now suppose that $(E(q), va) \xrightarrow[M'']{*} (R_1, a) \xrightarrow[M'']{\dashrightarrow} (P, \lambda)$ for

some P containing p and some R_1 such that $\delta''(R_1, a) = P$. Now by

the definition of δ'' , $\delta''(R_1, a)$ is the union of all sets

$E(r_2)$, where for every $r_1 \in R_1$, $(r_1, a, r_2) \in \Delta'$. Since $p \in P$ and

$P = \delta''(R_1, a)$, there is a particular r_2 such that $p \in E(r_2)$

and, for some $r_1 \in R_1$, $(r_1, a, r_2) \in \Delta'$. Then $(r_2, \lambda) \xrightarrow[M']{*} (p, \lambda)$

by the definition of $E(r_2)$. Also by the induction

hypothesis, $(q, v) \xrightarrow[M']{*} (r_1, \lambda)$, and therefore

$$(q, va) \xrightarrow[M']{*} (r_1, a) \xrightarrow[M']{\dashrightarrow} (r_2, \lambda) \xrightarrow[M']{*} (p, \lambda).$$

This completes the proof of the lemma and the theorem.

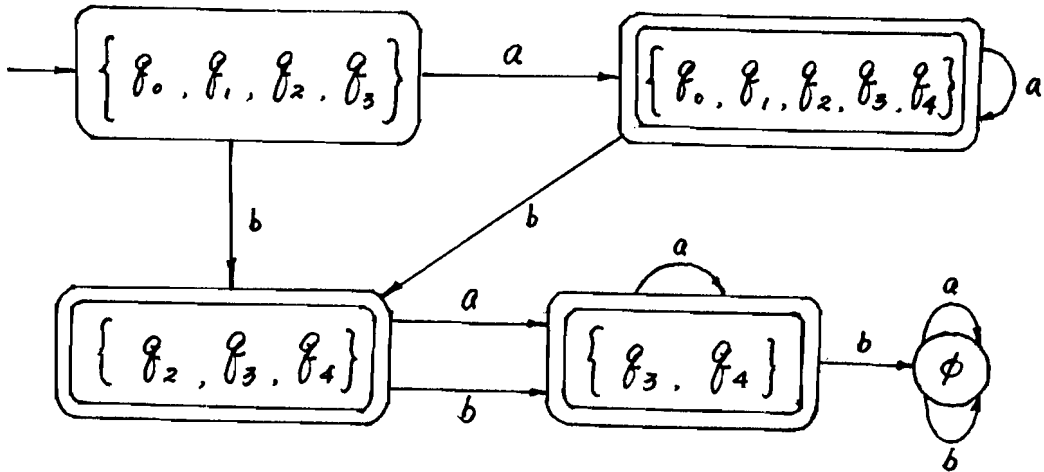


FIGURE 3-5

Example 3-4

This example continues Example 3-3. Let M' be the automaton of Fig. 3-4. Since M' has 5 states, M'' will have $2^5 = 32$ states. However, only a few of these states will be relevant to the operation of M'' , namely, those states that can be reached from state s'' by reading some input string. We shall build this part of M'' by starting from s'' and introducing a new state only when it is needed as the value of $\delta''(q, \sigma)$ for some state q already introduced and some $\sigma \in \Sigma$.

We have already defined $E(q)$ for each state q of M' . Since $s'' = E(q_0) = \{q_0, q_1, q_2, q_3\}$ and

(q_1, a, q_0) , (q_1, a, q_4) , and (q_3, a, q_4)

are all the moves (q, a, p) for some $q \in s''$. It follows that

$$\delta''(s'', a) = E(q_0) \cup E(q_4) = \{q_0, q_1, q_2, q_3, q_4\}.$$

Similarly,

(q_0, b, q_2) and (q_2, b, q_4)

are all the moves (q, b, p) for some $q \in s''$, so

$$\delta''(s'', b) = E(q_2) \cup E(q_4) = \{q_2, q_3, q_4\}.$$

Repeating this calculation for the newly introduced states, we have the following:

$$\delta''(\{q_0, q_1, q_2, q_3, q_4\}, a) = \{q_0, q_1, q_2, q_3, q_4\},$$

$$\delta''(\{q_0, q_1, q_2, q_3, q_4\}, b) = \{q_2, q_3, q_4\},$$

$$\delta''(\{q_2, q_3, q_4\}, a) = E(q_4) = \{q_3, q_4\},$$

$$\text{and } \delta''(\{q_2, q_3, q_4\}, b) = E(q_4) = \{q_3, q_4\}.$$

$$\text{Finally, } \delta''(\{q_3, q_4\}, a) = E(q_4) = \{q_3, q_4\},$$

$$\delta''(\{q_3, q_4\}, b) = \emptyset,$$

$$\text{and } \delta''(\emptyset, a) = \delta''(\emptyset, b) = \emptyset.$$

The relevant part of M'' is illustrated in Fig. 3-5. F'' , the set of final states, contains each set of states of which q_4 is a member, since q_4 is the sole member of F' . So in the illustration, the states $\{q_0, q_1, q_2, q_3, q_4\}$, $\{q_2, q_3, q_4\}$, and $\{q_3, q_4\}$ of M'' are final.

Chapter 4 The Relationship Between Regular Languages And Finite State Automata

Section 4-1. Introduction

In this chapter we shall relate regular languages to the finite state automata by introducing Kleene's famous theorem which was proved in 1956.

Depending on Kleene's theorem, we shall be able to recognize regular languages by finite state automata. That is, this theorem solves the membership problem of regular languages.

Section 4-2. Kleene's Theorem

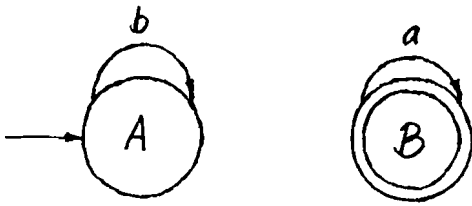
It is easy to see that certain simple languages, for example a^*b^* and $\{a, b\}^*$, can be specified either by regular expressions or by NFSA. We now show that any language that can be represented in one way can also be represented in the other.

Theorem 4-1 Kleene's Theorem

A language is regular if and only if can be accepted by a NFSA.

Proof. (Only If) We proceed through the five parts of

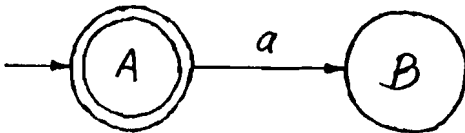
the induction definition of regular expressions and construct a NFSA that accepts exactly the language arising from the five parts of this definition.



$$\begin{aligned}
 M &= (S, \Sigma, \Delta, s, F) \\
 S &= \{ A, B \} \\
 \Sigma &= \{ a, b \} \\
 s &= A \quad F = \{ B \} \\
 \Delta &= \{ (B, a, B), (A, b, A) \}
 \end{aligned}$$

FIGURE 4-1

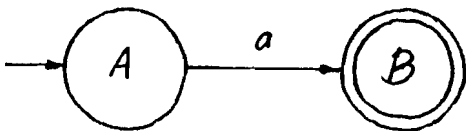
The automaton illustrated in Fig. 4-1 accepts exactly the empty set \emptyset .



$$\begin{aligned}
 M &= (S, \Sigma, \Delta, s, F) \\
 S &= \{ A, B \} \\
 \Sigma &= \{ a \} \\
 s &= A \quad F = \{ A \} \\
 \Delta &= \{ (A, a, B) \}
 \end{aligned}$$

FIGURE 4-2

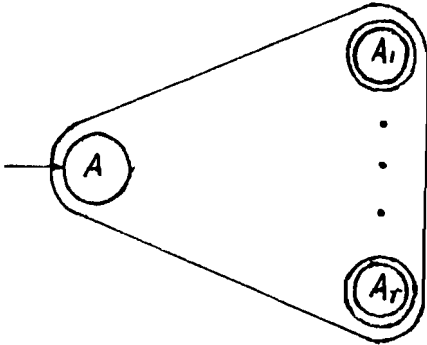
The automaton illustrated in Fig. 4-2 accepts only the empty language λ .



$$\begin{aligned}
 M &= (S, \Sigma, \Delta, s, F) \\
 S &= \{ A, B \} \\
 \Sigma &= \{ a, b, c \} \\
 s &= A \quad F = \{ B \} \\
 \Delta &= \{ (A, a, B) \}
 \end{aligned}$$

FIGURE 4-3

The automaton illustrated in Fig. 4-3 accepts only the language { a }.

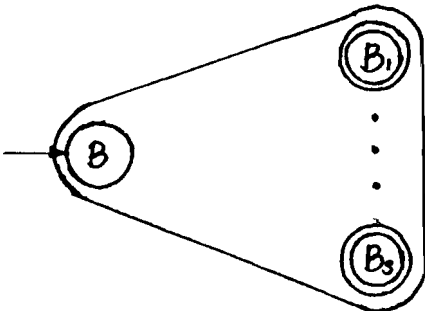


$$M_1 = (S_1, \Sigma_1, \Delta_1, s_1, F_1)$$

$$s_1 = A$$

$$F_1 = \{ A_1, \dots, A_r \}$$

$$L(M_1) = E$$



$$M_2 = (S_2, \Sigma_2, \Delta_2, s_2, F_2)$$

$$s_2 = B$$

$$F_2 = \{ B_1, \dots, B_s \}$$

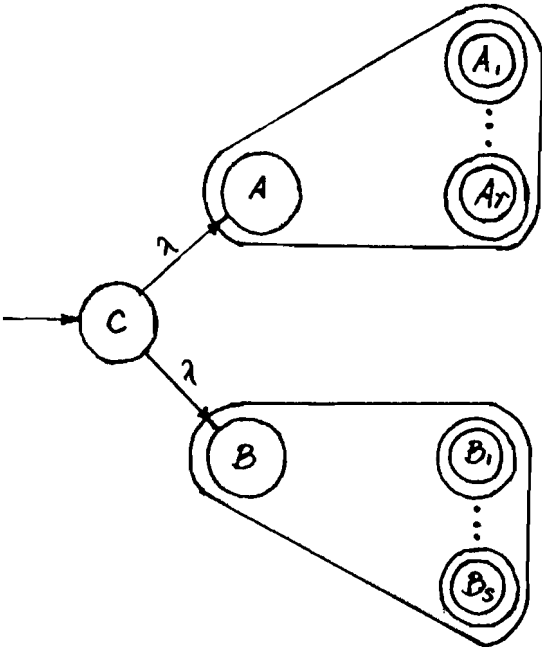
$$L(M_2) = F$$

FIGURE 4-4

Assume now that the regular languages E and F are $L(M_1)$ and $L(M_2)$ respectively, where both M_1 and M_2 are NFA. We designate M_1 and M_2 schematically as shown in Fig. 4-4.

Fig. 4-5 represents a NFA that accepts precisely the

language designated by $E + F$, i.e. $L(M_1) \cup L(M_2)$.



$$M = (S, \Sigma, \Delta, s, F)$$

$$S = S_1 \cup S_2 \cup \{C\}$$

$$\Sigma = \Sigma_1 \cup \Sigma_2$$

$$F = F_1 \cup F_2$$

$$\Delta = \Delta_1 \cup \Delta_2 \cup$$

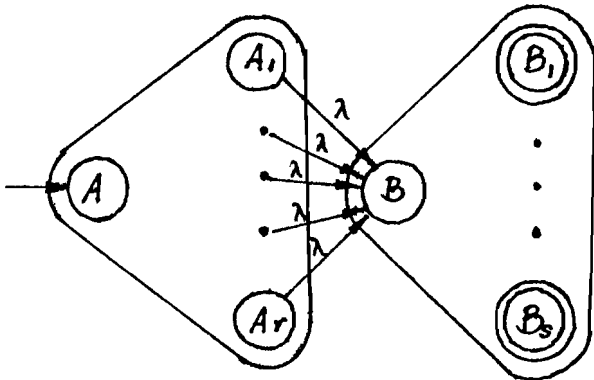
$$\{(C, \lambda, A), (C, \lambda, B)\}$$

$$s = C$$

$$L(M) = E + F$$

FIGURE 4-5

Fig. 4-6 illustrates a NFA that accepts exactly the language designated by EF , $L(M_1)L(M_2)$.



$$M = (S, \Sigma, \Delta, s, F)$$

$$S = S_1 \cup S_2$$

$$\Sigma = \Sigma_1 \cup \Sigma_2$$

$$\Delta = \Delta_1 \cup \Delta_2 \cup$$

$$\{(A_1, \lambda, B), \dots, (A_r, \lambda, B)\}$$

$$s = A$$

$$F = \{B_1, \dots, B_s\}$$

$$L(M) = EF$$

FIGURE 4-6

Finally, we must obtain a NFA that accepts exactly the

language designated by E^* . Fig. 4-7 illustrates such a NFSA.

This completes the inductive proof that every language represented by a regular expression is accepted by some NFSA. That is, every regular language is accepted by some NFSA.

Now let us prove the other part of the theorem.

Proof. (If) Let $L = L(M)$, where $M = (S, \Sigma, \Delta, s, F)$ is a NFSA.

We suppose that $S = \{q_1, q_2, \dots, q_n\}$ and $s = q_1$. We then set

$$R_{ij} = \{w \mid (q_i, w) \xrightarrow[M]{*} (q_j, \lambda)\} \text{ where } w \in \Sigma^*,$$

and it is then clear that

$$L = \bigcup \{R_{ij} \mid q_j \in F\}.$$

Hence, if we can prove that each R_{ij} , with $i, j = 1, 2, \dots, n$, is regular, we can conclude that L , being a finite union of such sets, is itself regular.

To prove the regularity of the R_{ij} 's, we introduce the auxiliary sets

$R_{ij}^k = \{w \mid w \text{ leads } M \text{ from } q_i \text{ to } q_j \text{ without passing through any state other than } q_1, q_2, \dots, q_k \text{ in between}\}$

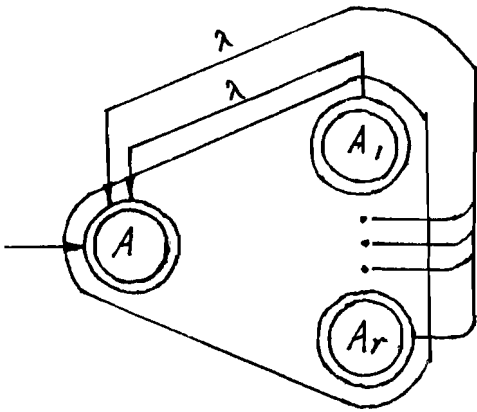
$$= \{w_s w_t \mid (q_i, w_s) \xrightarrow[M]{*} (q_r, \lambda) \text{ implies } 1 \leq r \leq k \text{ if } 1$$

$$\leq |w_t|, \text{ and } (q_i, w_s w_t) \xrightarrow[M]{*} (q_j, \lambda)$$

where $w_s, w_t \in \Sigma^*$.

We note immediately that $R_{ij}^0 = \{ w \in \Sigma^* \mid (q_i, w) \xrightarrow{*} (q_j, \lambda) \}$.

Because Δ is finite, R_{ij}^0 is finite and hence regular, while $R_{ij}^n = R_{ij}$. Hence, if we can prove, by induction on k , that each R_{ij}^k is regular, we are done. The basis step is secure, and it only remains to prove the validity of the induction step.



$$M = (S, \Sigma, \Delta, s, F)$$

$$S = S_1$$

$$\Sigma = \Sigma_1$$

$$\Delta = \Delta_1 \cup$$

$$\{ (A_1, \lambda, A), \dots, (A_r, \lambda, A) \}$$

$$s = A$$

$$F = F_1 \cup \{ A \}$$

FIGURE 4-7

Suppose, then, that we know R_{ij}^k to be regular for all i, j . We must prove that each R_{ij}^{k+1} is also regular. Now, consider (referring to Fig. 4-8) some w in R_{ij}^{k+1} . Either it never

reaches q_{k+1} , or else it can be broken up into segments $w_1 w_2 \dots w_m$, $m \geq 2$, where w_1 takes us from q_i to q_{k+1} via $\{ q_1, \dots, q_k \}$, and w_m takes us from q_{k+1} to q_j via $\{ q_1, \dots, q_k \}$, while each remaining w_r (if $m > 2$) takes us from q_{k+1} back to q_{k+1} via $\{ q_1, \dots, q_k \}$.

In short, we may write

$$R_{ij}^{k+1} = R_{i,k+1}^k (R_{k+1,k+1}^k)^* R_{k+1,j}^k \cup R_{ij}^k.$$

Since, by hypothesis, each of $A = R_{i,k+1}^k$, $B = R_{k+1,k+1}^k$, $C = R_{k+1,j}^k$, and $D = R_{ij}^k$ is regular, we deduce that $E = B^*$ is regular. Hence $F = AE$ is regular, and so $G = FC$ is regular, so that, finally,

$$R_{ij}^{k+1} = G \cup D$$

is regular. The regularity of L now follows.

Because of the equivalence of NFSA and DFSA, the result of the theorem follows as a corollary for DFSA.

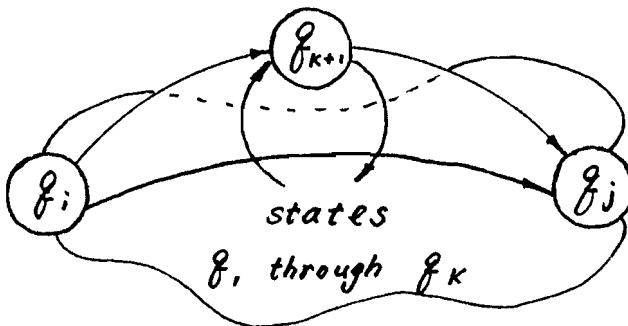


FIGURE 4-8

Chapter 5 Minimization

Section 5-1. Introduction

In this chapter, we shall first show what the inaccessible states of a DFSA are by an example. Then, we shall introduce an algorithm to eliminate the inaccessible states. Moreover, we shall introduce a more complicated algorithm to eliminate the structurally redundant states. To illustrate the algorithm we shall introduce the concept of congruence. Then in the last part of this chapter, we shall prove that the minimized DFSA is not only equivalent to the original but also the minimum.

Section 5-2. Elimination of Inaccessible States

Definition 5-1 Let M be a DFSA. We say that a state q of M is inaccessible if there is no input string x such that

$$(q_0, x) \xrightarrow{*} (q, \lambda).$$

Here q_0 is the initial state of M .

In other words, a state q of M is inaccessible if, beginning with the initial state q_0 , the machine will never

reach the state q , no matter what the input may be. For example, the state q_1 of the machine in Fig. 5-1 is inaccessible.

The following result is quite clear.

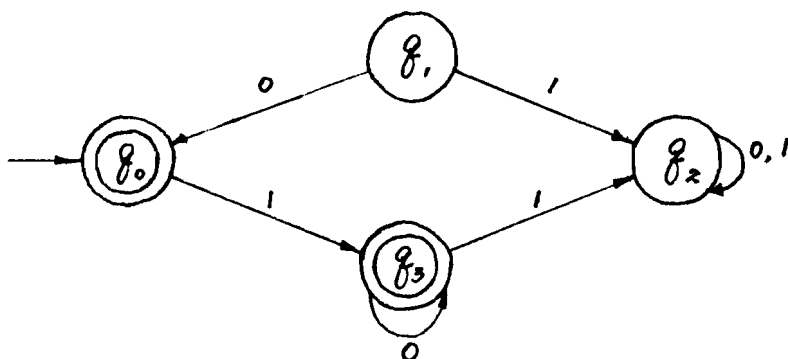


FIGURE 5-1

Theorem 5-1 Let M_1 be a DFSA and let M_2 be obtained from M_1 by removing all of the inaccessible states. Then $L(M_1) = L(M_2)$.

Given a DFSA M , all of its inaccessible states can be found by using the following algorithm.

Algorithm 5-1 Inaccessible States of a DFSA

Input: A DFSA $M = \{ S, \Sigma, s, \delta, F \}$.

Output: Collection I of Inaccessible States.

First, the collection of accessible states is constructed as follows. We form a sequence $\{ A_n \}$ of sets of states of M according to these rules:

1. $A_0 = \{ q_0 \}$, where q_0 is the initial state of M .

2. Suppose A_k is already constructed for some $k \geq 0$.

Form the set A_{k+1} by adding to A_k all the states of M which are accessible from A_k in a single move. That is,

$$A_{k+1} = A_k \cup \{q: \text{for some } p \text{ in } A_k \text{ and some } x \text{ in } \Sigma, \delta(p, x) = q\}.$$

3. If $A_k = A_{k+1}$, that is, if no new states are added to A_k , we stop and set $A = A_k$. Otherwise, go back to step 2.

Since there are only finite number, say n , of states in M , this process will eventually terminate in at most $n-1$ iterations. The set I of inaccessible sets is now obtained from S by removing from it all the elements of A , that is, $I = S \setminus A$.

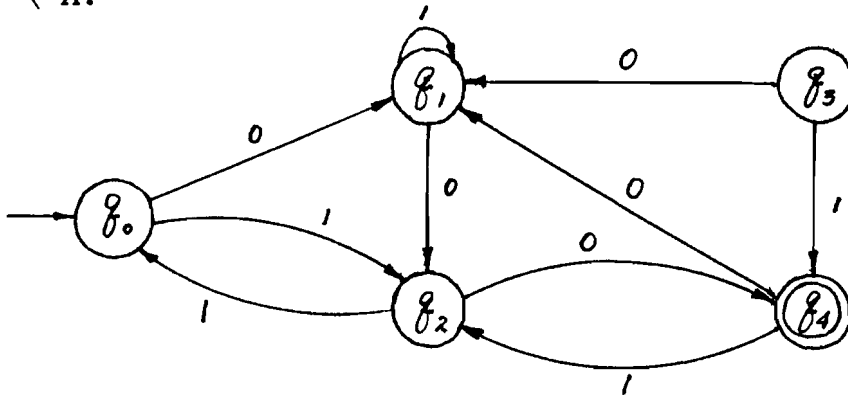


FIGURE 5-2

Example 5-1 Let M be the DFSA given by the state diagram in Fig. 5-2. The construction proceeds as follows.

$A_0 = \{q_0\}$ by definition.

$A_1 = \{q_0\} \cup \{q_1, q_2\} = \{q_0, q_1, q_2\}$, since

$\delta(q_0, 0) = q_1$ and $\delta(q_0, 1) = q_2$.

$A_2 = A_1 \cup \{ q_4 \} = \{ q_0, q_1, q_2, q_4 \}$, since $\delta(q_2, 0) = q_4$.

$A_3 = A_2$ since no new states can be reached from A_2 .

Thus, $A = A_2 = \{ q_0, q_1, q_2, q_4 \}$ and $I = \{ q_3 \}$ is the collection of inaccessible states. The state q_3 may be removed from the machine M .

Section 5-3. Elimination of Structurally Redundant States

Now we assume that every DFSA has no inaccessible states and concentrate on eliminating "structurally redundant" states. The procedure for doing it is based on the concept of congruence.

Definition 5-2 Let $M = \{ S, \Sigma, s, \delta, F \}$ be a DFSA and let k be a nonnegative integer. We say two states q and q' are k -congruent if the following is true:

Let x be any input string of length at most k , and suppose that $(q, x) \xrightarrow{*} (p, \lambda)$, $(q', x) \xrightarrow{*} (p', \lambda)$. Then

either both p and p' are final, or both p and p' are not final.

If q and q' are k -congruent, we denote this by $q \equiv_k q'$.

If q and q' are k -congruent for all $k = 0, 1, \dots$, we say that q and q' are congruent and denote it by $q \equiv q'$.

The heuristic meaning of k -congruent is as follows. Suppose t is a string to be tested for membership in $L(M)$. We start the machine M in configuration (q_0, t) . M moves along until it reaches configuration (q, r) , where r is a string of length at most k . If q and q' are k -congruent we may change the state from q to q' and let it run from there (on the input r), without changing the ultimate result. If, starting from configuration (q, r) , the machine will end up in a final state, the same would be true if the machine started in configuration (q', r) . Suppose now q and q' are congruent. We then may combine q and q' into a single state. It will have no effect on the set of strings accepted by M . thus, to minimize a DFSA M , we must determine which states are congruent to each other. It is clear from the definition that

1. If $q_1 \equiv_k q_2$ then $q_2 \equiv_k q_1$ (Symmetric).
2. For any state q we have $q \equiv_k q$ (Reflexive).
3. If $q_1 \equiv_k q_2$ and $q_2 \equiv_k q_3$ then $q_1 \equiv_k q_3$ (Transitive).

Thus the relation \equiv_k is an equivalence relation. It is also clear that the statements 1, 2, and 3, above, remain true if \equiv_k is replaced by \equiv , so the congruence \equiv is also an equivalence relation among the states of M . The task of minimization of the machine M consists essentially of dividing all the states of M into equivalence classes; any two states

within one class will be equivalent to each other, and any two states from different classes will not be equivalent. The minimized machine will then have these equivalence classes as its states. It also turns out that this process yields the smallest possible machine equivalent to the original one, that is, the one with the fewest states.

Before presenting an algorithm for doing this, we need two lemmas.

Lemma 5-1 If two states of a DFSA M are $(k+1)$ -congruent, they are also k -congruent.

The proof is immediate from the definition.

Lemma 5-2 Let $M = \{ S, \Sigma, s, \delta, F \}$ be a DFSA. Let k be a nonnegative integer and let G_1, G_2, \dots, G_n be the partition of the states of M into k -congruent classes: Two states from each class are k -congruent each other, and two states from different classes are not k -congruent. Let G be one of these classes and let q_1, q_2, \dots, q_m be the states of G . For each symbol $x \in \Sigma$ and each state $q \in G$ let $H_k(q, x)$ be the class G_i containing $\delta(q, x)$. Then the states q and q' from G are $(k+1)$ -congruent if and only if

$$H_k(q, x) = H_k(q', x) \text{ for each } x \in \Sigma \quad (*)$$

Proof. Suppose q and q' are two states belonging to the same class G and suppose that the equation $(*)$ is true. We want to show that q and q' are $(k+1)$ -congruent, that is, if σ is a string of length at most $k+1$, $(q, \sigma) \xrightarrow{*} (p, \lambda)$ and $(q', \sigma) \xrightarrow{*} (p', \lambda)$, then either both p and p' are final states, or both of them are not final states. If the string σ is of length at most k , we are done, since by hypothesis q and q' are in the same \equiv_k -equivalence class G , hence, $q \equiv_k q'$. Suppose then that σ has length $k+1$, that is, $\sigma = x\tau$, where $x \in \Sigma$ and τ is a string of length exactly k . But then

$$(q, \sigma) = (q, x\tau) \xrightarrow{*} (q_1, \tau) \xrightarrow{*} (p, \lambda) \text{ and}$$

$$(q', \sigma) = (q', x\tau) \xrightarrow{*} (q'_1, \tau) \xrightarrow{*} (p', \lambda)$$

and both q_1 and q'_1 belong to the same class G' (because equation $(*)$ holds, $H_k(q, x) = H_k(q', x)$). Thus $q_1 \equiv_k q'_1$ and since τ has length k , we see that p and p' are either both final or both not final.

Conversely, suppose that q and q' are two $(k+1)$ -congruent states, we want to prove that equation $(*)$ is true. Indeed,

if $H_k(q, x) \neq H_k(q', x)$ for some $x \in \Sigma$, and as we have known $q_1 = \delta(q, x)$, $q'_1 = \delta(q', x)$, then $q_1 \in H_k(q, x)$ and $q'_1 \in H_k(q', x)$ are not k -congruent. Therefore there is a string τ of length at most k , and $(q_1, \tau) \xrightarrow{*} (p, \lambda)$, and $(q'_1, \tau) \xrightarrow{*} (p', \lambda)$, such that $p \in F$ and $p' \in S \setminus F$ or $p' \in F$ and $p \in S \setminus F$. That is, there is a string $x\tau$ of length at most $k+1$, and $(q, x\tau) \xrightarrow{*} (p, \lambda)$, $(q', x\tau) \xrightarrow{*} (p', \lambda)$, such that $p \in F$ and $p' \in S \setminus F$ or $p' \in F$ and $p \in S \setminus F$. So this contradicts the hypothesis that q and q' are $(k+1)$ -congruent.

When $k = 0$ the division of the states into 0-congruent is particularly simple. Two states are 0-congruent if and only if they are either both final or both nonfinal. Thus, for $k = 0$, there are two equivalence classes: G_1^0 , all the nonfinal states, and G_2^0 , all the final states. The algorithm for dividing up the states into equivalence classes will basically operate as follows. First we divide the states of M into two groups G_1^0 and G_2^0 , the equivalence classes for the relation \equiv_0 . Next,

using Lemma 5-2, we divide each one of these into further subgroup; the resulting partition will form the equivalence classes for \equiv_1 . The operation is repeated over and over again, until no new subdivisions occur. The formal, precise description of this algorithm is as follows.

Algorithm 5-2 Minimal DFSA

Input: A DFSA $M = \{ S, \Sigma, s, \delta, F \}$.

Output: A DFSA K , with the fewest possible states, such that $L(K) = L(M)$.

1. Construction of states of K . The states of K will be the equivalence classes of the states of M under the equivalence relation \equiv . That is, the states of K are the sets G_1, G_2, \dots, G_r of states of M such that q_1, q_2 are members of the $G_i, i=1, 2, \dots, r$, if and only if $q_1 \equiv q_2$. The construction of these classes is as follows.

(1) Let G_1^0 be the set of all nonfinal states and G_2^0 be the set of all final states of M .

(2) Suppose the sets of M have been split into equivalence classes

$$G_1^k, G_2^k, \dots, G_m^k \quad (**)$$

under the relation \equiv_k . For each state q and each $x \in \Sigma$ let $H_k(q, x)$ be the class G_i^k that contains $\delta(q, x)$. Subdivide

each of the sets G_i^k of (**) further as follows. Two states q and q' of G_i^k will belong to the same class if and only if $H_k(q, x) = H_k(q', x)$ for any x in Σ . Let the resulting partition of the states of M be

$$G_1^{k+1}, G_2^{k+1}, \dots, G_n^{k+1} \quad (***).$$

(3) If the sets in (***) are identical with those in (**), then stop. The desired partition into equivalence classes has been obtained. If there are more classes in (***) than in (**), then go back to step (2).

2. The alphabet of K is the same as the alphabet of M .

3. The initial state of K is the equivalence class containing the initial state q_0 of M .

4. The transition function of K is defined as follows: Let G_1, G_2, \dots, G_r be the states of K . These are actually equivalence classes for that k for which the equivalence classes in (***) are identical with those of (**). From the construction in part 1, it follows that if q and q' are in the same G_i , then $H_k(q, x) = H_k(q', x)$ for all x in Σ . Thus the transition function of K

$$\delta'(G, x) = H_k(q, x) \text{ for any } q \text{ in } G$$

is well defined, that is, independent of q .

5. A state G is a final state if and only if it consists of final states of M .

We illustrate Algorithm 5-2 in the following example.

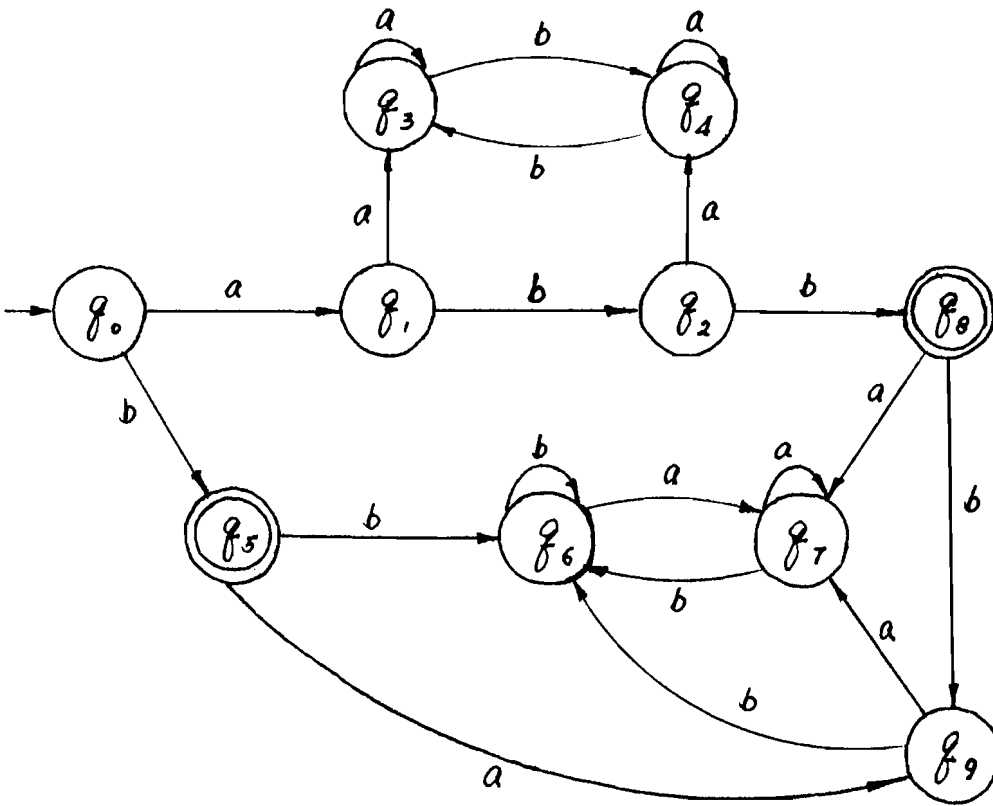


FIGURE 5-3

Example 5-2 Let M be the DFSA given by the diagram in Fig. 5-3. The class G_1^0 (all nonfinal states) is

$$\{ q_0, q_1, q_2, q_3, q_4, q_6, q_7, q_9 \}.$$

The class G_2^0 (all final states) is $\{ q_5, q_8 \}$. The next subdivision is obtained by considering the function $H_0(q, x)$ = the class G_1^0 containing $\delta(q, x)$. Thus for $k=1$ the class G_1^0 splits into two subclasses

$$G_1^1 = \{ q_0, q_2 \} \text{ and } G_2^1 = \{ q_1, q_3, q_4, q_6, q_7, q_9 \}.$$

The class G_2^0 does not split (q_5 and q_8 produce identical rows) and it becomes G_3^1 . Further splitting is summarized by the table in Table 5-1. We have omitted G's and q's, retaining only the relevant subscripts. Thus, G_3^3 is entered as 3 and q_5 as 5. The portion of $k=0$ in Table 5-1 is identical to Table 5-2. No new subclasses are introduced during $k=3$, so the equivalence classes of M, and thus the states of K, are

$$G_1 = \{ q_0 \}, G_2 = \{ q_2 \}, G_3 = \{ q_1 \},$$

$$G_4 = \{ q_3, q_4, q_6, q_7, q_9 \}, G_5 = \{ q_5, q_8 \}.$$

The initial state of K is G_1 and the final state of K is G_5 .

The transition function δ' is essentially given by the function H_3 in Table 5-1. For example,

$$\delta'(G_4, a) = G_4$$

since $H_3(q_3, a) = H_3(q_4, a) = H_3(q_6, a) = H_3(q_7, a) = H_3$. In

exactly the same way $\delta'(G_5, b) = G_4$ since

$$H_3(q_5, b) = H_3(q_8, b) = G_4.$$

Finally, the state diagram of K is given in Fig. 5-4.

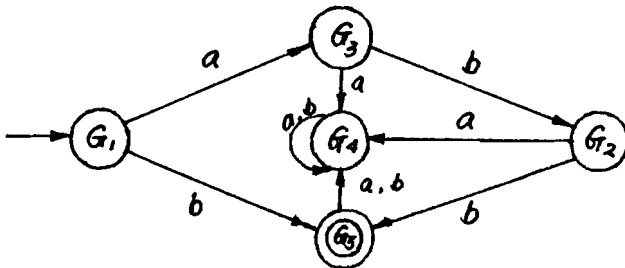


FIGURE 5-4

We conclude this section by showing that Algorithm 5-2 works correctly.

Theorem 5-2 Let M and K be DFSA from Algorithm 5-2. Then $L(K) = L(M)$. Moreover, if K_1 is another DFSA such that $L(K_1) = L(M)$ then the number of the states of K_1 is at least as large as the number of the states of K .

Proof. Let q and q' be two states of M and let G and G' be the states of K such that $q \in G$ and $q' \in G'$. From the construction of K it follows that for any $x \in \Sigma$

$$\text{if } \delta(q, x) = q' \text{ then } \delta'(G, x) = G' \quad (5-1).$$

Let $\sigma = x_1x_2\dots x_p$ be a string accepted by M . The sequence of moves of M on input σ is

$$q_0 \xrightarrow{x_1} q_{i_1} \xrightarrow{x_2} q_{i_2} \xrightarrow{x_3} \dots \xrightarrow{x_p} q_{i_p} \quad (5-2)$$

where q_{i_p} is a final state of M . Let $G_{i_1}, G_{i_2}, \dots, G_{i_p}$ be the states of K such that $q_{i_j} \in G_{i_j}$. It follows from 5-1 that the sequence of moves of K is

$$G_0 \xrightarrow{x_1} G_{i_1} \xrightarrow{x_2} G_{i_2} \xrightarrow{x_3} \dots \xrightarrow{x_p} G_{i_p}.$$

Since $q_{i_p} \in G_{i_p}$, the state G_{i_p} is a final state of K , so $\sigma \in L(K)$. Conversely, if σ is not in $L(M)$ then the state q_{i_p} in

5-2 is not a final state of M , and again, since $q_{i_p} \in G_{i_p}$, the state G_{i_p} is not a final state of K , so $\sigma \notin L(K)$. Thus $L(K) = L(M)$.

To show that the machine K has the fewest possible states, we argue as follows. Let K_1 be any DFSA with fewer states than K . We shall show that $L(K_1) \neq L(M) = L(K)$.

Since all the states of M are accessible, all the states of K are also accessible. Thus, for every G of K there is a

$\sigma(G) \in \Sigma^*$ such that $(G_0, \sigma(G)) \xrightarrow[K]{*} (G, \lambda)$, where G_0 is the

initial state of K . Let p_0 be the initial state of K_1 , and consider the movements of K_1 on the input $\sigma(G)$ for all possible G of K . For each such G the machine K_1 will end up in some configuration $(p(G), \lambda)$ for some state $p(G)$ of K_1 . Since K_1 has fewer states than K , there are going to be two distinct states of K , say G' and G'' , such that $p(G') = p(G'')$. In other words, there are two distinct strings $\sigma(G')$ and $\sigma(G'')$ such that

1. On the input of both $\sigma(G')$ and $\sigma(G'')$ the machine K_1 moves from p_0 to the same state p .

2. On the input of $\sigma(G')$ and $\sigma(G'')$ the machine K moves from G_0 to two distinct states G' and G'' .

Let q' and q'' be the states of M , and $q' \in G'$, $q'' \in G''$. Since $G' \neq G''$, the states q' and q'' are not congruent. Thus for some input τ , the machine M will move from q' to q_r and from q'' to q_s where one of the states q_r and q_s is final and the other one is not. This implies that one of the strings $\sigma(G')\tau$ and $\sigma(G'')\tau$ is in $L(M)$ while the other one is not. Consider however, what happens when these strings are fed into the machine K_1 . On the input $\sigma(G')\tau$ the machine K_1 moves from the configuration $(p_0, \sigma(G')\tau)$ to (p, τ) . Similarly, on the input $\sigma(G'')\tau$ the machine K_1 moves from $(p_0, \sigma(G'')\tau)$ to the same configuration (p, τ) and then proceeds further, until the string τ is exhausted. Thus, K_1 must either accept both strings $\sigma(G')\tau$ and $\sigma(G'')\tau$ or reject both of them. Thus $L(K_1) \neq L(M)$, since the machine M accepts exactly one of the strings.

Chapter 6 Conclusions

The results of the thesis establish that the regular languages are closed under a variety of operations and that regular languages can be specified either by regular expressions or by NFSA or DFSA. These facts, used singly or in combinations, provide a variety of techniques for showing languages to be regular. Moreover, we have also shown that although NFSA appear to be more general than DFSA, they are nevertheless no more powerful in terms of the languages they accept: A NFSA can always be converted into a DFSA. And the minimized DFSA can be found by applying the algorithms given in Chapter 5.

55

G^0	H_0			G^1	H_1			G^2	H_2			G^3	H_3		
	f	a	b		f	a	b		f	a	b		f	a	b
1	0	1	2	1	0	2	3	1	0	2	4	1	0	3	5
	1	1	1		2	2	3		2	3	4		2	2	4
	2	1	2	2	1	2	1	2	1	3	1	3	1	4	2
	3	1	1		3	2	2	3	3	3	3	3	4	4	
	4	1	1		4	2	2	4	3	3	4	4	4	4	
	6	1	1		6	2	2	3	6	3	3	4	5	4	4
	7	1	1		7	2	2		7	3	3		7	4	4
	9	1	1		9	2	2		9	3	3		9	4	4
2	5	1	1	3	5	2	2	4	5	3	3	5	5	4	4
	8	1	1		8	2	2		8	3	3		8	4	4
$K=0$				$K=1$				$K=2$				$K=3$			

Table 5-1

G_i°	f	H_o	
		$H_o(f.a)$	$H_o(f.b)$
G_{11}°	f_0	G_{11}°	G_{12}°
	f_1	G_{11}°	G_{11}°
	f_2	G_{11}°	G_{12}°
	f_3	G_{11}°	G_{11}°
	f_4	G_{11}°	G_{11}°
	f_6	G_{11}°	G_{11}°
	f_7	G_{11}°	G_{11}°
	f_9	G_{11}°	G_{11}°
	G_{12}°	f_5	G_{11}°
f_8		G_{11}°	G_{11}°

Table 5-2

REFERENCES

- [1] Harry R. Lewis, Christos H. Papadimitriou, ELEMENTS OF THE THEORY OF COMPUTATION. Prentice-Hall, Englewood Cliffs, N.J., 1981.
- [2] Leonard S. Bobrow, Michael A. Arbib, DISCRETE MATHEMATICS. W. B. Saunders, Philadelphia, P.A., 1974.
- [3] John E. Hopcroft, Jeffrey D. Ullman, FORMAL LANGUAGES AND THEIR RELATION TO AUTOMATA. Addison-Wesley, Reading, M.A., 1969.
- [4] Peter J. Denning, Jack B. Dennis, Joseph E. Qualitz, MACHINES, LANGUAGES, AND COMPUTATION. Prentice-Hall, Englewood Cliffs, N.J., 1978.
- [5] Zamir Bavel, INTRODUCTION TO THE THEORY OF AUTOMATA. Reston Publishing, Reston, V.A., 1983.
- [6] Vladimir Drobot, FORMAL LANGUAGES AND AUTOMATA THEORY. Computer Science Press, Rockville, M.D., 1989.
- [7] Paul F. Dierker, William L. Voxman, DISCRETE MATHEMATICS. Harcourt Brace Jovanovich, San Diego, C.A., 1986.

TO: All Graduate Students Who Submit a Thesis or Research Problem/Project as Partial Fulfillment of The Requirements for an Advanced Degree

FROM: Emporia State University Graduate School

I, Jun Wu, hereby submit this thesis/report to Emporia State University as partial fulfillment of the requirements for an advanced degree. I agree that the Library of the University may make it available for use in accordance with its regulations governing materials of this type. I further agree that quoting, photocopying, or other reproduction of this document is allowed for private study, scholarship (including teaching) and research purposes of a nonprofit nature. No copying which involves potential financial gain will be allowed without written permission of the author.

Jun Wu
Signature of Author

Dec. 1992
Date

Finite State Automata And Regular Languages
Title of Thesis/Research Project

Douglas K. Cooper
Signature of Graduate Office Staff Member

Dec 11, 1992
Date Received

Distribution: Director, William Allen White Library
Graduate School Office
Author