THE UNIVERSITY OF MICHIGAN

SYSTEMS ENGINEERING LABORATORY

Department of Electrical Engineering
College of Engineering

SEL Technical Report No. 8

LECTED TOPICS IN AUTOMATA THEORY

by

Richard M. Karp

September 1966

This research was supported by United States Air Force Contract AF 30(602)-3546.

Preface

During the 1964-65 academic year the staff of the Systems Engineering Laboratory undertook a systematic review of automata theory and its applications, in an effort to identify significant research problems. This report represents a part of that study. The contents of this report were presented by the author as a course (Communication Sciences 622) in the Communication Sciences Program at The University of Michigan during the spring of 1965. The author wishes to acknowledge the helpful suggestions of the students in that course, and to thank Professors H. L. Garner and E. L. Lawler for their advice and encouragement.

TABLE OF CONTENTS

1. INTRODUCTION

Automata theory is a young mathematical discipline dealing with conceptual models of information processing devices and methods. It is the purpose of this report to give a formal development of the properties of some of these models, and to discuss their relationships to some possible areas of application. Automata theory has its origins in several disciplines. One important model, the finite-state sequential machine, was originated by engineers seeking systematic methods for the analysis and synthesis of digital circuits. Other links with engineering include the similarity between communication channels and probabilistic sequential machines, and the use of abstract state-transition models in the study of control systems and sequential decision processes. There is a close relationship between formal generative grammars, devised by theoretical linguists and computer programmers to define the sentences of natural or artificial languages, and automata which distinguish sentences of a language from non-sentences. Finally, the Turing machine was invented in response to the inner needs of mathematics itself; Turing machines provide one satisfactory means of formalizing the intuitive notions of "effectively calculable function" and "effectively decidable proposition."

We shall not attempt to give a general definition of the term "automaton." Instead, we shall illustrate the concept by informally discussing the two most widely studied models: finite-state sequential machines and Turing machines. We shall then point out some features common to these two models, and briefly mention other related models.

The most direct motivation for introducing sequential machines is to provide a means of describing formally the structure and behavior of sequential switching circuits. Since the definition of a sequential circuit requires some preliminaries, we shall postpone it, and shall instead consider a simplified model of digital computer operation. Consider a digital computer with a fixed, finite amount of internal storage (core memory, tapes, registers, switches, etc.) whose input data is in the form of a stack of punch cards. The computer proceeds by reading a card, performing some computations, (which amount to a change of its "internal state"), reading the next card, and so on until the computer receives an indication that all the cards have been read. The computer then prints out a single "record," consisting of some of the data contained in its memory.

We can model this type of computer operation as follows: a finite-state sequential machine is specified by a sextuple $(A_I, Q, A_o, q_0, \lambda, \delta)$ where:

A_I is a finite set of input symbols

Q is a finite set of internal states

A_o is a finite set of output symbols

$q_0 \in Q$ is the initial state

λ is the transition function $\lambda: Q \times A_I \to Q$

δ is the output function $\delta: Q \to A_o$.

In modelling a digital computer, these entities may be interpreted as follows:

A_I: the set of possible configurations of holes in a punched card

Q : the set of all possible internal configurations of the computer (Thus, for an IBM 7090, the approximate number of states is $(2)^{2^{20}}$.)

A_o: the set of records that may possibly be printed out

q_0: the initial internal state of the computer

λ : If the computer is in the state q and reads the input a, its deterministic internal operations take it to the state $\lambda(q,a)$ just before reading the next card.

δ : $\delta(q)$ is the record printed out if the computer arrives at state q.

It is evident that certain important considerations about digital computers (the distinction between instructions and data, the subdivision into arithmetic unit, control unit, input-output units, and memory units with various characteristics, etc.) are not captured in this general model, and that the very large number of states is likely to preclude the use of the model for the exact description of any given computer. Still, it appears that general theorems about the computations that sequential machines represent could have implications concerning computers.

The second model we shall describe also has bearing on digital computers, but it existed before computers did. Beginning in the 1930's several mathematicians (including Church, Kleene, Turing, Post, and Markov) grappled with the problem of defining what is meant by the term "algorithm", or "effective calculation procedure." Their definitions of an effectively calculable (general recursive, computable, λ-definable) function are quite different in form, but the same in content. Turing's definition is based

on what are now called Turing machines. A Turing machine consists of a finite-state machine, together with a tape, ruled into a two-way infinite sequence of squares on which symbols may be written. The initial data, as well as all intermediate and final results of the calculation, are stored on the tape. The way in which the tape is read and the writing of symbols on the tape are controlled by the finite-state machine and the data it reads from the tape. A formal definition is as follows: a Turing machine is a sextuple $(A, Q, q_0, D, \omega, f)$ where

A	is a finite set of at least two symbols
Q	is a finite set of internal states
$q_0 \in Q$	is the initial state
$D = \{L,C,R\}$	is the set of directions of tape motion
ω	is a "halt" indicator
f	is a function $f: Q \times A \to \{\omega \cup A \times Q \times D\}$.

One symbol, $b \in A$, is designated as a "blank." When a Turing machine begins its operation, with some initial data on its tape, all but a finite number of squares contain the symbol b, and the rightmost "non-blank" square is being scanned (If every square is blank, it doesn't matter which square is being scanned.). Suppose, at some point in its operation, the Turing machine is in state q reading a square containing the symbol a. Then,

(a) if $f(q,a) = \omega$, the machine halts

(b) if $f(q,a) = (a', q', d')$ the machine

i) replaces the symbol a in the square being scanned by a'

ii) enters the state q'

iii) reads the square to the right of the present one if d' = R; reads the same square if d' = C; reads the square to the left if d' = L.

A Turing machine may be considered to represent a function. This function has the initial tape configuration (i.e. the unique shortest sequence of consecutive symbols bounded by the leftmost and rightmost non-blanks) as its argument, and the final tape configuration as its value; the function is undefined for initial tape configurations such that the Turing machine does not halt. With suitable conventions, a Turing machine can also be considered to represent a partial function $f(x_1,x_2,\ldots,x_n)$ whose arguments and values are nonnegative integers. To compute $f(a_1,a_2,\ldots,a_n)$, put the initial configuration $\bar{a}_1$ b $\bar{a}_2$ b ... b $\bar{a}_n$ on the tape, where bϵA, 1ϵA, and $\bar{a}$ represents a string of a+1 1's. The rightmost 1 is initially being scanned. If the Turing machine halts, then $f(a_1,a_2,\ldots,a_n)$ is equal to the number of occurrences of 1 in the final tape configuration; if the Turing machine does not halt, $f(a_1,a_2,\ldots,a_n)$ is undefined.

It is generally felt that the Turing machine (as well as the other equivalent formulations) gives a satisfactory solution to the problem of formalizing what is meant intuitively by an "effectively calculable function." Thus, it is an accepted proposition, but not a provable one, that any function that can be calculated at all can be calculated by a Turing machine. This proposition is called Church's thesis.

The finite-state sequential machine and the Turing machine have several properties in common. Both operate on symbols drawn from finite

alphabets, rather than symbols drawn from a continuum of values such as the real numbers. Thus, both are digital machines. Their basic operation steps, including the reading and writing of symbols, changes of internal state, and tape motion, occur in a discrete sequence, rather than continuously. Thus, they are discrete-time machines. Each basic operation depends on only a finite number of symbols, and affects only a finite number of symbols; the machines are therefore finitary. The sequence of steps, and the final results, are implicitly determined by the initial data. Thus, they are deterministic. The main respect in which finite-state machines and Turing machines differ is in the amount of memory they have The memory of a finite-state machine, as measured by its number of internal states, is fixed and finite. On the other hand, the tape of a Turing machine is available for the storage of intermediate results, and, usually, no a priori upper bound on the amount of tape a Turing machine will use can be given. Since, at any given time, the total amount of tape used is finite, it is misleading to call a Turing machine an infinite automaton; the terms "growing automaton" or "potentially infinite automaton" are preferred.

Our main concern in the present report will be with digital, discrete-time, finitary, deterministic automata. Accordingly, the main mathematical tools used will be algebra, combinatorial analysis, and one or two results from computability theory, rather than analysis and probability theory. Other types of automata are certainly of interest also. For example, the dynamical systems of classical mechanics are analog, continuous-time, deterministic automata; also, communication channels may be viewed as stochastic automata.

Most of the report will deal with finite-state sequential machines. We will develop characterizations of the functions representable by finite-state machines, methods of analysis and classification of such machines, and methods of realizing machines by interconnection of smaller machines. We then turn our attention to automata more powerful than finite-state machines, but less powerful than Turing machines. Since we accept Church's thesis, all the automata we use as models of information processing systems will be no more powerful than Turing machines; note, however, that we may find it technically useful to define automata which represent noncomputable functions (infinite-state sequential machines, for instance), but whose operation is not finitary, since it involves look-ups in infinite tables.

One simple example of such an intermediate automaton is a finite automaton which can look back and forth through its input tape (unlike a finite-state machine proper, which reads through the input sequence in single-pass fashion), but cannot write or erase symbols on the tape. Another variant is a linear bounded automaton, which operates like a Turing machine, except that it may write in or erase only those squares which originally contained symbols of the input sequence. A third type of machine is the pushdown automaton, which is a finite-state machine together with an infinite tape called the "pushdown" which is used subject to the following restrictions:

i) Initially there is only one square on the tape which is not blank; it contains a special symbol # , and is designated as the top of the pushdown.

ii) The only square that can be read is the top of the pushdown; upon being read it is erased (made blank), and the square to its left becomes the top.

iii) The only square that can be marked (written on) is directly to the right of the top; upon being marked, a square becomes the top.

When one studies the capabilities of pushdown automata and related intermediate automata, one discovers strong connections with the phrase structure grammars and associated phrase structure languages of mathematical linguistics. We can indicate the nature of these connections by means of an example. Consider the artificial language whose "sentences" are sequences of a's and b's such that:

i) the total number of b's exceeds the total number of a's by exactly 1;

ii) in any suffix of the sentence (obtained by deleting an initial sequence of symbols) the number of b's exceeds the number of a's by at least 1.

Some sentences:	Some non-sentences:
b	a
abb	bab
ababb	abbba

The set of sentences forms a so-called context-free language, which can be generated by the following context-free grammar:

$$A \rightarrow b$$

$$A \rightarrow aAA$$

A <u>derivation</u> in this particular grammar is a sequence $\varphi_0, \varphi_1, \ldots, \varphi_n$ of

words composed of symbols a,b,A such that $\varphi_0 = A$, φ_n does not contain A, and, for i=0,1,...,n-1, φ_{i+1} is obtained from φ_i by replacing an occurrence of A in φ_i either by b or by aAA.

Example:

φ_0	A
φ_1	aAA
φ_2	aAaAA
φ_3	abaAA
φ_4	ababA
φ_5	ababaAA
φ_6	ababaAb
φ_7	abababb

The set of sentences can also be recognized by a pushdown automaton which reads an input sequence of a's and b's from right to left and:

i) if b is read, a 1 is written in the pushdown

ii) if a is read, one symbol is read from the pushdown

iii) after the entire string has been read, two additional symbols are read from the pushdown.

An input sequence is recognized as a sentence if and only if each symbol read from the pushdown is a 1, except for the last symbol, which is a blank.

This example gives an indication of the connection between phrase-structure-languages and sentence-recognizing automata. More precise connections will be developed later in the report.

2. THE FUNCTIONS rp(x) AND res(x)

In this section we present some pictorial representations of finite-state sequential machines, and define and give some elementary properties of the functions rp(x) and res(x), which characterize the structure and behavior of a sequential machine. Our development will draw freely on [8] and [10].

Definition. A sequential machine is a system $\mathcal{M} = (A_I, Q, A_O, q_0, \lambda, \delta)$ where:

A_I	is a finite nonempty set	(input symbols)
Q	is a nonempty set	(internal states)
A_O	is a finite nonempty set	(output symbols)
$q_0 \in Q$		(initial state)
λ	is a function $\lambda: Q \times A_I \to Q$	(transition function)
δ	is a function $\delta: Q \to A_O$	(output function)

A sequential machine is said to be finite-state if Q is a finite set. Although we are interested mainly in finite-state machines, some of our early results will hold for infinite-state machines as well.

The definition we have given is known as the Moore model of a sequential machine. An alternate model, the Mealy model, is the same except that the output function δ has domain $Q \times A_I$ and range A_O. All results about the Moore model translate into closely analogous results about the Mealy model; the Mealy model will not occur in our formal development.

A finite-state sequential machine can be specified by tabulating the functions λ and δ.

Example 1

$A_I = \{0,1\}$

$Q = \{q_0, q_1, q_2, q_3\}$

$A_O = \{a,b\}$.

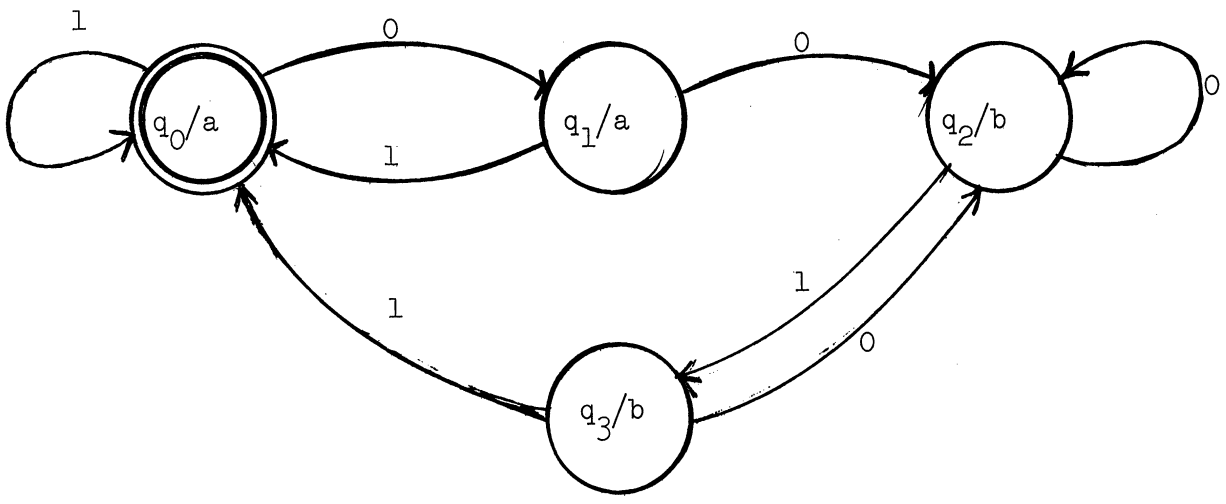
	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_3
q_3	q_2	q_0

Table for λ

q_0	a
q_1	a
q_2	b
q_3	b

Table for δ

An alternate representation employs a rooted, directed, labelled graph called a state diagram. The nodes are in one-to-one correspondence with the states. The node for state q is labelled $q/\delta(q)$. If $\lambda(q,a) = q'$, then there is a branch labelled "a" directed from the node associated with q to the node for q'. The node associated with q_0 is the root of the graph, and is given a distinguishing mark. The state diagram for the machine of Example 1 is:



From the state diagram, it is possible to visualize the response of the machine to sequences of inputs. For example, the input sequence 0010 takes the machine from the initial state successively to q_1, q_2, q_3, and q_2. The output produced at the end of this sequence is $\delta(q_2) = b$. In general, for the given machine, a sequence produces the output a if and only if the sequence does not contain two consecutive 0's, or it contains two consecutive 1's after the last pair of consecutive zeros.

Let us introduce some terminology, in order to be able to deal with the response of a machine to input sequences. Let A_I^* denote the set of all finite sequences of elements of A_I; including the null sequence, denoted by e. Sometimes A_I is referred to as the input alphabet, and the elements of A_I^* are called words in the input alphabet. Let A_I^+ denote the set of non-null sequences of elements of A_I. The length of a sequence x, denoted lg(x), is the number of symbols in the sequence x. Then

$$\lg(b_1, b_2 \cdots b_r) = r$$
$$\lg(e) = 0.$$

The concatenation of x and y, denoteed xy, is the sequence consisting of x followed by y; for example, if $x = b_1 \cdots b_r$ and $y = c_1 \cdots c_s$, then $xy = b_1 \cdots b_r\, c_1 \cdots c_s$. Note that

$$xe = ex = x$$
$$\lg(xy) = \lg x + \lg y$$
$$(xy)z = x(yz)$$

The algebraic system (A_I^+, concatenation) is a semigroup, and the

system (A_I^*, concatenation) is a semigroup with identity (monoid). Let us recall the definitions:

A <u>semigroup</u> is an algebraic system $(S,\cdot)$, where S is closed under the associative binary operation $\cdot$. The element $e \in S$ is a (two-sided) identity element if, for all $s \in S$, $es = se = s$. A <u>monoid</u> is a semigroup with a two-sided identity element.

> <u>Remark</u>. A semigroup has at most one (two-sided) identity element. The system (A_I^+, concatenation) is called the free <u>semigroup</u> generated by A_I, and the system (A_I^*, concatenation) is the <u>free monoid</u> generated by A_I.

We can now extend the domain of the transition function λ to $Q \times A_I^*$. The definition of the extended function is recursive.

For $x \in A_I^*$, $a \in A_I$;

$$\lambda(q,e) = q$$

$$\lambda(q,xa) = \lambda(\lambda(q,x),a).$$

An alternate recursive definition is:

$$\lambda(q,e) = q$$

$$\lambda(q,ax) = \lambda(\lambda(q,a),x).$$

<u>Lemma 2.1</u> The two recursive definitions are equivalent.

PROOF. We use induction on lg x.

<u>Basis</u>. The result certainly holds if $\lg x \leq 1$.

<u>Induction step</u>. Suppose the result holds whenever $\lg x < r$. Consider $x = a_1 a_2 \cdots a_r$. Then

$$\lambda(\lambda(q,a_1),\ a_2 \cdots a_r) =$$
$$\lambda(\lambda(\lambda(q,a_1),\ a_2 \cdots a_{r-1}),\ a_r) =$$
$$\lambda(\lambda(q,\ a_1 \cdots a_{r-1}),\ a_r).$$

Q.E.D.

Lemma 2.2

For all $q \in Q$, and for $x \in A_I^*$, $y \in A_I^*$, $\lambda(q,xy) = \lambda(\lambda(q,x),y)$.

The proof, which we omit, is by induction on lg x.

We define the response function rp(x) as follows: for all $x \in A_I^*$, $rp(x) = \lambda(q_0,x)$. In other words, rp(x) is the state the machine reaches upon application of the input sequence x. The domain of this function is A_I^*, and its range is Q.

Corollary 2.1

$$rp(xy) = \lambda(rp(x),y).$$

Corollary 2.2

If $rp(x) = rp(y)$, then, for all $z \in A_I^*$,

$$rp(xz) = rp(yz).$$

Another important function is the result function res(x): $A_I^* \to A_o$. This function is defined by: $res(x) = \delta(rp(x))$. This function gives the output obtained upon application of the input sequence x. The function res(x) is referred to as the function represented (defined) by the sequential machine $\mathcal{M}$. A basic question concerning the behavior of finite-state sequential machines is: what are necessary and sufficient conditions on a function f in order that it be represented by some finite-state sequential machine?

The proviso "finite-state" in this question is important, since every function f: $A_I^* \rightarrow A_o$ is represented by some infinite-state machine. A machine representing a given function f is obtained as follows. The input alphabet is A_I, and the output alphabet, A_o. The states are in one-to-one correspondence with the elements of A_I^*. The state corresponding to the word x is denoted $< x >$. The initial state is $< e >$, and the transition and output functions are defined as follows:

$$\lambda(< x >, a) = < x\ a >$$
$$\delta(< x >) = f(x).$$

It is easy to prove by induction that, for all x,

$$rp(x) = < x >.$$

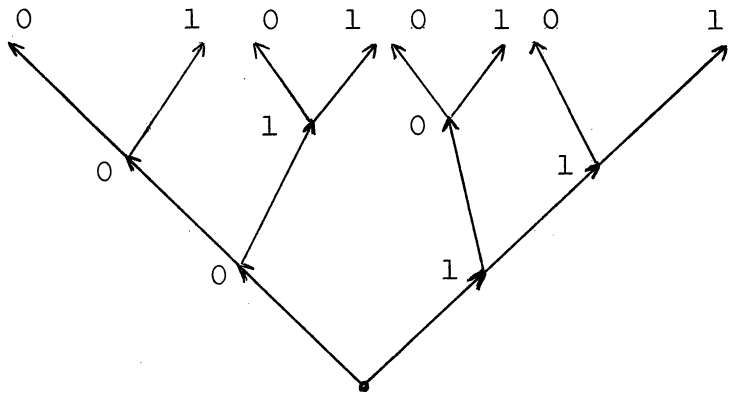
Then

$$res(x) = \delta(rp(x)) = \delta(< x >) = f(x),$$

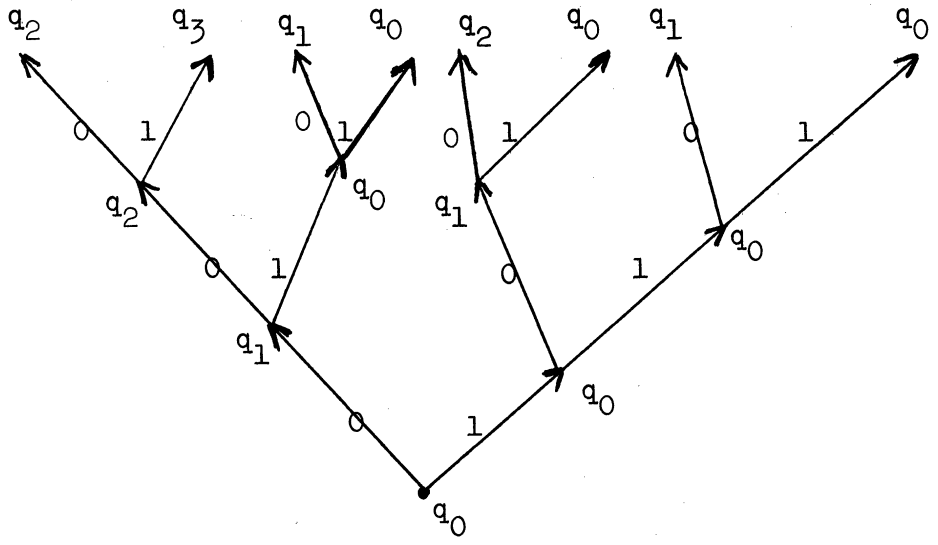
as desired.

3. THE RESPONSE TREE; ACCESSIBILITY

A convenient way to represent A_I^* pictorially is by means of the free tree generated by A_I. This is a rooted infinite directed tree, in which the branches directed out of any node are labelled in 1-1 correspondence with the elements of A_I. Part of the free tree for $A_I = \{0,1\}$ is shown below:



There is a one-to-one correspondence between A_I^* and the set of nodes; if $x \in A_I^*$, let its corresponding node be n_x. Then the root is n_e, and, if $x = a_1 a_2 \ldots a_n$, n_x is reached by following the successive branch labels $a_1, a_2, \ldots, a_n$ up from the root. The response tree for a sequential machine $\mathcal{M}$ with input alphabet A_I is obtained by labelling each node n_x with the state $rp(x)$. For the infinite-state machine discussed at the end of the last section, $rp(x) = \langle x \rangle$, so n_x is given the label $\langle x \rangle$. For this particular machine, but not in general, the response tree is a state diagram, except that the outputs are not shown. For the machine of Example 2.1, a part of the response tree is:



A state q is <u>accessible</u> if, for some x, q = rp(x). If q is accessible, let

$$d(q) = \min_{\{x \mid rp(x)=q\}} lg(x).$$

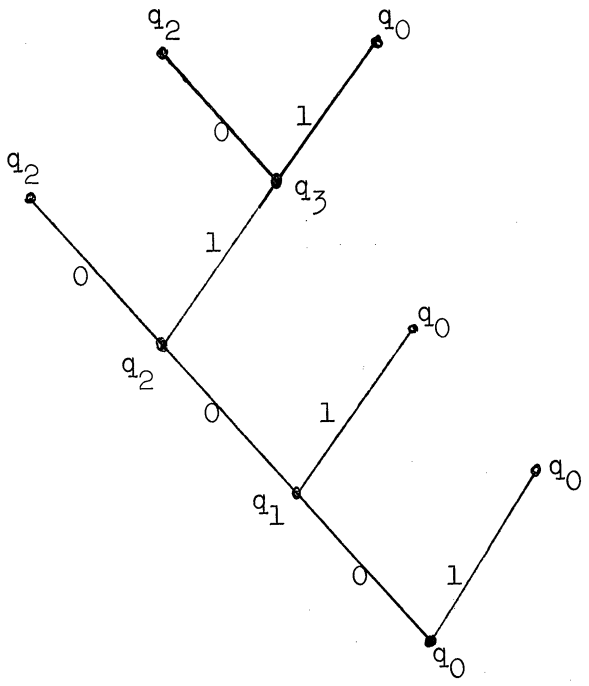
Evidently, q occurs as a label in the response tree for $\mathcal{M}$ if and only if q is accessible; the label q will appear at the $d(q)^{th}$ "level" of the response tree, and will not appear at any earlier level.

In general, the response tree for a machine contains redundant information, since the transitions out of a given accessible state may be exhibited more than once. This redundancy is eliminated in the <u>pruned</u> response tree for $\mathcal{M}$, which is constructed as follows. Starting with the root the tree is generated level-by-level, and from left to right within a given level. Each node already generated is examined, but the branches out of a node, and its successors at the next level, are generated if and only if the label of the node has not been encountered before. A unique left-to-right ordering of the nodes at a given level is induced by putting A_I in some alphabetic order; the branches out of a

node are arranged left to right in ascending alphabetic order of their labels.

Example 1:

The pruned response tree for the finite-state machine of Example 2.1 is:



The property that every accessible state is a node label is preserved even in the pruned response tree.

Lemma 1

The node label q occurs in the pruned response tree for $\mathcal{M}$ if and only if q is accessible. If q is accessible, the shortest path in the response tree from the root to a node labelled q is of length d(q).

PROOF: If q is not accessible, then there is no x such that $q = rp(x)$, and therefore no node is labelled q. To complete the proof, we show by induction on $d(q)$ that, if q is accessible, there is a node labelled q, and that the shortest path from the root to such a node is of length $d(q)$.

Basis. $d(q) = 0$; then $q = q_0 = rp(e)$. The root is labelled q_0.

Induction step. Suppose the result holds for $d(q) < r$. Consider q, with $d(q) = r$. Then there exists an ordered pair (q',a) such that $d(q') = r-1$ and $\lambda(q',a) = q$. By the induction hypothesis, there is a node n_y labelled q' at a distance r-1 from the root. Therefore, the first node labelled q' to be encountered in the construction of the pruned response tree is at a distance r-1 from the root (the distance is $\leq$ r-1, since it is encountered earlier than n_y in the construction process, and it cannot be < r-1 since, if $rp(z) = q'$, $lg(z) \geq r-1$); let this first node be n_z. Then there are branches out of n_z in the pruned response tree, which therefore contains the node n_{za_r}. But $rp(za_r) = \lambda(rp(z),a_r) = \lambda(q',a_r) = q$, and the node n_{za_r} is at a distance r from the root. Finally, the by definition of $d(q)$, no node n_y at a distance < r from the root can be labelled q.

Q.E.D.

Lemma 2

Let $\mathcal{M}$ have n accessible states. Then, if q is accessible, $d(q) \leq n-1$.

PROOF: Let x be a word of minimum length such that $rp(x) = q$. Suppose x can be written as $x = u\ v\ w$ where $v \neq e$ and $rp(u) = rp(uv)$. Then $rp(uw) = \lambda(rp(u),w) = \lambda(rp(uv),w) = rp(uvw) = q$, contradicting the minimality of x. If this contradiction is to be avoided, all the prefixes of x (including the null sequence) must give different responses, and each response must be an accessible state. Since the number of such states is n, x may have only n prefixes, and $lg(x) \leq n - 1$.

Q.E.D.

The pruned response tree for a finite-state machine can certainly be constructed with a finite number of operations. For, if A_I has k elements, and the number of accessible states of $\mathcal{M}$ is n, the response tree has kn branches, and, as an easy consequence of Lemma 2, every node is at a distance $\leq$ n from the root. Thus, we have an algorithm for determining the accessible states of a finite-state machine. Stating this result more precisely, we claim:

Corollary 1

There is an effective procedure for determining the accessible states of a finite-state sequential machine.

To prove this corollary quite rigorously, we would have to define a Turing machine to carry out this procedure, given a conventional representation of a finite-state machine as input data. We shall not be so fussy, and shall accept "effective procedure" more intuitively defined. More care will be required, of course, when we prove the nonexistence of algorithms for certain decision problems.

Some other results now follow quite easily.

Corollary 2

Let $\mathcal{M}$ be a finite-state sequential machine, and let $b \in A_o$. There is an algorithm to determine whether there exists $x \in A_I^*$ such that res(x) = b.

PROOF: The algorithm is as follows: list the accessible states of $\mathcal{M}$, and determine whether any accessible state q has $\delta(q) = b$.

Q.E.D.

Corollary 3

Let $\mathcal{M} = (A_I, Q, A_o, q_0, \lambda, \delta)$ and $\mathcal{M}' = (A_I, Q', A_o, q_0', \lambda', \delta')$ be finite-state machines. There is an algorithm to determine whether there exists $x \in A_I^*$ such that $\text{res}_{\mathcal{M}}(x) = \text{res}_{\mathcal{M}'}(x)$.

PROOF: Let $\bar{\mathcal{M}} = \mathcal{M} \times \mathcal{M}'$, the direct product of $\mathcal{M}$ and $\mathcal{M}'$, be a finite-state machine defined as follows:

$$\bar{\mathcal{M}} = (A_I, Q \times Q', A_o \times A_o, (q_0, q_0'), \bar{\lambda}, \bar{\delta})$$

where

$$\bar{\lambda}((q,q'),a) = (\lambda(q,a), \lambda'(q',a))$$

and

$$\bar{\delta}(q,q') = (\delta(q), \delta'(q')).$$

The desired algorithm consists of determining the accessible states of $\bar{\mathcal{M}}$, and checking whether there is an accessible state (q,q') such that $\delta(q) = \delta'(q')$.

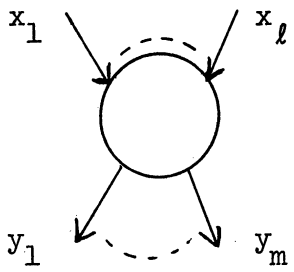
Q.E.D.

Since the inaccessible states of a machine play no part in the determination of the functions rp(x) and res(x), they can be ignored for most purposes. In what follows, we shall sometimes restrict our attention to machines in which all states are accessible.

4. SEQUENTIAL CIRCUITS

One of the principal reasons for studying sequential machines is their usefulness in the analysis and synthesis of synchronous sequential circuits. In this section we present a moderately realistic model of a certain class of sequential circuits, and establish a connection with sequential machines.

A sequential circuit is a finite collection of interconnected elements of two types: gates and latches. A gate with ℓ inputs and m outputs is depicted below. Each input and output is a function of a real variable t (time)

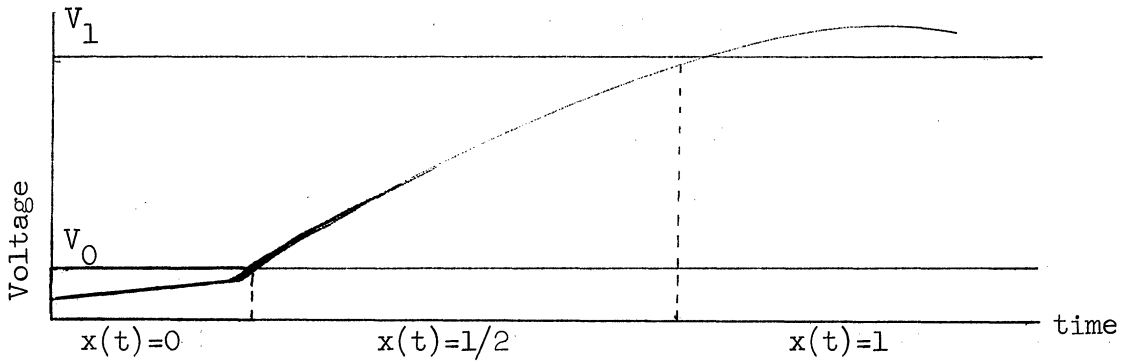


and has 3 possible values: 0, 1/2, 1. Whenever an input is 1/2, we assume that the outputs are undetermined. If we were modelling a physical circuit with voltages as inputs, the interpretation might be as follows:

1: a voltage above a given threshold V_1

0: a voltage below a given threshold V_0

1/2: a voltage between V_0 and V_1, for which the response of the circuit is uncertain.



A <u>switching function</u> of ℓ variables has as its domain $\{0,1\}^{(\ell)}$ (ℓ-tuples of 0's and 1's), and, as its range, $\{0,1\}$. An <u>ℓ-input m-output gate</u> is specified by:

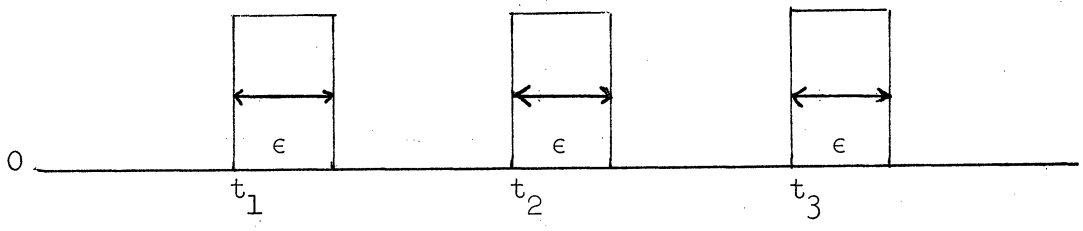
a) m switching functions of ℓ variables,

$f_i(x_1,x_2,\ldots,x_\ell)$ $\quad i=1,\ldots,m.$

b) a response time Δ.

The behavior of a gate is determined as follows: let $(a_1,a_2,\ldots,a_\ell) \in \{0,1\}^{(\ell)}$. If, for $t \leq \tau \leq t+\Delta$, and for $j=1,2,\ldots,\ell$, $x_j(\tau) = a_j$, then, for all i, $y_i(t+\Delta) = f_i(a_1,a_2,\ldots,a_\ell)$. In all other cases, the outputs are undetermined.

There is a special input signal to the circuit, called θ, which is a <u>clock signal of width ϵ</u>. The graph of the function $\theta(t)$ is of the form:

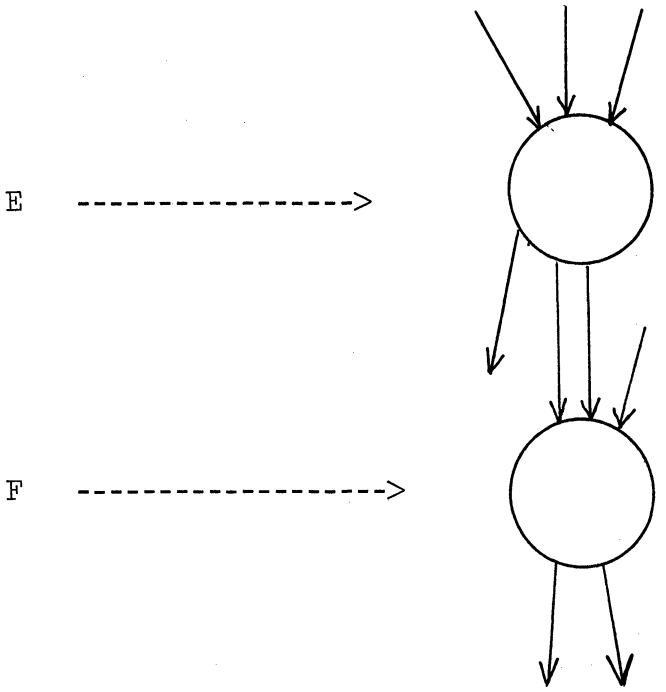


In other words,

$$\theta(\tau) = \begin{cases} 1 \text{ if, for some } j,\ t_j \leq \tau \leq t_j + \epsilon \\ 0 \text{ otherwise.} \end{cases}$$

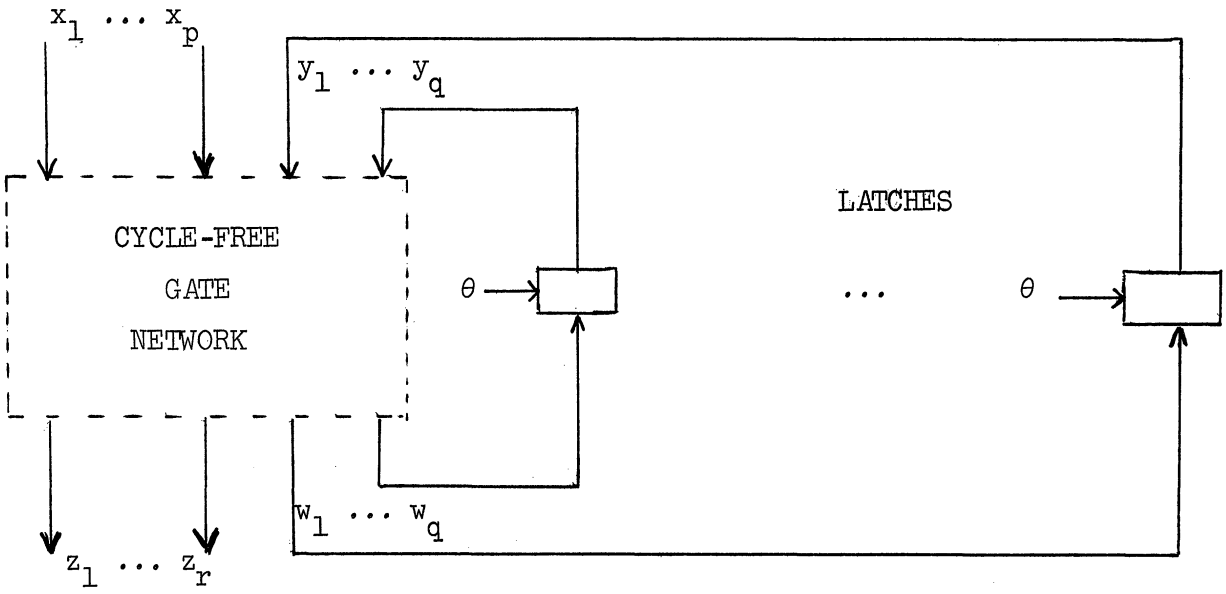
A latch with response time μ has θ as an input, one additional input x, and an output y. If, for $t_j \leq \tau \leq t_j + \epsilon$, $x(\tau) = a$, where $a \in \{0,1\}$, then, for $t_j + \epsilon + \mu \leq \tau < t_{j+1} + \epsilon$, $y(\tau) = a$. In all other cases, y is undetermined.

The element E is connected to F (denoted $E \rightarrow F$) if some of the outputs of E are identified one-to-one with inputs of F. A cycle is a



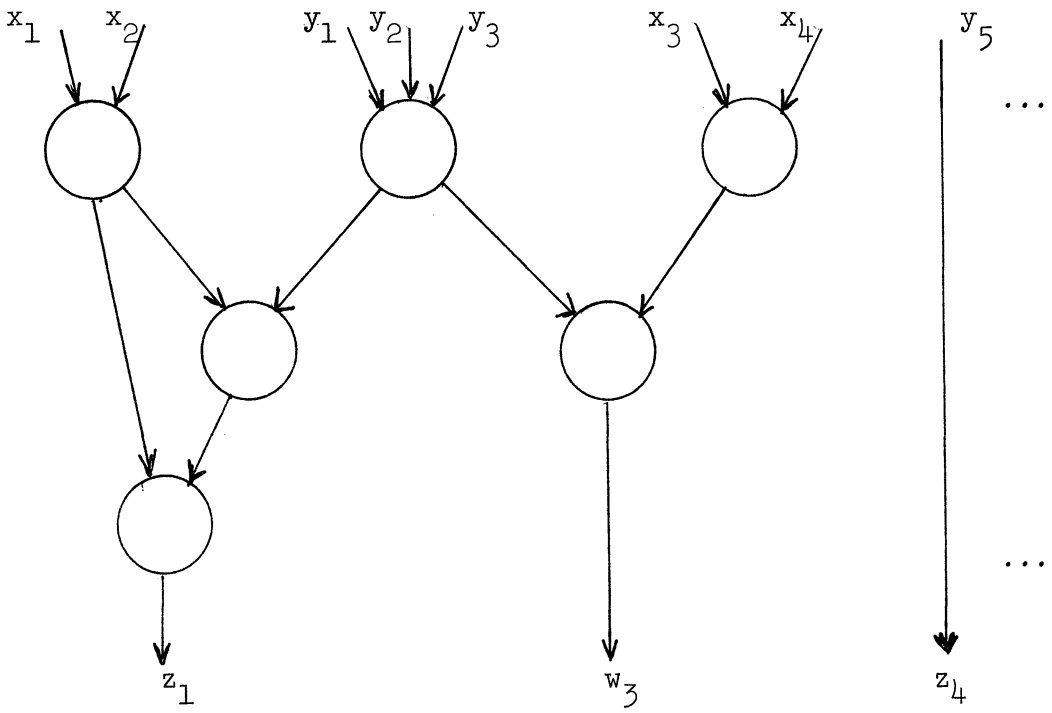
sequence of elements $E_1, E_2, \ldots, E_n$ such that $E_1 \rightarrow E_2 \ldots E_{n-1} \rightarrow E_n \rightarrow E_1$.

We make the following interconnection assumption: every cycle includes a latch. With this assumption, a sequential circuit takes the following form:



A cycle-free gate network is sometimes called a <u>combinational circuit</u>. With regard to its behavior, the entire cycle-free gate network may be represented as a (p+q)-input, (r+q)-output gate, with a response time $\bar{\Delta}$ which is equal to the maximum, over all directed paths in the gate network, of the sum of response times of the elements in the path:

<u>Example 1</u>:



If each element has the same response time Δ, then all the outputs will be determined if the inputs are held at constant binary values for a period 3Δ.

Let the switching functions associated with the "equivalent" gate be divided into an r-tuple $F = (f_1, f_2, \ldots, f_r)$, determining $z_1, \ldots, z_r$, and a q-tuple $G = (q_1, q_2, \ldots, q_q)$, determining $w_1, \ldots, w_q$.

We shall also make the following <u>timing assumptions</u>:

i) there is a constant $\eta > \bar{\Delta}$ such that, for all j, $t_j - t_{j-1} > \epsilon + \bar{\mu} + \eta$, where $\bar{\mu}$ is the maximum response time of any latch in the circuit.

ii) a fixed binary input p-tuple $X(j)$ is applied to the gate network throughout the interval $[t_{j+1} - \eta, t_{j+1} + \epsilon]$.

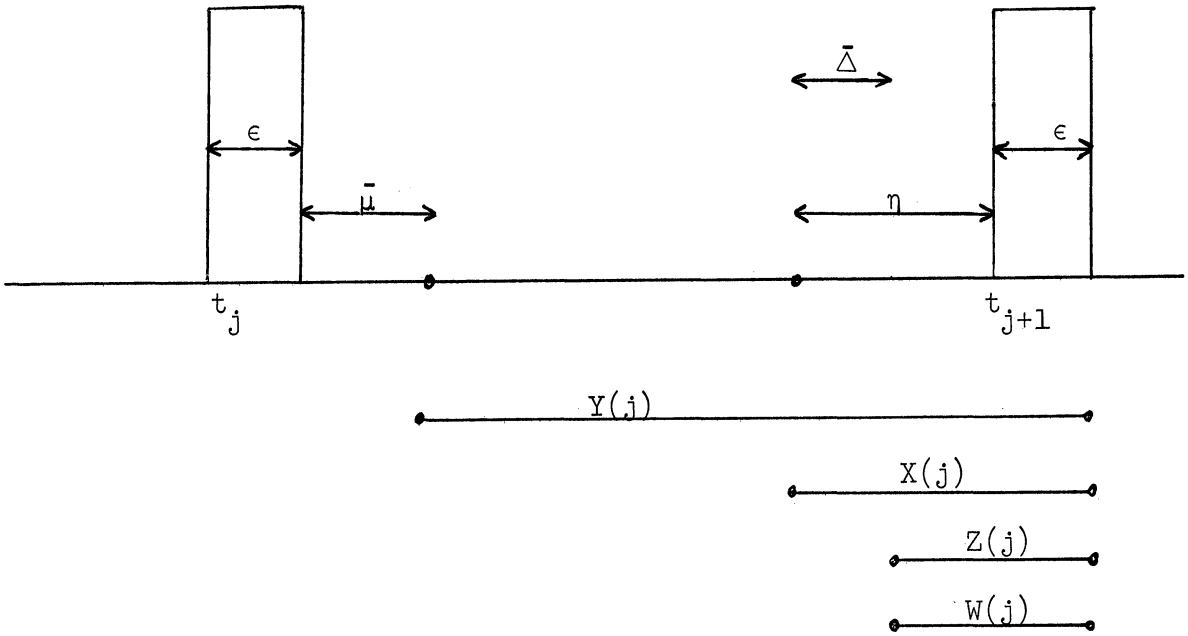
iii) in the period $[t_1 - \eta, t_1 + \epsilon]$, the outputs of the latches are given by a fixed binary q-tuple $Y(0)$.

It follows by induction that, for $j \geq 1$, the outputs of the latches assume fixed binary values $Y(j)$ in the interval $[t_j + \bar{\mu}, t_{j+1} + \epsilon]$, and that the outputs $z_1, \ldots, z_r$ assume fixed binary values given by $Z(j)$ in the interval $[t_{j+1} - \eta + \bar{\Delta}, t_{j+1} + \epsilon]$, where

$$Z(j) = F(X(j), Y(j))$$

$$Y(j+1) = W(j) = G(X(j), Y(j)).$$

Perhaps a picture will clarify the timing.



Thus, because of the interconnection and timing restrictions we have imposed, the essentials of the behavior of the circuit are reduced to difference equations, giving a digital, finitary, discrete-time, deterministic representation. In fact, the circuit can be represented by a Mealy-model sequential machine

$$\mathcal{M} = (\{0,1\}^{(p)}, \{0,1\}^{(q)}, \{0,1\}^{(r)}, Y(0), G, F).$$

The transition and output functions are determined by the combinational circuit, and the internal state is "stored" in the outputs of the latches.

If the functions $F = (f_1,\ldots,f_r)$ are independent of $x_1,\ldots,x_p$, a Moore model representation is obtained.

It is possible to show that, in a sense to be defined later, every sequential machine can be realized by a sequential switching circuit. Later, when we discuss state minimization and decomposition of sequential machines, we shall comment on the implications for sequential circuit design.

5. RIGHT CONGRUENCES AND THE EQUIRESPONSE RELATION

In this section we derive necessary and sufficient conditions for a function

$$f: A_I^* \to Q$$

to be the function rp(x) for some sequential machine with input alphabet A_I and the state set Q. In the course of our development we introduce the concept of a transition system (sequential machine without outputs), define the equiresponse relation of a transition system, and discuss the lattice of equiresponse relations (right congruences) over A_I^*.

First, we review some important concepts. A relation R over a set S is a subset of $S \times S$. The statement $(a,b) \in R$ is sometimes written aRb. A relation R over S is:

reflexive if, for all $a \in S$, aRa

symmetric if aRb $\Longrightarrow$ bRa

antisymmetric if aRb and bRa $\Longrightarrow$ a=b

transitive if aRb and bRc $\Longrightarrow$ aRc.

A relation which is reflexive, symmetric, and transitive is an equivalence relation. Associated with an equivalence relation $\equiv$ over S is a unique partition of S into equivalence classes, such that $a \equiv b$ if and only if a and b are in the same equivalence class. An example of an equivalence relation over the integers is congruence modulo n; the equivalence classes are the residue classes modulo n. An equivalence relation over the set of American citizens is the relation "A lives in the same state as B;" in this case, there are fifty equivalence classes (if we exclude citizens living abroad, etc.). The rank of an equivalence relation is the number of equivalence classes. An equivalence relation is of finite

rank if the number of equivalence classes is finite.

A relation which is reflexive, antisymmetric, and transitive is a partial ordering relation. An example of a partial ordering relation over the subsets of a set is $S_1 \subseteq S_2$.

Let $(S,\cdot)$ be a semigroup. An equivalence relation $\equiv$ over S is called a right regular equivalence relation or a right congruence if

$$x \equiv y \implies \text{for all } w \in S,\ xw \equiv yw;$$

left congruence if

$$x \equiv y \implies \text{for all } u \in S,\ ux \equiv uy;$$

two-sided congruence if

$$x \equiv y \implies \text{for all } u,w \in S,\ uxw \equiv uyw.$$

Equivalently, $\equiv$ is a two-sided congruence if and only if it is both a right congruence and a left congruence.

Two-sided congruences over semigroups are closely related to the homomorphisms of semigroups. In general, a homomorphism is a mapping from one algebraic system onto (or into) another, such that certain relationships among elements are preserved; i.e., if the relationship holds for the original elements, it holds for their images.

Let $(S,\cdot)$ be a semigroup, and let $(T,*)$ be a multiplicative system (T is closed under the binary operation $*$.). A homomorphism of S $\{\substack{\text{into}\\\text{onto}}\}$ T is a function φ from S $\{\substack{\text{into}\\\text{onto}}\}$ T such that:

$$\text{for } x \in S,\ y \in S,\ \varphi(x\cdot y) = \varphi(x) * \varphi(y).$$

Here, the relationship preserved among x,y, and z is this:

$$\text{if } x\cdot y = z, \text{ then } \varphi(x) * \varphi(y) = \varphi(z).$$

Example:

The function $\varphi(x) = lg(x)$ is a homomorphism from $(A_I^*$, concatenation) onto (nonnegative integers, +).

A 1-1, onto homomorphism is an isomorphism; the inverse of an isomorphism is an isomorphism. If a homomorphism exists from a semigroup $(S,\cdot)$ onto a multiplicative system $(T,*)$, then $(T,*)$ is also a semigroup (i.e., * is associative):

$$(\varphi(x) * \varphi(y)) * \varphi(z) = \varphi(x\cdot y) * \varphi(z) = \varphi(x\cdot y\cdot z);$$

$$\varphi(x) * (\varphi(y) * \varphi(z)) = \varphi(x) * (\varphi(y\cdot z)) = \varphi(x\cdot y\cdot z).$$

Any two-sided congruence ρ over S determines a homomorphism of S. For any $x \in S$, let [x] denote the equivalence class of x with respect to ρ. We define S/ρ, the factor (or quotient) semigroup of S modulo ρ as:

$$(\{[x],\ x \in S\},\ \circ), \text{ where } [x] \circ [y] = [x\cdot y].$$

To show that the operation $\circ$ is well defined, it is necessary to show that its result does not depend on which representations of the equivalence classes [x] and [y] are considered. Thus,

suppose $x_1 \in [x]$ and $x_2 \in [x]$; therefore $x_1 \rho x_2$.

Also,

$y_1 \in [y]$ and $y_2 \in [y]$; therefore $y_1 \rho y_2$.

Then $x_1y_1 \rho x_1y_2$, since ρ is a left congruence, and $x_1y_2 \rho x_2y_2$, since ρ is a right congruence. Therefore, by transitivity, $x_1y_1 \rho x_2y_2$. It is now evident that $\circ$ is well defined, and that the function

$$x \to [x]$$

is a homomorphism of S onto S/ρ.

Conversely, let φ be a homomorphism of $(S,\cdot)$ onto $(T,*)$, and let the relation $\underset{\varphi}{\equiv}$ be defined as follows:

$$x \underset{\varphi}{\equiv} y \iff \varphi(x) = \varphi(y).$$

Lemma 1

The relation $\underset{\varphi}{\equiv}$ is a two-sided congruence, and T is isomorphic with $S/\underset{\varphi}{\equiv}$.

PROOF: First we show

$$x \underset{\varphi}{\equiv} y \implies \text{for all } u,w,\ uxw \underset{\varphi}{\equiv} uyw:$$

$$x \underset{\varphi}{\equiv} y \implies \varphi(x) = \varphi(y);$$

$$\varphi(uxw) = \varphi(u)* \varphi(x)* \varphi(w) = \varphi(u)* \varphi(y)* \varphi(w) = \varphi(uyw);$$

therefore

$$uxw \underset{\varphi}{\equiv} uyw.$$

The isomorphism from $S/\underset{\varphi}{\equiv}$ onto T is $f\colon [x] \to \varphi(x)$, where $[x]$ is the equivalence class of x with respect to $\underset{\varphi}{\equiv}$. We must show that

$$f([x] \circ [y]) = f([x])* f([y]).$$

But $$f([x] \circ [y]) = f([x \cdot y]) = \varphi(x \cdot y),$$

and $$f([x])* f([y]) = \varphi(x)* \varphi(y) = \varphi(x \cdot y).$$

Thus we have established the following result.

Theorem 1

Every homomorphic image of a semigroup $(S,\cdot)$ is isomorphic to the factor semigroup with respect to some two-sided congruence, and every factor semigroup $(S/\rho,\circ)$ is a homomorphic image of $(S,\cdot)$

With this introduction to right, left, and two-sided congruences, we may return to the question of characterizing response functions (i.e., determining the possible ways in which the response function may classify the elements of A_I^*). The answer to this question will involve the right congruences over A_I.

Since the function rp(x) does not involve outputs, it will be convenient to consider transition systems (sequential machines without outputs). A transition system is a quadruple

$$\mathbf{T} = (A_I, Q, q_0, \lambda),$$

each element of which is defined exactly as in the definition of a sequential machine. Since rp(x) ranges only over accessible states, we shall further restrict attention to transition systems with every state accessible.

The equiresponse relation $\perp$ of a transition system $\mathbf{T}$ is an equivalence relation over A_I^* defined as follows:

$$x \perp y \iff rp(x) = rp(y).$$

Thus, $\perp$ is of finite rank if and only if the number of (accessible) states of $\mathbf{T}$ is finite.

Lemma 2

For any transition system $\mathbf{T}$, the equiresponse relation is a right congruence over A_I^*.

PROOF: $x \perp y \Longrightarrow rp(x) = rp(y)$

$\Longrightarrow$ for all $w \in A_I^*$, $rp(xw) = rp(yw)$

$\Longrightarrow$ for all $w \in A_I^*$, $x\,w \perp yw$.

Lemma 3

Let ρ be a right congruence over A_I^*. Then ρ is the equiresponse relation of a transition system.

PROOF: For $x \in A_I^*$, let $[x]$ be the equivalence class associated with ρ containing x. Consider the transition system

$$\mathbf{T}(\rho) = (A_I, \{[x], x \in A_I^*\}, [e], \lambda),$$

where $\lambda([x], a) = [xa]$. The function λ is well defined by this equation since

$$x_1 \rho x_2 \Longrightarrow x_1 a \rho x_2 a;$$

i.e. $$[x_1] = [x_2] \Longrightarrow [x_1 a] = [x_2 a].$$

It is easy to prove that, for all x,

$$rp(x) = [x].$$

Therefore,

$$x \perp y \Longleftrightarrow rp(x) = rp(y)$$

$$\Longleftrightarrow [x] = [y]$$

$$\Longleftrightarrow x \rho y.$$

Theorem 2

Every equiresponse relation is a right congruence over A_I^*, and every right congruence over A_I^* is an equiresponse relation.

Thus, we have a complete characterization of equiresponse relations, and, therefore, of the possible functions rp(x). It is interesting to compare right congruences (equiresponse relations) with regard to how finely they classify input sequences. Let R_1 and R_2 be right congruences over A_I^*. Then

$$R_1 \leq R_2 \iff x\ R_1\ y \implies x\ R_2\ y.$$

Thus, when $R_1 \leq R_2$, each equivalence class for R_1 is contained in an equivalence class for R_2.

It is easy to verify that $\leq$ is reflexive, antisymmetric, and transitive. Thus, it is a partial ordering of $\mathcal{R}$, the set of all right congruences over A_I^*; equivalently, we say that $(\mathcal{R}, \leq)$ is a partially ordered set. We shall establish the stronger result that $(\mathcal{R}, \leq)$ is a lattice.

Let us review the definition of a lattice. Let $(S, \leq)$ be a partially ordered set, and let x and y be elements of S. The element $z \in S$ is the greatest lower bound of x and y (denoted glb(x,y)) if

a) $z \leq x$ and $z \leq y$, and

b) if $w \leq x$ and $w \leq y$, then $w \leq z$.

Similarly, v is the least upper bound of x and y (denoted lub(x,y)) if

a) $x \leq v$ and $y \leq v$, and

b) if $x \leq u$ and $y \leq u$, then $v \leq u$.

A lattice is a partially ordered set in which any two elements have a glb and a lub.

Example 1:

The partially ordered set $(S, \leq)$ where S is the positive integers, and $x \leq y$ if x is a divisor of y, is a lattice. In this lattice,

$$\text{lub}(x,y) = \text{lcm}(x,y), \text{ and } \text{glb}(x,y) = \text{gcd}(x,y).$$

Example 2:

The partially ordered set $(S, \leq)$, where

$$S = \{a,b,c,d\},$$

and the relation $\leq$ is

$$\{(a,c), (a,d), (b,c), (b,d)\},$$

is not a lattice.

Theorem 3

The partially ordered set $(\mathcal{R}, \leq)$ is a lattice.

PROOF: If $R_1 \in \mathcal{R}$ and $R_2 \in \mathcal{R}$, we shall explicitly define

$$P = \text{glb}(R_1,R_2)$$

and

$$Q = \text{lub}(R_1,R_2),$$

and prove that P and Q have the required properties. We define P as follows:

$$x \; P \; y \iff x \; R_1 \; y \text{ and } x \; R_2 \; y$$

It is evident that P is an equivalence relation. Also, P is a right congruence:

$$x\ P\ y \implies x\ R_1\ y \text{ and } x\ R_2\ y$$

$$\implies \forall\ w,\ xw\ R_1\ yw \text{ and } xw\ R_2\ yw$$

$$\implies \forall\ w,\ xw\ P\ yw.$$

Also, it is clear that $x\ P\ y \implies x\ R_1\ y$ and

$x\ P\ y \implies x\ R_2\ y$, so that

$P \leq R_1$ and $P \leq R_2$. Now, suppose $R \leq R_1$ and $R \leq R_2$. Then

$$x\ R\ y \implies x\ R_1\ y$$

and $$x\ R\ y \implies x\ R_2\ y;$$

thus $$x\ R\ y \implies x\ P\ y.$$

This completes the proof that $P = \text{glb}(R_1,R_2)$.

It is tempting to define

$$Q = \text{lub}(R_1,R_2)$$

by $x\ Q\ y \iff x\ R_1\ y$ or $x\ R_2\ y \iff x(R_1 \cup R_2)y$.

Unfortunately, this is not a transitive relation. For example, take

$$x\ R_1\ y \text{ if } \lg(x) \equiv \lg(y) \bmod 2$$

and $$x\ R_2\ y \text{ if } \lg(x) \equiv \lg(y) \bmod 3\ .$$

Then, if $a \in A_I$, $\quad e\ R_1\ a^2$, $\quad$ and $\quad a^2\ R_2\ a^5$,

but it is neither true that $\quad e\ R_1\ a^5 \quad$ nor that $e\ R_2\ a^5$.

(Here, a^n denotes the concatenation of n a's.)

Instead, we must define Q as the <u>transitive closure</u> of the relation

$R_1 \cup R_2$: $\quad x\ Q\ y \iff \exists\ z_1,z_2,\ldots,z_r$ such that

$x=z_1$, $\quad y=z_r$, $\quad$ and $\quad z_1(R_1 \cup R_2)z_{i+1}$ for $i=1,2,\ldots,r-1$.

In this case, we say that x is <u>chain connected</u> to y by the sequence

$z_1,z_2,\ldots,z_r$. Then Q is an equivalence relation; to check transitivity,

for example, note that, if x is chain connected to y by the sequence $z_1,z_2,\ldots,z_r$, and y to z by the sequence $w_1,w_2,\ldots,w_s$, then x is chain connected to z by the sequence $z_1,z_2,\ldots,z_r, w_2,\ldots,w_s$. Also, Q is a right congruence; for, if x is chain connected to y by $z_1,z_2,\ldots,z_r$, then xw is chain connected to yw by $z_1w, z_2w, \ldots, z_rw$.

It is further clear that Q is an upper bound for R_1 and R_2:

$$x\ R_1\ y \implies x\ Q\ y$$

$$x\ R_2\ y \implies x\ Q\ y.$$

Finally, suppose R is an upper bound for both R_1 and R_2;

$$x\ R_1\ y \implies x\ R\ y$$

and

$$x\ R_2\ y \implies x\ R\ y.$$

Suppose x Q y. Then x and y are chain connected by $z_1,\ldots,z_r$, where, for each i,

$$z_i\ R_1\ z_{i+1}$$

or

$$z_i\ R_2\ z_{i+1};$$

thus

$$z_i\ R\ z_{i+1}.$$

Since R is transitive,

$$z_1\ R\ z_r; \qquad \text{i.e.,}\quad x\ R\ y.$$

Therefore, x Q y $\implies$ x R y, and $Q \leq R$. This completes the proof of Theorem 3.

It may be helpful to discuss the equivalence classes of the right congruences

$$P = \text{glb}(R_1,R_2) \text{ and } Q = \text{lub}(R_1,R_2).$$

The equivalence classes of P are simply those nonempty sets obtained as the intersection of an equivalence class of R_1 with an equivalence class of R_2. The determination of the equivalence classes of Q can be illustrated

by an example. Suppose R_1 and R_2 are both of finite rank. Let R_1 have equivalence classes E_1, E_2, E_3, E_4, and R_2, equivalence classes F_1, F_2, F_3, F_4, F_5. These classes are indicated below, with lines drawn between the classes which are assumed to have an element in common.

Then $E_1 \cup E_2 = F_1 \cup F_2 \cup F_3$,

and $E_3 \cup E_4 = F_4 \cup F_5$.

The equivalence classes of Q are $E_1 \cup E_2$ and $E_3 \cup E_4$.

The lattice $(\mathcal{K}, \leq)$ has the universal lower bound 0 and the universal upper bound I, where:

$$\forall x \quad \forall y, \; xOy,$$

and

$$xIy \Longleftrightarrow x=y.$$

The sets of left congruences and two-sided congruences over A_I^* are lattices, with the same ordering relation and definitions of glb and lub as in $(\mathcal{K}, \leq)$. Each of these lattices is a sublattice of the lattice of equivalence relations over A_I^*.

6. THE LATTICE OF STRUCTURE TYPES OF TRANSITION SYSTEMS

It is natural to introduce an ordering $\rightarrow$ of transition systems with input alphabet A_I induced by the ordering of their equiresponse relations:

$$\mathcal{T} \rightarrow \mathcal{T}' \iff \perp_{\mathcal{T}} \leq \perp_{\mathcal{T}'}.$$

This is a quasi-ordering (reflexive and transitive), but not a partial ordering (not antisymmetric), since distinct transition systems may have the same equiresponse relation. If we say that two transition systems are of the same structure type if their equiresponse relations are the same, then the structure types form a lattice isomorphic with $(\mathcal{K}, \leq)$.

In this section we develop the lattice ordering of structure types of transition systems somewhat more indirectly than in the above discussion, and thereby introduce the important concepts 'homomorphism of a transition system,' and 'equivalence relation with the substitution property.'

Let $\mathcal{T} = (A_I, Q, q_0, \lambda)$

and $\bar{\mathcal{T}} = (A_I, \bar{Q}, \bar{q}_0, \bar{\lambda})$

be transition systems for which all states are accessible. A homomorphism from $\mathcal{T}$ onto $\bar{\mathcal{T}}$ is a function $h: Q \rightarrow \bar{Q}$ such that

a) $h(q_0) = \bar{q}_0$

b) $\forall\, q \in Q, \forall\, a \in A_I,\ h(\lambda(q,a)) = \bar{\lambda}(h(q),a)$.

Thus, if $\lambda(q,a) = q'$,

$$\bar{\lambda}(h(q),a) = h(q').$$

A one-to-one homomorphism is an isomorphism; the inverse of an isomorphism is an isomorphism.

There is an immediate application of the concept of a homomorphism to the "series connection" of transition systems. In what follows we use the terms "series connection" and "realization" heuristically without defining them. Let h: $Q \to \bar{Q}$ be a homomorphism of $\mathcal{T}$ onto $\bar{\mathcal{T}}$, let R be a set, and let f be a one-to-one function from Q onto S, where $S \subseteq \bar{Q} \times R$, such that, for all q,

$$f(q) = (h(q), k(q)).$$

Since f is one-to-one and onto, it has an inverse f^{-1}.

Each element of Q thus has a "representation" in S as an ordered pair (h(q), k(q)), and, corresponding to the function

$$\lambda: Q \times A_I \to Q,$$

there is a function μ which operates on the representations:

$$\mu: S \times A_I \to S.$$

It is required that

$$\mu(f(q),a) = f(\lambda(q,a)).$$

Thus, if $(\bar{q},r) \in S$,

$$\mu((\bar{q},r),a) = F\,\lambda(f^{-1}(\bar{q},r),a) = (h\,\lambda(f^{-1}(\bar{q},r),a),\ k\,\lambda(f^{-1}(\bar{q},r),a).$$

Since h is a homomorphism,

$$h\,\lambda(f^{-1}(\bar{q},r),a) = \bar{\lambda}(h\,f^{-1}(\bar{q},r),a).$$

But, if

$$f^{-1}(\bar{q},r) = q^*,$$

then

$$h(q^*) = \bar{q};$$

thus,

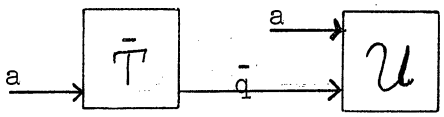
$$h\,f^{-1}(\bar{q},r) = \bar{q},$$

and the first component of $\mu(\bar{q},r),a)$ is $\bar{\lambda}(\bar{q},a)$, which is independent of r, and is given by the transition function of $\bar{\mathcal{T}}$. The second component, $k\,\lambda(f^{-1}(\bar{q},r),a)$ will, in general, be dependent on $\bar{q}$, r, and a. Let us

write this function in the form $(\varphi(\bar{q},a),r)$; if $(\bar{q},r) \notin S$, φ may be assigned an arbitrary value in R. Then φ can be taken as the transition function of a transition system

$$\mathcal{U} = (\bar{Q} \times A_I, R, k(q_0), \varphi).$$

Strictly speaking, $\mathcal{U}$ is a transition system only if $\bar{Q}$ if finite; otherwise, $\mathcal{U}$ has an infinite number of input symbols. The function μ is the transition function of the transition system which is the series connection of $\bar{\mathcal{T}}$ and $\mathcal{U}$:



Thus, the homomorphism h has yielded a series realization of $\mathcal{T}$. $\mathcal{T}$ has been decomposed into "simpler" transition systems; note that, if $\mathcal{T}$ is finite-state, and h is neither an isomorphism nor the constant function

$$h(q) = \bar{q}_0,$$

then R can be chosen such that $\bar{Q}$ and R each have fewer elements than Q.

Just as semigroup homomorphisms are determined by two-sided congruences, the homomorphisms of a transition system $\mathcal{T}$ are determined by the equivalence relations with the substitution property over Q. An equivalence relation ρ over Q has the <u>substitution property</u> if

$$q_1 \rho q_2 \implies \text{for all } a \in A_I,\ \lambda(q_1,a) \rho\ \lambda(q_2,a).$$

Associated with ρ, an equivalence relation with S.P., is a homomorphism of $\mathcal{T}$ onto the <u>factor transition system modulo ρ</u>, $\mathcal{T}/\rho$. For $q \in Q$, let $[q]$ denote the equivalence class of q with respect to ρ. Then

$$\mathcal{T}/\rho \;=\; (A_I,\ \{[q],\ q \in Q\},\ [q_0],\ \bar{\lambda}),$$

where

$$\bar{\lambda}([q],a) \;=\; [\lambda(q,a)].$$

We omit the easy verification that $\bar{\lambda}$ is well defined, and that the mapping $q \to [q]$ is a homomorphism of $\mathcal{T}$ onto $\mathcal{T}/\rho$.

Going in the reverse direction, let h be a homomorphism of $\mathcal{T}$ onto $\bar{\mathcal{T}}$. Then the equivalence relation $\equiv$ defined by:

$$q_1 \equiv q_2 \iff h(q_1) = h(q_2)$$

has S.P., and $\bar{\mathcal{T}}$ is isomorphic with $\mathcal{T}/\equiv$. The verification of this is analogous to the corresponding step in our discussion of semigroup homomorphisms.

The following theorem summarizes the discussion so far.

Theorem 1

Every homomorphic image of a transition sytem $\mathcal{T}$ is isomorphic with $\mathcal{T}/\rho$, for some equivalence relation with S.P. ρ, and every factor transition system $\mathcal{T}/\rho$ is a homomorphic image of $\mathcal{T}$.

Let $\mathcal{T}$ and $\bar{\mathcal{T}}$ be transition systems with input alphabet A_I and with all states accessible. Let their equiresponse relations be, respectively, $\perp_{\mathcal{T}}$ and $\perp_{\bar{\mathcal{T}}}$. We shall show that there is a homomorphism from $\mathcal{T}$ onto $\bar{\mathcal{T}}$ if and only if $\perp_{\mathcal{T}} \leq \perp_{\bar{\mathcal{T}}}$, and that such a homomorphism, if it exists, is unique.

Lemma 1

Let h be a homomorphism from $\mathcal{T}$ onto $\bar{\mathcal{T}}$. Then, for all $x \in A_I^*$,

$rp_{\bar{\mathcal{T}}}(x) = h(rp_{\mathcal{T}}(x))$.

PROOF: The proof is by induction on lg(x).

Basis. If $lg(x) = 0$,

then $x = e$,

$rp_{\bar{\mathcal{T}}}(e) = \bar{q}_0$,

and $h(rp_{\mathcal{T}}(e)) = h(q_0) = \bar{q}_0$.

Induction step. Assuming that the statement of the lemma holds when

$lg(x) = r-1$,

consider $y = xa$,

where $a \in A_I$ and $lg(y) = r$.

Then $rp_{\bar{\mathcal{T}}}(xa) = \bar{\lambda}(rp_{\bar{\mathcal{T}}}(x), a)$

$= \bar{\lambda}(h(rp_{\mathcal{T}}(x)), a)$

$= h(\lambda(rp_{\mathcal{T}}(x), a))$

$= h(rp_{\mathcal{T}}(xa))$.

Let the relations $\rightarrow$ and $\leftrightarrow$ be defined by

$\mathcal{T} \rightarrow \bar{\mathcal{T}} \iff$ there is a homomorphism from $\mathcal{T}$ onto $\bar{\mathcal{T}}$,

and $\mathcal{T} \leftrightarrow \bar{\mathcal{T}}$ if and only if there is an isomorphism from $\mathcal{T}$ onto $\bar{\mathcal{T}}$.

The following corollaries of Lemma 1 are immediate.

Corollary 1

$x \perp_{\mathcal{T}/\rho} y \iff rp_{\mathcal{T}}(x)\ \rho\ rp_{\mathcal{T}}(y)$.

Corollary 2

If $T \rightarrow \bar{T}$, then $\perp_T \leq \perp_{\bar{T}}$.

Corollary 3

If $T \leftrightarrow \bar{T}$, then $\perp_T = \perp_{\bar{T}}$.

Corollary 4

If $T \rightarrow \bar{T}$, there is exactly one homomorphism from T to $\bar{T}$.

PROOF: Since all states are accessible, h is completely determined by the equation

$$rp_T(x) = h(rp_{\bar{T}}(x)).$$

We next establish the converse of Lemma 1.

Lemma 2

If $\perp_T \leq \perp_{\bar{T}}$, then $T \rightarrow \bar{T}$.

PROOF: Let the function h from Q onto $\bar{Q}$ be defined as follows:

$$h(q) = \bar{q} \iff rp_T(x) = q \implies rp_{\bar{T}}(x) = \bar{q}.$$

Since $rp_T(x) = rp_T(y) \implies rp_{\bar{T}}(x) = rp_{\bar{T}}(y)$,

and since every state q is accessible, the given definition determines h completely.

To verify that h is a homomorphism, we note that:

a) $rp_T(e) = q_0$

and $rp_{\bar{T}}(e) = \bar{q}_0$;

therefore, $h(q_0) = \bar{q}_0$.

b) Suppose $rp_{\mathcal{T}}(x) = q$,

so that $rp_{\bar{\mathcal{T}}}(x) = h(q)$.

Then $rp_{\mathcal{T}}(xa) = \lambda(q,a)$,

and $rp_{\bar{\mathcal{T}}}(xa) = \bar{\lambda}(rp_{\bar{\mathcal{T}}}(x),a)$

$= \bar{\lambda}(h(q),a)$.

Therefore $h(\lambda(q,a)) = \bar{\lambda}(h(q),a)$.

Summarizing, we have:

Theorem 2

$$\mathcal{T} \rightarrow \bar{\mathcal{T}} \iff \perp_{\mathcal{T}} \leq \perp_{\bar{\mathcal{T}}}$$

Corollary 5

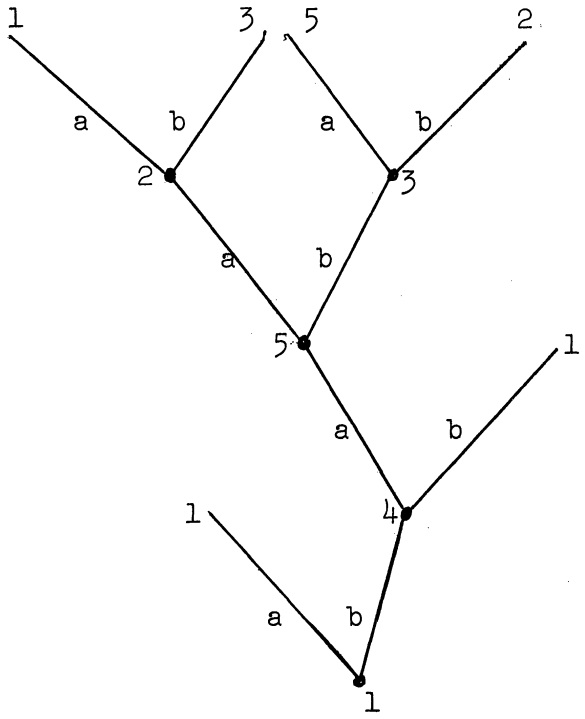
$$\mathcal{T} \leftrightarrow \bar{\mathcal{T}} \iff \perp_{\mathcal{T}} = \perp_{\bar{\mathcal{T}}}$$

Example 1:

Let us apply our results so far by constructing a homomorphic image of a finite-state system $\mathcal{T}$ with $A_I = \{a,b\}$, $Q = \{1,2,3,4,5\}$, $q_0 = 1$ and λ specified by the following table:

$\lambda(q,a)$		A_I	
		a	b
Q	1	1	4
	2	1	3
	3	5	2
	4	5	1
	5	2	3

The pruned response tree for $\mathcal{T}$ is:

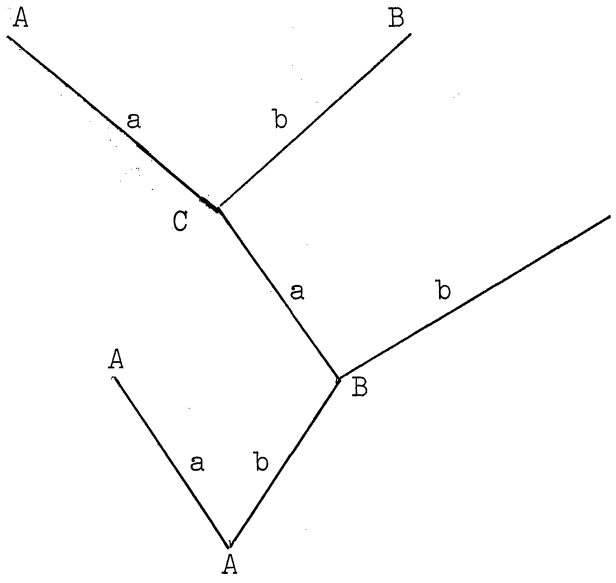


The equivalence relation ρ with equivalence classes $\{1,2\} = A$, $\{3,4\} = B$, and $\{5\} = C$ has the substitution property. The response function for $\mathcal{T}/\rho$ is given by

$$rp_{\mathcal{T}/\rho}(x) = [rp_{\mathcal{T}}(x)].$$

Thus $$\bar{Q} = \{A,B,C\}.$$

The pruned response tree for $\mathcal{T}/\rho$ is:



A series decomposition of $\mathcal{T}$ based on the homomorphism $\mathcal{T} \to \mathcal{T}/\rho$ can easily be constructed. Let R = $\{\alpha,\beta\}$, and let f: $Q \to S \subseteq \bar{Q} \times R$ be as follows:

q	f(q)
1	A,α
2	A,β
3	B,α
4	B,β
5	C,α

Then μ is given by the following table:

	a	b
A,α	A,α	B,β
A,β	A,α	B,α
B,α	C,α	A,β
B,β	C,α	A,α
C,α	A,β	B,α

The transition system $\mathcal{T}$ may be realized by the series connection of $\mathcal{T}/\rho$ and $\mathcal{U}$, where $\mathcal{U} = (Q \times A_I, R, \alpha, \varphi)$, and φ has the following table, where — indicates that the entry is arbitrary.

		A,a	A,b	B,a	B,b	C,a	C,b
R	α	α	β	α	β	β	α
	β	α	α	α	α	-	-

Corollary 5 establishes that the relation $\leftrightarrow$ is an equivalence relation over the set of transition systems with input alphabet A_I for which all states are accessible. Let the equivalence classes associated with this relation be called the structure types of transition systems. Let the structure type of $\mathcal{T}$ be denoted $< \mathcal{T} >$, and let the equiresponse relation common to all the elements of $< \mathcal{T} >$ be denoted $\perp_{< \mathcal{T} >}$. The relation $\rightarrow$ between structure types is defined in the natural way:

$$< \mathcal{T} > \rightarrow < \bar{\mathcal{T}} >$$

if and only if, for all $\mathcal{K} \in < \mathcal{T} >$ and $\mathcal{L} \in < \bar{\mathcal{T}} >$, $\mathcal{K} \rightarrow \mathcal{L}$.

Then the following is an easy corollary of Theorem 2.

Corollary 6

$< \mathcal{T} > \rightarrow < \bar{\mathcal{T}} >$ if and only if $\perp_{< \mathcal{T} >} \leq \perp_{< \bar{\mathcal{T}} >}$.

The relation $\rightarrow$ is a partial ordering relation over the set of structure types. To discuss the relationship between the system $(\{< \mathcal{T} >\}, \rightarrow)$ and $(\mathcal{K}, \leq)$, we need some further general concepts. The partially ordered systems (S, $\leq$) and (T, $\subseteq$) are isomorphic if there is a one-to-one function φ from S onto T such that:

$$x \leq y \iff \varphi(x) \subseteq \varphi(y).$$

If (S, $\leq$) and (T, $\subseteq$) are isomorphic, then one is a lattice if and only if

the other is. If $(S, \leq)$ and $(T, \subseteq)$ are isomorphic lattices, then

$$\varphi(\text{glb}(x,y)) = \text{glb}(\varphi(x), \varphi(y)),$$

and

$$\varphi(\text{lub}(x,y)) = \text{lub}(\varphi(x), \varphi(y)).$$

Theorem 3

The partially ordered systems $(\{<\mathcal{T}>\}, \rightarrow)$ and $(\mathcal{R}, \leq)$ are isomorphic lattices.

PROOF: The structure type $<\mathcal{T}> \in \{<\mathcal{T}>\}$ corresponds to the right congruence $\perp_{<\mathcal{T}>} \in \mathcal{R}$. Since every right congruence $\mathcal{R}$ is the equiresponse relation of some transition system $\mathcal{T}$, every right congruence $\mathcal{R}$ is $\perp_{<\mathcal{T}>}$ for some structure type $<\mathcal{T}>$. It is immediate that the conditions for isomorphism are satisfied.

To complete our discussion of the lattice $(\{<\mathcal{T}>\}, \rightarrow)$, we shall explicitly determine the greatest lower bound and least upper bound of two arbitrary structure types $<\mathcal{T}>$ and $<\bar{\mathcal{T}}>$, and shall specify universal upper and lower bounds. Let us keep in mind that $\text{glb}(<\mathcal{T}>, <\bar{\mathcal{T}}>)$ is the structure type of the "least complex" transition system that has both $\mathcal{T}$ and $\bar{\mathcal{T}}$ as homomorphic images, and that $\text{lub}(<\mathcal{T}>, <\bar{\mathcal{T}}>)$ is the structure type of the "most complex" transition system that is a homomorphic image of both $\mathcal{T}$ and $\bar{\mathcal{T}}$.

Let us recall that $\mathcal{T} \times \bar{\mathcal{T}}$, the direct product of the transition systems $\mathcal{T}$ and $\bar{\mathcal{T}}$, is defined as follows:

$$\mathcal{T} \times \bar{\mathcal{T}} = (A_I, Q \times \bar{Q}, (q_0, \bar{q}_0), \hat{\lambda}),$$

where

$$\hat{\lambda}((q,\bar{q}),a) \;=\; (\lambda(q,a),\ \lambda(\bar{q},a)).$$

Since $\mathcal{T} \times \bar{\mathcal{T}}$ may have inaccessible states, we define ac($\mathcal{T} \times \bar{\mathcal{T}}$) (the accessible part of $\mathcal{T} \times \bar{\mathcal{T}}$) as the transition system

$$(A_I,\ R,\ (q_0,\bar{q}_0),\ \lambda'),$$

where R is the set of accessible states of $\mathcal{T} \times \bar{\mathcal{T}}$, and λ' is $\hat{\lambda}$ restricted to the domain $R \times A_I$.

Lemma 3

For any two structure types $<\mathcal{T}>$ and $<\bar{\mathcal{T}}>$,

$$\text{glb}\ (<\mathcal{T}>\ ,\ <\bar{\mathcal{T}}>)\ =\ <\text{ac}(\mathcal{T} \times \bar{\mathcal{T}})>.$$

PROOF: Since all states of ac($\mathcal{T} \times \bar{\mathcal{T}}$) are accessible, the structure type $<\text{ac}(\mathcal{T} \times \bar{\mathcal{T}})>$ is well defined. Because of the isomorphism between $(\{<\mathcal{T}>\}, \rightarrow)$ and $(\mathcal{K}, \leq)$, it is sufficient to show that

$$\text{glb}(\perp_{<\mathcal{T}>},\ \perp_{<\bar{\mathcal{T}}>})\ =\ \perp_{<\text{ac}(\mathcal{T} \times \bar{\mathcal{T}})>}.$$

But

$$\text{glb}(\perp_{<\mathcal{T}>},\ \perp_{<\bar{\mathcal{T}}>})\ =\ \text{glb}(\perp_{\mathcal{T}},\ \perp_{\bar{\mathcal{T}}}),$$

and

$$\perp_{<\text{ac}(\mathcal{T} \times \bar{\mathcal{T}})>}\ =\ \perp_{\text{ac}(\mathcal{T} \times \bar{\mathcal{T}})}\ =\ \perp_{\mathcal{T} \times \bar{\mathcal{T}}}.$$

But, clearly,

$$\text{rp}_{\mathcal{T} \times \bar{\mathcal{T}}}(x)\ =\ (\text{rp}_{\mathcal{T}}(x),\ \text{rp}_{\bar{\mathcal{T}}}(x)),$$

so that

$$x \perp_{\mathcal{T} \times \bar{\mathcal{T}}} y \iff x \perp_{\mathcal{T}} y \text{ and } x \perp_{\bar{\mathcal{T}}} y.$$

Thus

$$\perp_{\mathcal{T} \times \bar{\mathcal{T}}} = \text{glb}(\perp_{\mathcal{T}}, \perp_{\bar{\mathcal{T}}}).$$

The definition of the least upper bound of two structure types is somewhat more complex than the definition of their greatest lower bound. Let ρ be the following equivalence relation over R, the set of states of $ac(\mathcal{T} \times \bar{\mathcal{T}})$: $(p,\bar{p})\rho(q,\bar{q})$ if and only if there exists a sequence $(q_1,\bar{q}_1),\ldots,(q_r,\bar{q}_r)$ such that:

i) $(q_1,\bar{q}_1) = (p,\bar{p})$

ii) $(q_r,\bar{q}_r) = (q,\bar{q})$

iii) for $i=1,2,\ldots,r$, $(q_i,\bar{q}_i) \in R$, and

iv) for $i=1,2,\ldots,r-1$, $q_i = q_{i+1}$ or $\bar{q}_i = \bar{q}_{i+1}$.

If these conditions are met, then $(p,\bar{p})$ is said to be <u>chain connected</u> to $(q,\bar{q})$ by the sequence $(q_1,q_1),\ldots,(q_r,\bar{q}_r)$. The equivalence relation ρ has S.P.; for, if $(p,\bar{p})$ is chain connected to $(q,\bar{q})$ by $(q_1,\bar{q}_1),\ldots,(q_r,\bar{q}_r)$, then $\lambda'((p,\bar{p}),a)$ is chain connected to $\lambda'((q,\bar{q}),a)$ by the sequence $\lambda'((q_1,\bar{q}_1),a),\ldots\lambda'((q_r,\bar{q}_r),a)$.

<u>Lemma 4</u>

For any two structure types $<\mathcal{T}>$ and $<\bar{\mathcal{T}}>$

$$\text{lub}(<\mathcal{T}>, <\bar{\mathcal{T}}>) = < ac(\mathcal{T} \times \bar{\mathcal{T}})/\rho > .$$

PROOF: Since every state of $ac(\mathcal{T} \prec \bar{\mathcal{T}})$ is accessible, every state of its homomorphic image $ac(\mathcal{T} \prec \bar{\mathcal{T}})/\rho$ is accessible, therefore, the structure type $< ac(\mathcal{T} \prec \bar{\mathcal{T}})/\rho >$ is well defined. We have to show that

$$\perp_{< ac(\mathcal{T} \prec \bar{\mathcal{T}})/\rho >} = lub(\perp_{<\mathcal{T}>}, \perp_{<\bar{\mathcal{T}}>}).$$

But

$$\perp_{< ac(\mathcal{T} \prec \bar{\mathcal{T}})/\rho >} = \perp_{ac(\mathcal{T} \prec \bar{\mathcal{T}})/\rho},$$

and, by Corollary 1,

$$x \perp_{ac(\mathcal{T} \prec \bar{\mathcal{T}})/\rho} y$$

if and only if

$$rp_{ac(\mathcal{T} \prec \bar{\mathcal{T}})}(x) \; \rho \; rp_{ac(\mathcal{T} \prec \bar{\mathcal{T}})}(y),$$

and this is equivalent to $rp_{\mathcal{T} \prec \bar{\mathcal{T}}}(x)$ being chain connected to $rp_{(\mathcal{T} \prec \bar{\mathcal{T}})}(y)$ by a sequence $(q_1,\bar{q}_1),(q_2,\bar{q}_2),\ldots,(q_r,\bar{q}_r)$ meeting the conditions stated above. On the other hand,

$$lub(\perp_{<\mathcal{T}>}, \perp_{<\bar{\mathcal{T}}>}) = lub(\perp_{\mathcal{T}}, \perp_{\bar{\mathcal{T}}}),$$

and

$$x \perp_{(lub(<\mathcal{T}>, <\bar{\mathcal{T}}>)} y$$

if and only if x is chain connected to y by a sequence $z_1,z_2,\ldots,z_r$ meeting the conditions stated in Theorem 5.3. But the two types of chain connectedness are indeed equivalent. For if $rp_{(\mathcal{T} \prec \bar{\mathcal{T}})}$ x is chain connected to $rp_{(\mathcal{T} \prec \bar{\mathcal{T}})}$ y by $(q_1,\bar{q}_1),(q_2,\bar{q}_2),\ldots,(q_r,\bar{q}_r)$, then x is chain connected to y by $z_1,z_2,\ldots,z_r$, where z_1=x, z_r=y, and, for i=2,3,...,r-1, z_i is any element of A_I^* such that

$$rp_{T \times \bar{T}}(z_i) = (q_i, \bar{q}_i);$$

such a z_i is guaranteed to exist, since $(q_i, \bar{q}_i)$ is an accessible state of $T \times \bar{T}$. Going the other way, we observe that, if x is chain connected to y by the sequence $z_1, z_2, \ldots, z_r$, then $rp_{T \times \bar{T}}(x)$ is chain connected to $rp_{T \times \bar{T}}(y)$ by the sequence

$$rp_{T \times \bar{T}}(z_1),\ rp_{T \times \bar{T}}(z_2),\ \ldots,\ rp_{T \times \bar{T}}(z_r).$$

Example 2:

Let T and $\bar{T}$ be transition systems, where

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

$$Q = \{a,b,c\},$$

$$\bar{Q} = \{A,B,C\}.$$

q_0=a, $\bar{q}_0$=A, and λ and $\bar{\lambda}$ are specified as follows:

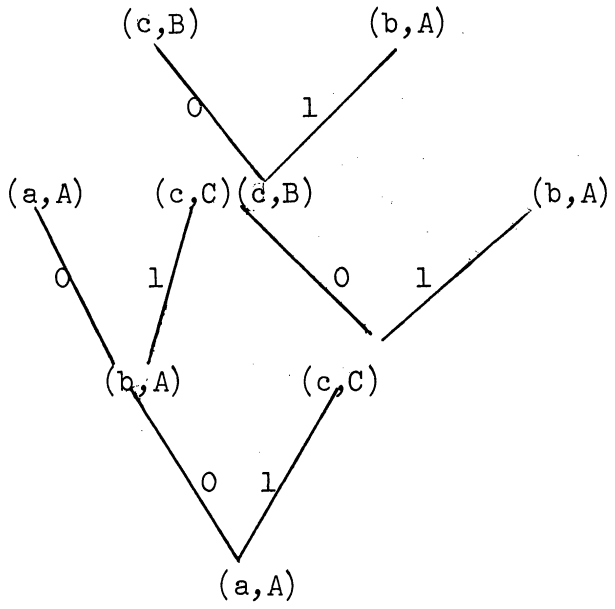
	0	1
a	b	c
b	a	c
c	c	b

λ

	0	1
A	A	C
B	B	A
C	B	A

$\bar{\lambda}$

The response tree for $ac(T \times \bar{T})$ is as follows:



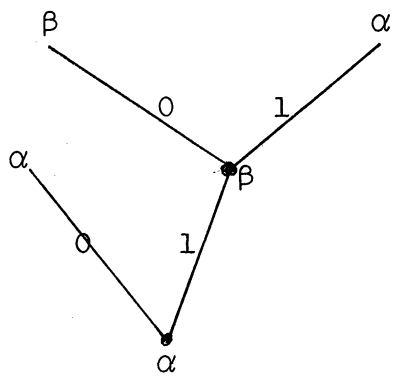
Thus, every transition system in glb($<\mathcal{T}>$, $<\bar{\mathcal{T}}>$) has four states, and is isomorphic to ac($\mathcal{T} \times \bar{\mathcal{T}}$). There is a homomorphism h from ac($\mathcal{T} \times \bar{\mathcal{T}}$) onto $\mathcal{T}$, where h(a,A) = a, h(b,A) = b, and h(c,B) = h(c,C) = c. There is a homomorphism $\bar{h}$ from ac($\mathcal{T} \times \bar{\mathcal{T}}$) onto $\bar{\mathcal{T}}$, where $\bar{h}$(a,A) = $\bar{h}$(b,A) = A, $\bar{h}$(c,B) = B, and $\bar{h}$(c,C) = C. The equivalence relation ρ has the equivalence classes

$$\alpha = \{(a,A), (b,A)\}$$

and

$$\beta = \{(c,B), (c,C)\}.$$

The pruned transition tree for ac($\mathcal{T} \times \bar{\mathcal{T}}$)/$\rho$ is:



The transition system $ac(\mathcal{T} \times \bar{\mathcal{T}})/\rho$ is the "most complex" transition system which is a homomorphic image of both $\mathcal{T}$ and $\bar{\mathcal{T}}$. There is a universal lower bound $<\mathcal{F}>$ in the lattice $(\{<\mathcal{T}>\}, \rightarrow)$. The transition system $\mathcal{F}$ is the "free" transition system

$$\mathcal{F} = (A_I, A_I^*, e, \varphi),$$

where $\varphi(x,a) = xa$. The equivalence relations with S.P. for $\mathcal{F}$ are precisely the right congruences over A_I^*, and any transition system $\mathcal{T}$ is isomorphic to $\mathcal{F}/\perp_{\mathcal{T}}$. There is also a universal upper bound $<\mathcal{Y}>$, where $\mathcal{Y}$ is a transition system with only one state.

7. SEQUENTIAL MACHINES WITH INPUT ALPHABET A_I

Building on the results of the previous section, we shall next develop the lattice of types of sequential machines with input alphabet A_I. We shall also derive and apply a necessary and sufficient condition for a function to be representable by some finite-state sequential machine, and shall show that any function with domain A_I^* has a unique "least complex" machine representing it. We shall also provide an algorithm for determining the machine with a minimum number of states which represents the same function as a given finite-state machine. Our development in this chapter as well as Chapters 2, 3, 5 and 6, is largely derived from [8], and from the exposition of [10] given in [8].

We have already seen an important equivalence relation over A_I^* which may be associated with a sequential machine

$$\mathcal{M} = (A_I, Q, A_O, q_0, \lambda, \delta),$$

namely the equiresponse relation $\perp_{\mathcal{M}}$, determined as follows:

$$x \perp_{\mathcal{M}} y \iff rp_{\mathcal{M}}(x) = rp_{\mathcal{M}}(y).$$

It is natural to introduce a second equivalence relation, the <u>equiresult relation</u> $\Delta_{\mathcal{M}}$, as follows:

$$x \Delta_{\mathcal{M}} y \iff res_{\mathcal{M}}(x) = res_{\mathcal{M}}(y).$$

As we have shown, $\perp_{\mathcal{M}}$ is a right congruence. Also, $\perp_{\mathcal{M}} \leq \Delta_{\mathcal{M}}$;

i.e. $$x \perp_{\mathcal{M}} y \implies x \Delta_{\mathcal{M}} y.$$

This is evident from the definition of the function $res_{\mathcal{M}}(x)$ as $\delta(rp_{\mathcal{M}}(x))$;

clearly, $$rp_{\mathcal{M}}(x) = rp_{\mathcal{M}}(y) \implies res_{\mathcal{M}}(x) = res_{\mathcal{M}}(y).$$

Let R and S be equivalence relations over A_I^* such that R is a right congruence and $R \leq S$. Then there is a natural way to construct a machine $\mathcal{M}$ such that $R = \perp_{\mathcal{M}}$ and $S = \Delta_{\mathcal{M}}$. For $x \in A_I^*$, let [x] denote the equivalence class associated with R to which x belongs, and let $< x >$ denote the equivalence class of x in S. Then

$$\mathcal{M} = (A_I, \{[x], x \in A_I^*\}, \{< x >, x \in A_I^*\}, [e], \lambda', \delta'),$$

where $$\lambda'([x],a) = [xa],$$

and $$\delta'([x]) = < x >.$$

We omit the simple proofs that λ' and δ' are well defined, and that $\mathcal{M}$ has the stipulated properties. Summing up, we have the following theorem.

Theorem 1

Let (R,S) be an ordered pair of equivalence relations over A_I^*. Then there exists a sequential machine $\mathcal{M}$ such that $R = \perp_{\mathcal{M}}$ and $S = \Delta_{\mathcal{M}}$ if and only if R is a right congruence and $R \leq S$.

Corollary 1

There exists a finite-state machine $\mathcal{M}$ with $R = \perp_{\mathcal{M}}$ and $S = \Delta_{\mathcal{M}}$ if and only if R is a right congruence of finite rank, and $R \leq S$.

Corollary 2

Let f be a function from A_I^* to A_o. Then f is representable by a finite-state sequential machine if and only if the equivalence relation $\equiv_f$ given by

$$x \equiv_f y \iff f(x) = f(y)$$

has, as a refinement, a right congruence of finite rank (R is a refinement of S if $R \leq S$).

Later, we shall illustrate the usefulness of Corollary 2.

A lattice ordering on the ordered pairs (R,S) satisfying the conditions of Theorem 1 can be introduced in a natural way, as follows:

$$(R,S) \leq (R',S') \iff R \leq R' \text{ and } S \leq S'.$$

Let this lattice be denoted $(\mathcal{P}, \leq)$. We could then say that the machines $\mathcal{M}$ and $\mathcal{N}$ are of the same type if

$$(\perp_{\mathcal{M}}, \Delta_{\mathcal{M}}) = (\perp_{\mathcal{N}}, \Delta_{\mathcal{N}}),$$

and a lattice ordering on machine types would be induced. Following the strategy of the previous section, we prefer to develop the lattice of machine types from the point of view of homomorphisms.

Let

$$\mathcal{M} = (A_I, Q, A_o, q_0, \lambda, \delta)$$

and

$$\bar{\mathcal{M}} = (A_I, \bar{Q}, \bar{A}_o, \bar{q}_0, \bar{\lambda}, \bar{\delta}).$$

We assume throughout the following discussion that all states are accessible, and that the functions δ: $Q \to A_o$ and $\bar{\delta}$: $\bar{Q} \to \bar{A}_o$ are both onto. A homomorphism from $\mathcal{M}$ onto $\bar{\mathcal{M}}$ is a pair of functions,

$$h: \; Q \xrightarrow{\text{onto}} \bar{Q} \quad \text{and} \quad \varphi: \; A_o \xrightarrow{\text{onto}} \bar{A}_o$$

such that

i) $h(q_0) = \bar{q}_0$

ii) $h(\lambda(q,a)) = \bar{\lambda}(h(q),a)$

iii) $\bar{\delta}(h(q)) = \varphi(\delta(q))$.

Conditions (i) and (ii) are, of course, familiar from our discussion of transition systems. Condition (iii) states that the output of the homomorphic image of q is the homomorphic image of the output of q. Applying

Lemma 6.1, we have the result that, for all $x \in A_I^*$,

$$rp_{\bar{\mathcal{M}}}(x) = h(rp_{\mathcal{M}}(x)). \tag{1}$$

Now,
$$\begin{aligned} res_{\mathcal{M}}(x) &= \bar{\delta}(rp_{\bar{\mathcal{M}}}(x)) \\ &= \bar{\delta}(h(rp_{\mathcal{M}}(x)) \\ &= \varphi(\delta(rp_{\mathcal{M}}(x)) \\ &= \varphi(res_{\mathcal{M}}(x)). \end{aligned}$$

Thus, for all $x \in A_I^*$,

$$res_{\bar{\mathcal{M}}}(x) = \varphi(res_{\mathcal{M}}(x)). \tag{2}$$

Let $\mathcal{M} \to \bar{\mathcal{M}}$ denote that there exists a homomorphism from $\mathcal{M}$ onto $\bar{\mathcal{M}}$. Then the following lemma is a consequence of equations (1) and (2).

Lemma 1

If $\mathcal{M} \to \bar{\mathcal{M}}$, then $(\perp_{\mathcal{M}}, \Delta_{\mathcal{M}}) \leq (\perp_{\bar{\mathcal{M}}}, \Delta_{\bar{\mathcal{M}}})$.

The converse of Lemma 1 is quite easy.

Lemma 2

If $(\perp_{\mathcal{M}}, \Delta_{\mathcal{M}}) \leq (\perp_{\bar{\mathcal{M}}}, \Delta_{\bar{\mathcal{M}}})$, then $\mathcal{M} \to \bar{\mathcal{M}}$.

PROOF: The required function h and φ are constructed as follows. If there exists $x \in A_I^*$ such that

$$rp_{\mathcal{M}}(x) = q \text{ and } rp_{\bar{\mathcal{M}}}(x) = \bar{q},$$

then $h(q) = \bar{q}$.

If $h(q) = \bar{q}$, then $\varphi(q) = \bar{\delta}(\bar{q})$.

We omit the simple proofs that h and φ are well defined functions, and that they determine a homomorphism from $\mathcal{M}$ onto $\bar{\mathcal{M}}$.

Q.E.D.

Combining Lemmas 1 and 2, we have the following result.

Theorem 2

Let $\mathcal{M}$ and $\bar{\mathcal{M}}$ be such that all states are accessible, and the functions $\delta\colon Q \to A_0$ and $\bar{\delta}\colon \bar{Q} \to \bar{A}_0$, are onto. Then $\mathcal{M} \to \bar{\mathcal{M}} \iff (\perp_{\mathcal{M}}, \Delta_{\mathcal{M}}) \leq (\perp_{\bar{\mathcal{M}}}, \Delta_{\bar{\mathcal{M}}})$.

A strong homomorphism from $\mathcal{M}$ onto $\bar{\mathcal{M}}$ is a homomorphism such that the function φ is one-to-one. An isomorphism from $\mathcal{M}$ onto $\bar{\mathcal{M}}$ is a homomorphism such that the functions h and φ are both one-to-one. Let the expression $\mathcal{M} \twoheadrightarrow \bar{\mathcal{M}}$ denote the existence of a strong homomorphism from $\mathcal{M}$ onto $\bar{\mathcal{M}}$, and let the existence of an isomorphism be denoted by $\mathcal{M} \leftrightarrow \bar{\mathcal{M}}$. It is evident that $\mathcal{M} \leftrightarrow \bar{\mathcal{M}}$ if and only if $\bar{\mathcal{M}} \leftrightarrow \mathcal{M}$. The methods used to establish Theorem 2 yield the following corollaries.

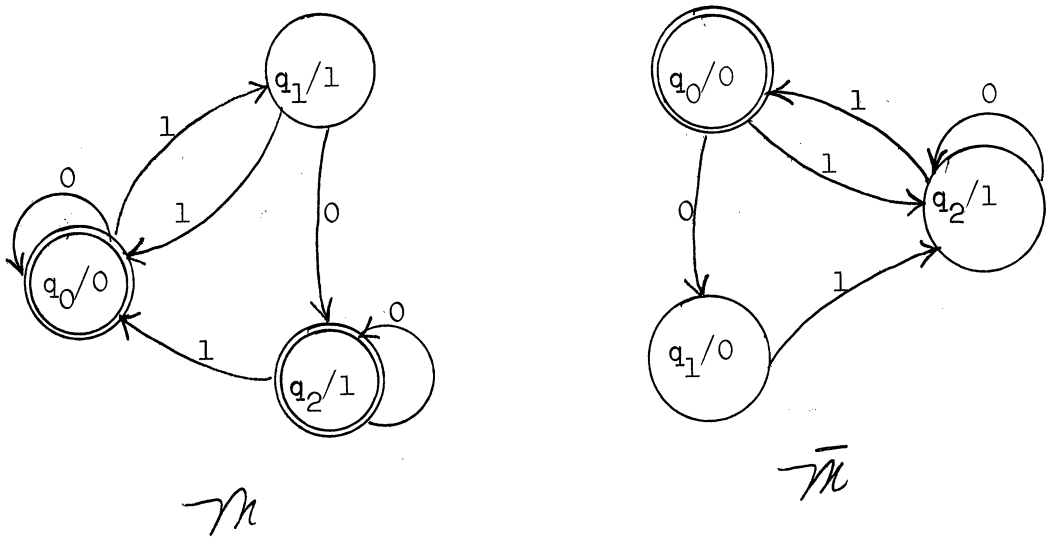
Corollary 3

$\mathcal{M} \twoheadrightarrow \bar{\mathcal{M}}$ if and only if $\perp_{\mathcal{M}} \leq \perp_{\bar{\mathcal{M}}}$ and $\Delta_{\mathcal{M}} = \Delta_{\bar{\mathcal{M}}}$.

Corollary 4

$\mathcal{M} \leftrightarrow \bar{\mathcal{M}}$ if and only if $(\perp_{\mathcal{M}}, \Delta_{\mathcal{M}}) = (\perp_{\bar{\mathcal{M}}}, \Delta_{\bar{\mathcal{M}}})$.

Two machines $\mathcal{M}$ and $\bar{\mathcal{M}}$ may represent the same function, and therefore have the same equiresult relation, even though their equiresponse relations are incomparable, so that neither is a homomorphic image of the other. The following two machines provide such an example.



The functions $res_{\mathcal{M}}(x)$ and $res_{\bar{\mathcal{M}}}(x)$ are identical; each function is equal to 1 if and only if the number of 1's in the sequence x is odd.

It is also true, of course, that two machines may have the same equiresponse relation and have incomparable equiresult relations. Finally, it is possible to have $\perp_{\mathcal{M}} \leq \perp_{\bar{\mathcal{M}}}$ and $\Delta_{\bar{\mathcal{M}}} \leq \perp_{\mathcal{M}}$.

The relation $\leftrightarrow$ is an equivalence relation. Let us call the equivalence classes associated with this relation machine types, and let the machine type of $\mathcal{M}$ be denoted $<\mathcal{M}>$. Following the approach used in dealing with transition systems, we define a partial ordering $\rightarrow$ on machine types as follows: $<\mathcal{M}> \rightarrow <\bar{\mathcal{M}}>$ if and only if, for all $\mathcal{M} \in <\mathcal{M}>$ and $\bar{\mathcal{M}} \in <\bar{\mathcal{M}}>$, $\mathcal{M} \rightarrow \bar{\mathcal{M}}$. Then the partially ordered system $(\{<\mathcal{M}>\}, \rightarrow)$ forms a lattice isomorphic with the lattice $(\mathcal{P}, \leq)$. The machine type $<\mathcal{M}>$ corresponds to the ordered pair $(\perp_{\mathcal{M}}, \Delta_{\mathcal{M}})$. The greatest lower bound and least upper bound of two machine types are easily obtained by analogy with the case of transition systems. Let

$$\mathcal{M} = (A_I, Q, A_o, q_0, \lambda, \delta),$$

and let

$$\bar{\mathcal{M}} = (A_I, \bar{Q}, \bar{A}_o, \bar{q}_0, \bar{\lambda}, \bar{\delta}).$$

Then $\mathcal{M} \times \bar{\mathcal{M}} = (A_I, Q \times \bar{Q}, A_o \times \bar{A}_o, (q_0,\bar{q}_0), \lambda', \delta')$

where $\lambda'((q,\bar{q}),a) = (\lambda(q,a), \bar{\lambda}(\bar{q},a))$,

and $\delta'(q,\bar{q}) = (\delta(q), \bar{\delta}(\bar{q}))$.

Let $ac(\mathcal{M} \times \bar{\mathcal{M}})$ be the machine obtained by retaining only the accessible states of $\mathcal{M} \times \bar{\mathcal{M}}$, and retaining only those elements of $A_o \times \bar{A}_o$ which are of the form $(\delta(q), \bar{\delta}(\bar{q}))$, where $(q,\bar{q})$ is accessible.

Lemma 3

Let $\mathcal{M}$ and $\bar{\mathcal{M}}$ be sequential machines belonging to the machine types $<\mathcal{M}>$ and $<\bar{\mathcal{M}}>$. Then

$$glb(<\mathcal{M}>, <\bar{\mathcal{M}}>) = <ac(\mathcal{M} \times \bar{\mathcal{M}})>.$$

The construction of $lub(<\mathcal{M}>, <\bar{\mathcal{M}}>)$ is a direct extension of the corresponding construction for transition systems. The equivalence relation with S.P. ρ over the states of $ac(\mathcal{M} \times \bar{\mathcal{M}})$ is defined exactly as it was defined in the last section:

$$(p,\bar{p}) \rho (q,\bar{q}) \iff \begin{cases} \text{there exists a sequence} \\ (q_1,\bar{q}_1),\ldots,(q_r,\bar{q}_r) \text{ of states} \\ \text{of } ac(\mathcal{M} \times \bar{\mathcal{M}}) \text{ such that} \end{cases}$$

$$(p,\bar{p}) = (q_1,\bar{q}_1),$$

$$(q,\bar{q}) = (q_r,\bar{q}_r),$$

and, for $i=1,2,\ldots,r-1$,

$$q_i = q_{i+1}$$

or

$$\bar{q}_i = \bar{q}_{i+1}.$$

A second equivalence relation μ has the following definition:

$$(p,\bar{p})\ \mu\ (q,\bar{q}) \iff \begin{cases} \text{there exists a sequence} \\ (q_1,\bar{q}_1),\ldots,(q_r,\bar{q}_r) \text{ of states} \\ \text{of } ac(\mathcal{M} \times \bar{\mathcal{M}}) \text{ such that} \end{cases}$$

$$(p,\bar{p}) = (q_1,\bar{q}_1),$$

$$(q,\bar{q}) = (q_r,\bar{q}_r),$$

and, for $i=1,2,\ldots,r-1$,

$$\delta(q_1) = \delta(q_{i+1})$$

or

$$\bar{\delta}(\bar{q}_i) = \bar{\delta}(\bar{q}_{i+1}).$$

It is evident that ρ is a refinement of μ. Let the equivalence class of $(q,\bar{q})$ in ρ be denoted $[q,\bar{q}]$, and let its equivalence class in μ be $< q,\bar{q} >$. Let the machine $\tilde{\mathcal{M}}$ be defined as follows:

$$\tilde{\mathcal{M}} = (A_I,\ \{[q,\bar{q}]\},\ \{< q,\bar{q} >\},\ [q_0,\bar{q}_0],\ \tilde{\lambda},\ \tilde{\delta}),$$

where

$$\tilde{\lambda}([q,\bar{q}],a) = [\bar{\lambda}(q,\bar{q}),a]$$

and

$$\tilde{\delta}([q,\bar{q}]) = < q,\bar{q} >.$$

The functions $\tilde{\lambda}$ and $\tilde{\delta}$ are well defined by virtue of the fact that ρ is an equivalence relation with S.P. and a refinement of μ.

Lemma 4

Let $\mathcal{M}$ and $\bar{\mathcal{M}}$ be sequential machines belonging to the machine types $< \mathcal{M} >$ and $< \bar{\mathcal{M}} >$, and let $\tilde{\mathcal{M}}$ be derived from $\mathcal{M}$ and $\bar{\mathcal{M}}$ by the construction given above. Then

$$\text{lub}(< \mathcal{M} >, < \bar{\mathcal{M}} >) = < \tilde{\mathcal{M}} > .$$

We omit the proof of Lemma 4.

The lattice $(\{<\mathcal{M}>\}, \rightarrow)$ has a universal lower bound, namely the "free" machine type of which the following machine $\mathcal{F}$ is a representative:

$$F = (A_I, A_I^*, A_I^*, e, \lambda, \delta),$$

where

$$\lambda(x,a) = xa,$$

and

$$\delta(x) = x.$$

There is also a universal upper bound, the machine type whose elements are all the one-state machines.

Let Δ be an equivalence relation over A_I^*. Then the set of all elements $(R,S) \in (\mathcal{P},\leq)$ such that $S = \Delta$ forms a sublattice of $(\mathcal{P},\leq)$:

$$glb((R,\Delta), (R',\Delta)) = (glb(R,R'),\Delta),$$

and

$$lub((R,\Delta), (R',\Delta)) = (lub(R,R'),\Delta).$$

We shall show that this sublattice has a universal upper bound. The element in the lattice $(\{<\mathcal{M}>\}, \rightarrow)$ corresponding to this universal upper bound is then the least upper bound of all machine types having the equiresult relation Δ.

Let the equivalence relation R_Δ over A_I^* be defined as follows:

$$x\ R_\Delta\ y \iff \text{for all } w \in A_I^*,\ xw\ \Delta\ yw.$$

Then R_Δ is a refinement of Δ, and R_Δ is a right congruence:

$$x\ R_\Delta\ y \Rightarrow \forall\ z \in A_I^*, \quad \forall\ w \in A_I^*$$

$$xzw\ \Delta\ yzw \Rightarrow xz\ R_\Delta\ yz.$$

Moreover, if R is any right congruence which is a refinement of Δ, then R is a refinement of R_Δ:

$$x\ R\ y \implies \text{for all } w \in A_I^*,\ xw\ R\ yw$$

$$\implies \text{for all } w \in A_I^*,\ xw\ \Delta\ yw$$

$$\implies x\ R_\Delta\ y$$

Therefore R_Δ is the least upper bound of all right congruences which refine Δ, and (R_Δ,Δ) is the universal upper bound of the sublattice of $(\mathcal{P},\leq)$ consisting of all the ordered pairs (R, Δ). The right congruence R_Δ is called the <u>right congruence induced by Δ</u>. The machine type $<\mathcal{M}>$ corresponding to (R_Δ,Δ) in the lattice $(\{<\mathcal{M}>\}, \rightarrow)$ is a strong homomorphic image of every machine type having the equiresult relation Δ.

Although the following result is a direct consequence of Corollary 1, we designate it as a theorem because of its importance.

<u>Theorem 3</u>

The equivalence relation Δ over A_I^* is the equiresult relation of a finite-state sequential machine if and only if R_Δ is of finite rank.

Let f be a function from A_I^* to a set A_o. Then any sequential machine realizing f has the equiresult relation $\equiv_f$:

$$x \equiv_f y \iff f(x) = f(y).$$

Then $R_{\equiv_f}$, the right congruence induced by $\equiv_f$, has the following definition:

$$x\ R_{\equiv_f}\ y \iff \text{for all } w,\quad f(xw) = f(yw).$$

For any element $x \in A_I^*$, let the function

$$f_x: A_I^* \to A_o,$$

be defined by the equation

$$f_x(w) = f(xw).$$

Then $x R_{\equiv_f} y$ if and only if the functions $f_x(w)$ and $f_y(w)$ are identical.

Theorem 4

A function f from A_I^* to A_o is representable by a finite-state machine if and only if the number of distinct functions $f_x(w)$ as x ranges over A_I^* is finite.

PROOF: If f is representable by a finite-state machine then $R_{\equiv_f}$ is of finite rank, which means that the number of distinct functions f_x is finite. Conversely, if the number of distinct functions f_x is finite, then $R_{\equiv_f}$ is of finite rank. Let the equivalence class of x in $R_{\equiv_f}$ be denoted [x]. Then the following finite-state machine realizes f:

$$\mathcal{M} = (A_I, \{[x], x \in A_I^*\}, A_o, [e], \lambda, \delta),$$

where

$$\lambda([x],a) = [xa]$$

and

$$\delta([x]) = f(x).$$

Q.E.D.

Let us make some applications of Theorem 4. First, we shall prove that, in a certain sense, a finite-state sequential machine cannot

recognize perfect squares. Let

$$f: \{a\}^* \to \{0,1\}$$

be defined as follows:

$$f(a^n) = \begin{cases} 1 \text{ if } n = s^2 \text{ for some nonnegative integer } s \\ 0 \text{ otherwise.} \end{cases}$$

Let k and ℓ be integers, with $k < \ell$. Then

$$f_{a^{k^2}}(a^{2k+1}) = f(a^{(k+1)^2}) = 1,$$

and

$$f_{a^{\ell^2}}(a^{2k+1}) = 0.$$

Therefore the function

$$f_{a^{k^2}}, \quad k=0,1,2,\ldots$$

are all distinct, and, by Theorem 4, f is not representable by a finite-state machine.

As another example, we shall show that the set of palindromes over an alphabet with more than one letter cannot be recognized by a finite-state machine. For $x \in A_I^*$, let $\tilde{x}$ denote the <u>reversal</u> of x:

$$\tilde{e} = e,$$

and, if $x=a_1a_2\ldots a_{n-1}a_n$,

$$\tilde{x} = a_n a_{n-1}\ldots a_2 a_1.$$

Then x is a <u>palindrome</u> if and only if $x = \tilde{x}$. If punctuation and spacing are ignored, the following is a palindrome over the Roman alphabet:

A MAN A PLAN A CANAL — PANAMA!

Let

$$f: A_I^* \to \{0,1\}$$

be defined as follows:

$$f(x) = \begin{cases} 1 & \text{if } x = \tilde{x} \\ 0 & \text{otherwise.} \end{cases}$$

If A_I has only the single element a, then every element of A_I^* is a palindrome, and the one-state machine with output 1 recognizes f. Suppose, however, that A_I has two distinct elements a and b. Let k and ℓ be nonnegative integers, with $k < \ell$. Then

$$f_{a^k b}(a^k) = 1,$$

and

$$f_{a^\ell b}(a^k) = 0.$$

Therefore there is an infinite number of distinct functions $f_{a^k b}$, k=0,1,2,..., and f is not representable by a finite-state machine.

Before proceeding to the next application, it will be convenient to derive a variant of Theorem 3. Let Δ be an equivalence relation over A_I^*. Then T_Δ, the (two-sided) congruence induced by Δ, is defined as follows:

$$x\ T_\Delta\ y \iff \forall\ v\ \ \forall\ w,\ vxw\ \Delta\ vyw.$$

Clearly, T_Δ is a refinement of Δ. Also, T_Δ is a two-sided congruence:

$$x\ T_\Delta\ y \implies \forall\ v\ \ \forall\ w\ \ \forall\ v'\ \ \forall\ w',$$

$$v'v\ xw\ w'\ \Delta\ v'vyw\ w' \implies \forall\ v\ \ \forall\ w,\ vxw\ T_\Delta\ vyw.$$

Finally, if T is a two-sided congruence which is a refinement of Δ,

$$\begin{aligned} x\ T\ y &\Longrightarrow \forall\ v\ \ \forall\ w,\ vxw\ T\ vyw \\ &\Longrightarrow \forall\ v\ \ \forall\ w,\ vxw\ \Delta\ vyw \\ &\Longrightarrow x\ T_{\Delta}\ y. \end{aligned}$$

Theorem 5

The equivalence relation Δ over A_I^* is the equiresult relation of a finite-state sequential machine if and only if T_{Δ} is of finite rank.

PROOF: Since T_{Δ} is a congruence, it is a right congruence, and $T_{\Delta} \leq R_{\Delta}$. If T_{Δ} is of finite rank, then so is R_{Δ}, and, therefore, Δ is the equiresult relation of a finite-state machine. Going in the other direction, suppose Δ is the equiresult relation of a finite-state machine $\mathcal{M}$ with n states. For the rest of this chapter, let the equivalence relation $\sim$ over A_I^* be defined with reference to the machine $\mathcal{M}$ as follows:

$$x \sim y \Longleftrightarrow \forall\ q,\ \lambda(q,x) = \lambda(q,y).$$

With x fixed, the function

$$\lambda_x(q) = \lambda(q,x)$$

is a transformation from Q into Q. Since Q has n elements, only n^n such transformations exist; therefore, $\sim$ is of finite rank. But $\sim$ is a two-sided congruence:

$$\begin{aligned} x \sim y &\Longrightarrow \forall\ q,\ \lambda(q,x) = \lambda(q,y) \\ &\Longrightarrow \forall\ q\ \ \forall\ v\ \ \lambda(q,vx) = \lambda(\lambda(q,v),x) \\ &\qquad\qquad\qquad\qquad = \lambda(\lambda(q,v),y) \\ &\qquad\qquad\qquad\qquad = \lambda(q,vy) \end{aligned}$$

$$\Longrightarrow \forall\, q \quad \forall\, v \quad \forall\, w,\ \lambda(q,uxw) = \lambda(\lambda(q,ux),w)$$
$$= \lambda(\lambda(q,uy),w)$$
$$= \lambda(q,uyw).$$

Also, $\sim$ is a refinement of Δ. Therefore, $\sim \leq T_{\Delta}$, and, since $\sim$ is of finite rank, so is T_{Δ}.

Corollary 5

A function f from A_I^* to A_o is representable by a finite-state machine if and only if the congruence relation $T_{\equiv_f}$ is of finite rank.

Let f be a function from A_I^* to A_o, and let the function $\tilde{f}$ be defined as follows:

$$\tilde{f}(\tilde{x}) = f(x).$$

Using Corollary 5 we shall prove that, if f is representable by a finite-state machine, so is $\tilde{f}$. If f is representable, then $T_{\equiv_f}$ is of finite rank. But

$$x\ T_{\equiv_f}\ y \Longrightarrow \forall\, v \quad \forall\, w,\ vxw \equiv_f vyw$$
$$\Longrightarrow \forall\, \tilde{w} \quad \forall\, \tilde{v},\ \tilde{w}\tilde{x}\tilde{v} \equiv_{\tilde{f}} \tilde{w}\tilde{y}\tilde{v}$$
$$\Longrightarrow \tilde{x}\ T_{\equiv_{\tilde{f}}}\ \tilde{y}.$$

Therefore, $T_{\equiv_{\tilde{f}}}$ is of finite rank, and, by Corollary 5, $\tilde{f}$ is representable by a finite-state machine.

Having shown how results associated with the induced right congruence R_{Δ} and the induced congruence T_{Δ} are used in determining whether certain

functions are representable by finite-state machines, we shall next apply the same ideas to a problem of machine minimization: given a machine $\mathcal{M}$, find a "least complex" machine having the same equiresult relation as $\mathcal{M}$. More precisely, the problem is this: given $\mathcal{M}$, find a representative $\mathcal{N}$ of the machine type $<\mathcal{N}>$ such that

$$\Delta_{<\mathcal{N}>} = \Delta_{\mathcal{M}} \quad \text{and} \quad \perp_{<\mathcal{N}>} = R_{\Delta_{\mathcal{M}}}.$$

From our knowledge of the properties of induced right congruences and of the isomorphism between the lattices $(\mathcal{P}, \leq)$ and $(\{<\mathcal{M}>\}, \rightarrow)$, we know the following facts.

1) A machine $\mathcal{N}$ with the desired properties exists.

2) There is a strong homomorphism from $\mathcal{M}$ onto $\mathcal{N}$.

3) If there is a strong homomorphism from $\mathcal{N}$ onto a machine $\overline{\mathcal{N}}$, then $\mathcal{N}$ and $\overline{\mathcal{N}}$ are isomorphic.

4) $\mathcal{N}$ is a finite-state machine if and only if $R_{\Delta_{\mathcal{M}}}$ is of finite rank, and, if $\mathcal{N}$ is finite-state, it has as many states as $R_{\Delta_{\mathcal{M}}}$ has equivalence classes.

5) If $\mathcal{M}$ is finite-state then, up to isomorphism, $\mathcal{N}$ is the unique minimum-state machine having the equiresult relation $\Delta_{\mathcal{M}}$.

6) If f is a function such that $\equiv_f$ is $\Delta_{\mathcal{M}}$, then $\mathcal{N}$ is isomorphic to the minimum-state machine realizing f.

Our next endeavor is to determine the strong homomorphism from $\mathcal{M}$ onto $\mathcal{N}$. In the case where $\mathcal{M} = (A_I, Q, A_O, q_0, \lambda, \delta)$ is finite-state, we will then be able to give an algorithm for the construction of $\mathcal{N}$.

Let an equivalence relation over Q be defined as follows: the states q_1 and q_2 are <u>indistinguishable</u> (denoted $q_1 \equiv q_2$) if,

$$\text{for all } w \in A_I^*$$

$$\delta(\lambda(q_1,w)) = \delta(\lambda(q_2,w));$$

q_1 and q_2 are <u>distinguishable</u> if they are not indistinguishable.

<u>Lemma 5</u>

Indistinguishability is an equivalence relation with S. P.

PROOF: It is evident that $\equiv$ is an equivalence relation. Also,

$$q_1 \equiv q_2 \Rightarrow \forall\, a \in A_I^*,\ \forall\, w \in A_I^*,\ \lambda(q_1,aw) = \lambda(q_2,aw)$$

$$\Rightarrow \forall\, w \in A_I^*,\ \lambda(\lambda(q_1,a),w) = \lambda(\lambda(q_2,a),w)$$

$$\Rightarrow \lambda(q_1,a) \equiv \lambda(q_2,a).$$

Therefore $\equiv$ has S.P.

<u>Lemma 6</u>

Let x and y be elements of A_I^*. Then

$$x\ R_\Delta\ y \iff rp_{\mathcal{M}}(x) \equiv rp_{\mathcal{M}}(y).$$

PROOF:

$$x\ R_\Delta\ y \iff \forall\, w,\ res_{\mathcal{M}}(xw) = res_{\mathcal{M}}(yw)$$

$$\iff \forall\, w,\ \delta(\lambda(rp_{\mathcal{M}}(x),w) = \delta(\lambda(rp_{\mathcal{M}}(y),w)$$

$$\iff rp_{\mathcal{M}}(x) \equiv rp_{\mathcal{M}}(y).$$

For $q \in Q$, let $[q]$ denote the equivalence class of q in the relation $\equiv$. We shall prove that the desired machine $\mathcal{N}$ is the image of the strong homomorphism in which h maps q into $[q]$, and φ is the identity function.

Theorem 6

Let $\mathcal{N} = (A_I, \{[q], q \in Q\}, A_O, [q_0], \bar{\lambda}, \bar{\delta})$,

where $\bar{\lambda}([q],a) = [\lambda(q,a)]$,

and $\bar{\delta}([q]) = \delta(q)$.

Then $(\perp_{\mathcal{N}}, \Delta_{\mathcal{N}}) = (R_{\Delta_{\mathcal{M}}}, \Delta_{\mathcal{M}})$.

PROOF: The function $\bar{\lambda}$ is well defined, since $\equiv$ has S. P., and $\bar{\delta}$ is well defined, since

$$q_1 \equiv q_2 \Longrightarrow \delta(q_1) = \delta(q_2).$$

Also, it is immediate that $\mathcal{M} \twoheadrightarrow \mathcal{N}$; the functions associated with the strong homomorphism are h, mapping q onto $[q]$, and φ, the identity function. Since the homomorphism is strong, $\Delta_{\mathcal{M}} = \Delta_{\mathcal{N}}$. It remains only to prove that $\perp_{\mathcal{N}} = R_{\Delta_{\mathcal{M}}}$. By Lemma 6,

$$\begin{aligned} x \, R_{\Delta_{\mathcal{M}}} \, y &\Longleftrightarrow rp_{\mathcal{M}}(x) \equiv rp_{\mathcal{M}}(y) \\ &\Longleftrightarrow [rp_{\mathcal{M}}(x)] = [rp_{\mathcal{M}}(y)] \\ &\Longleftrightarrow rp_{\mathcal{N}}(x) = rp_{\mathcal{N}}(y) \\ &\Longleftrightarrow x \perp_{\mathcal{N}} y. \end{aligned}$$

A sequential machine $\mathcal{N}$ is said to be reduced if $\perp_{\mathcal{N}} = R_{\Delta_{\mathcal{N}}}$, or, equivalently, if any two states of $\mathcal{N}$ are distinguishable. As an application of the foregoing development, we shall give an algorithm for

explicitly determining the equivalence relation $\equiv$, and thereby constructing $\mathcal{N}$, the reduced form of $\mathcal{M}$, when $\mathcal{M}$ is finite-state. This algorithm is useful in the design of sequential circuits, since the number of latches required in a circuit "realizing" an n-state machine is q, where $2^{q-1} < n \leq 2^q$; q is therefore a monotonic function of n.

The states q_1 and q_2 are k-indistinguishable (denoted $q_1 \underset{k}{\equiv} q_2$) if, for all $w \in A_I^*$ such that $lg(w) \leq k$,

$$\delta(\lambda(q_2,w)) = \delta(\lambda(q_2,w)).$$

The equivalence relation $\underset{k}{\equiv}$ does not necessarily have S.P.

Lemma 7

For $q_1 \in Q$ and $q_2 \in Q$,

i) $q_1 \underset{0}{\equiv} q_2 \iff \delta(q_1) = \delta(q_2)$

ii) for $k \geq 0$, $q_1 \underset{k+1}{\equiv} q_2 \iff q_1 \underset{k}{\equiv} q_2$ and, for all $a \in A_I$, $\lambda(q_1,a) \underset{k}{\equiv} \lambda(q_2,a)$.

PROOF:

$$q_1 \underset{0}{\equiv} q_2 \iff \delta(\lambda(q_1,e)) = \delta(\lambda(q_2,e))$$

$$\iff \delta(q_1) = \delta(q_2);$$

$q_1 \underset{k+1}{\equiv} q_q$ if and only if the following two conditions hold:

a) for all $w \ni lg(w) \leq k$, $\delta(\lambda(q_1,w)) = \delta(\lambda(q_2,w))$;

b) for all $w \ni lg(w) \leq k$, for all $a \in A_I$,

$$\delta(\lambda(q_1,aw) = \delta(\lambda(q_2,aw)).$$

Condition (a) is equivalent to $q_1 \underset{k}{\equiv} q_2$. Condition (b) is equivalent to: for all $a \in A_I$, and for all w such that $lg(w) \leq k$,

$$\delta(\lambda(\lambda(q_1,a),w)) = \delta(\lambda(\lambda(q_2,a),w));$$

i.e., for all $a \in A_I$,

$$\lambda(q_1,a) \underset{k}{\equiv} \lambda(q_2,a).$$

Corollary 6

If, for some $k \geq 0$, the relations $\underset{k}{\equiv}$ and $\underset{k+1}{\equiv}$ are equal, then each of them is equal to the relation $\equiv$.

Corollary 7

If $\mathcal{M}$ has n states and p output symbols, then the relations $\equiv$ and $\underset{n-p}{\equiv}$ are equal.

PROOF: As a consequence of the definitions of indistinguishability and k-indistinguishability,

$$\equiv \leq \underset{n-1}{\equiv} \leq \underset{n-2}{\equiv} \leq \cdots \leq \underset{1}{\equiv} \leq \underset{0}{\equiv} .$$

If, for some $k < n-p$, $\underset{k}{\equiv}$ is equal to $\underset{k+1}{\equiv}$, then $\underset{k}{\equiv}$ is equal to $\equiv$, and both are equal to $\underset{n-p}{\equiv}$. Otherwise, for $k=0,1,\ldots,n-p-1$, $\underset{k+1}{\equiv}$ strictly refines $\underset{k}{\equiv}$, and therefore has at least one more equivalence class than $\underset{k}{\equiv}$. Therefore, since $\underset{0}{\equiv}$ has p equivalence classes, $\underset{n-p}{\equiv}$ has at least $p + (n-p) = n$ equivalence classes. Then $\underset{n-p}{\equiv}$ is the identity relation, and, since the identity relation has no refinement except itself, $\underset{n-p}{\equiv}$ is equal to $\equiv$.

Q.E.D.

Corollary 7 has the following interpretation: if the states q_1 and q_2 are distinguishable, then there is an "experiment" $w \in A_I^*$ of length n-p or less that distinguishes them:

i.e., $$\delta(\lambda(q_1,w)) \neq \delta(\lambda(q_2,w)).$$

The algorithm for determining $\equiv$ is based on Lemma 7. The equivalence relations $\underset{0}{\equiv}$, $\underset{1}{\equiv}$, ... are determined successively until, for some k, $\underset{k}{\equiv}$ is equal to $\underset{k+1}{\equiv}$. By Corollary 7, this procedure must terminate.

Example

Consider the machine $\mathcal{M}$ specified by the following two tables

		A_I	
		a	b
Q	0*	5	1
	1	4	3
	2	2	5
	3	3	0
	4	1	2
	5	5	4

λ

q	$\delta(q)$
0	α
1	β
2	β
3	β
4	β
5	α

δ

The computation is summarized in the following table.

Equivalence relation	Equivalence classes
$\equiv_0$	{0,5}, {1,2,3,4}
$\equiv_1$	{0,5}, {1,4}, {2,3}
$\equiv_2$	{0,5}, {1,4}, {2,3}

The relation $\equiv$ has three equivalence classes:

$$A = \{0,5\},\ B = \{1,4\},\ \text{and}\ C = \{2,3\}.$$

The three-state machine $\mathcal{N}$ is specified as follows:

		A_I	
		a	b
$\bar{Q}$	A*	A	B
	B	B	C
	C	C	A

$\bar{q}$	$\bar{\delta}(\bar{q})$
A	α
B	β
C	γ

8. SEMIGROUPS AND SIMULATION

In this section we discuss two important congruence relations over the set of input words of a sequential machine $\mathcal{M}$. Properties of the associated factor monoids (the transition monoid of $\mathcal{M}$ and the function monoid of $\mathcal{M}$) are derived, and the Krohn-Rhodes normal form, involving the function monoid of $\mathcal{M}$, is introduced. The main result of the section shows how the capabilities of two machines, possibly with different input alphabets, can be compared by considering the normal forms of the machines. Most of the material of this section is drawn from [13], and from J. Myhill's paper, "Finite Automata, Semigroups and Simulation," University of Michigan Engineering Summer Conference on Automata Theory, 1963. [16].

Both of the two-sided congruence relations that we wish to consider occurred in the last section, but we reintroduce them here. Let

$$\mathcal{M} = (A_I, Q, A_O, q_0, \lambda, \delta)$$

be a sequential machine with all states accessible, and with every element of A_O equal to $\delta(q)$ for some $q \in Q$. The equivalence relations $\sim$ and φ over A_I^* are defined as follows:

$$x \sim y \iff \forall\, v \;\; \forall\, w,\ rp_{\mathcal{M}}(v\, x\, w) = rp_{\mathcal{M}}(v\, y\, w).$$

$$x \varphi y \iff \forall\, v \;\; \forall\, w,\ res_{\mathcal{M}}(v\, x\, w) = res_{\mathcal{M}}(v\, y\, w).$$

Then, using the notation of the last section, $\varphi = T_{\Delta_{\mathcal{M}}}$. It is evident from the definitions that $\sim$ and φ are two-sided congruences, and that $\sim \leq \varphi$.

Using the fact that all states of $\mathcal{M}$ are accessible, we can give an

alternate characterization of each of these relations. The first lemma shows the equivalence of the definition of $\sim$ with that on page 78.

Lemma 1

Let x and y be elements of A_I^*. Then

$$x \sim y \iff \forall\, q,\ \lambda(q,x) = \lambda(q,y).$$

PROOF: From the definition,

$$x \sim y \iff \forall\, v\ \ \forall\, w,\ rp_{\mathcal{M}}(vxw) = rp_{\mathcal{M}}(vyw).$$

Since

$$rp_{\mathcal{M}}(vx) = rp_{\mathcal{M}}(vy) \implies \forall\, w,\ rp_{\mathcal{M}}(vxw) = rp_{\mathcal{M}}(vyw),$$

$$x \sim y \iff \forall\, v,\ rp_{\mathcal{M}}(vx) = rp_{\mathcal{M}}(vy)$$

$$\iff \forall\, v,\ \lambda(rp_{\mathcal{M}}(v),x) = \lambda(rp_{\mathcal{M}}(v),y).$$

Since all states are accessible, every state q is $rp_{\mathcal{M}}(v)$ for some v; therefore,

$$x \sim y \iff \forall\, q,\ \lambda(q,x) = \lambda(q,y).$$

The following lemma relates the relation φ to the indistinguishability relation $\equiv$.

Lemma 2

Let x and y be elements of A_I^*. Then

$$x\ \varphi\ y \iff \forall\, q,\ \lambda(q,x) \equiv \lambda(q,y).$$

PROOF: From the definition,

$$x \varphi y \iff \forall v \ \forall w, \ res_{\mathcal{M}}(vxw) = res_{\mathcal{M}}(vyw)$$

$$\iff \forall v \ \forall w \ \delta(\lambda(rp_{\mathcal{M}}(v), xw)) = \delta(\lambda(rp_{\mathcal{M}}(v), yw)$$

$$\iff \forall v, \ \lambda(rp_{\mathcal{M}}(v), x) \equiv \lambda(rp_{\mathcal{M}}(v), y).$$

Since all states are accessible, every state q is $rp_{\mathcal{M}}(v)$ for some v; therefore

$$x \varphi y \iff \forall q, \ \lambda(q,x) \equiv \lambda(q,y).$$

Corollary 1

If $\mathcal{M}$ is reduced, then the congruence relations $\sim$ and φ are equal.

The converse of Corollary 1 is false.

The factor monoid $A_I^*/\sim$ is called the transition monoid of $\mathcal{M}$, and A_I^*/φ is called the function monoid of $\mathcal{M}$.

Just as every group is isomorphic to a group of permutations, every semigroup is isomorphic to a semigroup of transformations—functions from a set to itself. A semigroup of transformations isomorphic to $(S,\cdot)$ is $(\{R_a, \ a \in S\}, *)$, where R_a is the transformation of S defined by the equation

$$R_a(s) = sa,$$

and

$$R_a * R_b = R_{a\cdot b}.$$

It is immediate that the function φ: $a \to R_a$ is an isomorphism. The semigroup $(\{R_a, \ a \in S\}, *)$ is called the regular representation of $(S,\cdot)$.

The transition monoid of a machine $\mathcal{M}$ has another, more useful, representation as a monoid of functions. For $x \in A_I^*$, let the function

$$\lambda_x: Q \to Q,$$

be defined by the equation

$$\lambda_x(q) = \lambda(q,x).$$

Then, by Lemma 1, the functions $\lambda_x(q)$ and $\lambda_y(q)$ are identically equal if and only if $x \sim y$. Let the monoid of transformations $T_{\mathcal{M}}$ be defined as follows:

$$T_{\mathcal{M}} = (\{\lambda_x, x \in A_I^*\}, *),$$

where

$$\lambda_x * \lambda_y = \lambda_{xy}.$$

Lemma 3

The monoids $A_I^*/\sim$ and $T_{\mathcal{M}}$ are isomorphic.

PROOF: Let [x] be the equivalence class of x in the equivalence relation $\sim$. The function

$$\varphi: A_I^*/\sim \xrightarrow{\text{onto}} T_{\mathcal{M}}$$

given by the equation

$$\varphi([x]) = \lambda_x$$

is well defined and one-to-one, by Lemma 1. Clearly, φ is onto. Finally, φ is a homomorphism:

$$\varphi([x]\cdot[y]) = \varphi([xy]) = \lambda_{xy}.$$

$$\varphi([x]) * \varphi([y]) = \lambda_x * \lambda_y = \lambda_{xy}.$$

Let $(S,\cdot)$ be a monoid. A set $A \subset S$ is a <u>set of generators</u> of S if every element of S is a product of elements of A (by convention, the empty product is equal to the identity element).

<u>Lemma 4</u>

Any monoid $(S,\cdot)$ with a finite set of generators A is isomorphic with both the transition monoid and the function monoid of some sequential machine.

PROOF: The required machine is

$$\mathcal{M}(S) = (A, S, S, e, \lambda', \delta'),$$

where

$$\lambda'(s,a) = s\cdot a,$$

and

$$\delta'(s) = s.$$

Then the semigroup $T_{(S)}$ is isomorphic with the regular representation of S. The isomorphism φ is as follows: if

$$\{s_1, s_2, \ldots, s_m\} \subseteq A,$$

and

$$s_1\cdot s_2\cdot\ldots\cdot s_m = a,$$

then

$$\varphi(\lambda'_{s_1\ s_2\ \ldots\ s_m}) = R_a.$$

Then the semigroups

$$A^*/\sim,$$

$$T_{\mathcal{M}(S)},$$

$$(\{R_a,\ a \in S\},\ *),$$

and

$$(S,\cdot)$$

are all isomorphic. Since every state of $\mathcal{M}(S)$ has a different output, $\mathcal{M}(S)$ is a reduced machine, and A^*/φ is isomorphic with the other semigroups mentioned above.

Since $A_I^*/\sim$ can have as many as n^n elements if $\mathcal{M}$ has n states, its primary usefulness is theoretical, rather than computational. Nevertheless, it is instructive to work through the construction of this semigroup, and the isomorphic semigroup $T_{\mathcal{M}}$, in a "toy" example. It would be possible to formalize the methods we shall use in this example, but it is doubtful whether such a formalization would serve a useful purpose, in view of the typically unwieldly size of transition monoids.

Example 1

We consider a machine $\mathcal{M}$ with

$$Q = \{q_0,q_1,q_2\},$$

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

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

and λ and δ specified by the following tables.

		A_I	
		a	b
Q	q_0^*	q_1	q_0
	q_1	q_0	q_2
	q_2	q_2	q_1

λ

q	$\delta(q)$
q_0	1
q_1	0
q_2	0

δ

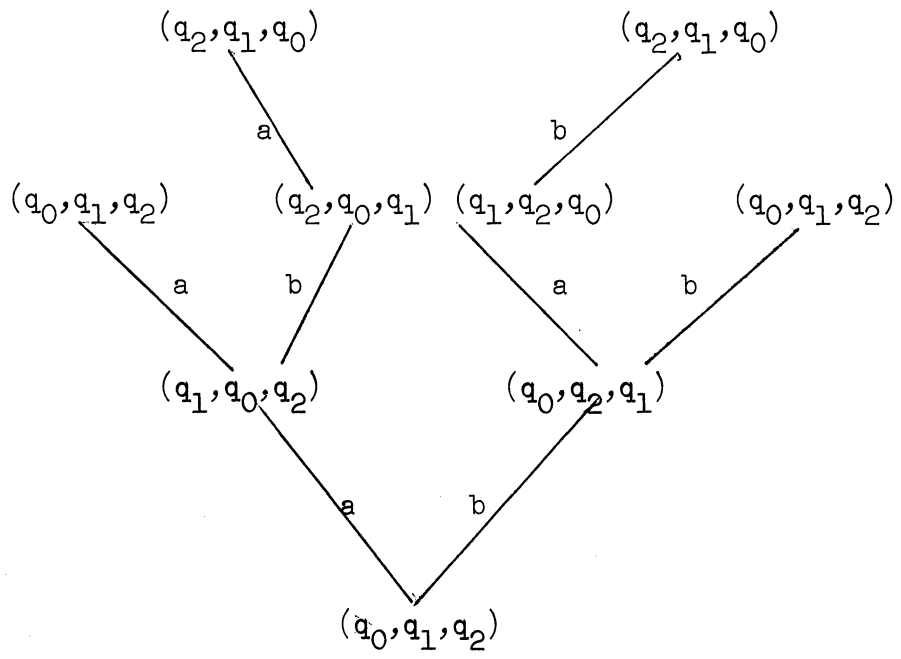
It is easy to check that $\mathcal{M}$ is reduced; therefore $A_I^*/\sim$ and A_I^*/φ are isomorphic. Since the functions $\lambda_a(q)$ and $\lambda_b(q)$ are permutations, every function $\lambda_x(q)$ is a composition of permutations, and, therefore, a permutation. Thus, $T_{\mathcal{M}}$ is a semigroup of permutations of the finite set Q, and $T_{\mathcal{M}}$ is necessarily a group. By isomorphism, $A_I^*/\sim$ and A_I^*/φ are also groups.

The following diagram shows part of the free tree generated by A_I. Each node n_x is labelled with the 3-tuple

$$(\lambda_x(q_0),\ \lambda_x(q_1),\ \lambda_x(q_2))$$

specifying the function λ_x. The label of the node n_{xa} may be obtained conveniently from the labels of n_x using the identity

$$\lambda_{xa}(q) = \lambda(\lambda_x(q), a).$$



Each of the six possible permutations of Q appears as a function λ_x in the subtree. Therefore, $T_{\mathcal{M}}$ is isomorphic to S_3, the symmetric group on three letters. Choosing the first (i.e., minimum-length, and leftmost within a given length) representative of each equivalence calss for the equivalence relation ~, we find that the equivalence classes are [e], [a], [b], [ab], [ba], and [aba]. Also, we note that the functions λ_{aa}, λ_{bb}, and λ_e are all identical, and that the functions λ_{aba} and λ_{bab} are identical. Therefore:

$$a\ a \sim e$$

$$b\ b \sim e$$

$$b\ a\ b \sim a\ b\ a.$$

The multiplication table for $A_I^*/\sim$ is as follows:

	[e]	[a]	[b]	[a b]	[b a]	[aba]
[e]	[e]	[a]	[b]	[a b]	[b a]	[aba]
[a]	[a]	[e]	[a b]	[b]	[aba]	[b a]
[b]	[b]	[b a]	[e]	[aba]	[a]	[a b]
[a b]	[a b]	[aba]	[a]	[b a]	[e]	[b]
[b a]	[b a]	[b]	[aba]	[e]	[a b]	[a]
[aba]	[aba]	[a b]	[b a]	[a]	[b]	[e]

The construction of this table was carried out with the aid of (1). For example, the computation of $[aba]\cdot[ba]$ was carried out as follows:

$$[aba]\cdot[ba] = [ababa]$$

Since $bab \sim aba$ $\qquad a(bab)a \sim a(aba)a$

Since $aa \sim e$ $\qquad (aa)baa \sim e(baa) = baa$

Since $aa \sim e$ $\qquad b(aa) \sim b(e) = b.$

Therefore,

$$[aba]\cdot[ba] = [b].$$

Since the set of identities (1) is sufficient to determine each of the entries in the above table, these identities are called <u>defining relations</u> for the semigroup $A_I^*/\sim$.

The defining relations are sufficient to determine the equivalence class to which a given word belongs. Consider, for example, the word

$$x = aababababbaababababa$$

$$x = (aa)baba(bb)(aa)babababa$$

Using the identities $aa \sim e$ and $bb \sim e$,

$$x \sim bababababaaba = (bab)a(bab)a(bab)a.$$

Since $bab \sim aba$

$$x \sim abaaabaaabaa = ab(aa)\ ab(aa)\ ab(aa).$$

Since $aa \sim e$

$$x \sim ababab = aba(bab)$$

Since $bab \sim aba$,

$$x \sim abaaba = ab(aa)ba$$

Therefore, $x \sim a(bb)a$

$$x \sim aa$$

$$x \sim e.$$

Therefore

$$[x] = [e],$$

$$\lambda_x = \lambda_e,$$

so that λ_x is the identity function. This tells us, for example, that

$$rp(x) = q_0.$$

We shall next show how concepts about semigroups and semigroup homomorphisms may be used to compare the capabilities of sequential machines, and in particular, to determine the conditions under which one machine can simulate another. In the following discussion, all the relevant characteristics of a machine are determined by the function which the machine represents. Thus, a machine is specified by a function

$$f: A^* \rightarrow B,$$

and we speak interchangeably of the machine f and the function f. We shall not exclude the possibility that A, the set of generators of the

free monoid A^*, may be infinite.

Let us recall that the congruence relation φ over A^* for the machine f is defined as follows:

$$x \varphi y \iff \forall v \quad \forall w \qquad f(vxw) = f(vyw).$$

The factor monoid A^*/φ is called the function monoid of f; its elements are equivalence classes associated with φ, and its multiplication operation * is as follows:

$$[x] * [y] = [xy],$$

where [x] denotes the equivalence class of x. Henceforth, this monoid will be denoted S_f. If $[x] \in S_f$, $x_1 \in [x]$, and $x_2 \in [x]$, then, clearly,

$$f(x_1) = f(x_2).$$

An equivalence relation ρ_f over the elements of S_f may be defined as follows:

$$[x] \rho_f [y] \iff f(x) = f(y).$$

The ordered pair (S_f,ρ_f) is called the normal form of f, and denoted NF(f). A reflexive, transitive relation over the set of normal forms of machines may be introduced as follows: NF(f) | NF(g) (' | ' is read 'divides') if S_g has a subsemigroup S, such that there is a homomorphism θ from S onto S_f satisfying the following conditions:
for all $s_1 \in S$, $s_2 \in S$,

$$s_1 \rho_g s_2 \Longrightarrow \theta(s_1) \rho_f \theta(s_2).$$

It should be noted that this definition involves only the abstract properties of S_f, as given by its multiplication table, and has nothing to do with the fact that the elements of S_f are equivalence classes. Also,

the definition applies even if the input and output alphabets of f are different from those for g. By contrast, the relation

$$(\perp_{\mathcal{M}}, \Delta_{\mathcal{M}}) \leq (\perp_{\mathcal{N}}, \Delta_{\mathcal{N}})$$

studied in the last section makes sense only for machines with the same input alphabet, and involves the state-transition structure of a machine as well as the function it represents.

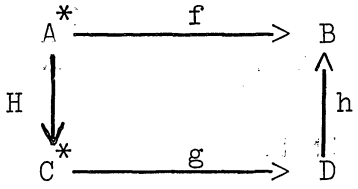
Let

$$f: A^* \to B$$

and

$$g: C^* \to D$$

be machines. We say that $f \mid g$ (f divides g) if there exists a homomorphism H from $(A^*,\cdot)$ into $(C^*,\cdot)$, where '$\cdot$' denotes concatenation, and a function h from D to B, such that the functions f(x) and hgH(x) are identical. The mappings involved in this definition may be diagrammed as follows.



Since H is a homomorphism, it is completely determined by the values it assumes for the elements of A; for, if

$$x = a_1 a_2 \cdots a_n,$$

where each of the a_i is an element of A, then

$$H(x) = H(a_1)H(a_2)\ldots H(a_n).$$

Thus, the statement $f \mid g$ may be interpreted as meaning that, given an appropriate input encoder and output decoder, the machine g can

simulate f. The input encoder replaces each letter a by the string H(a); the output decoder maps the final output symbol d into h(d), an element of B. In digital computer terms, we may say somewhat fancifully that, with suitable data format conversions, the computer g can simulate the computer f.

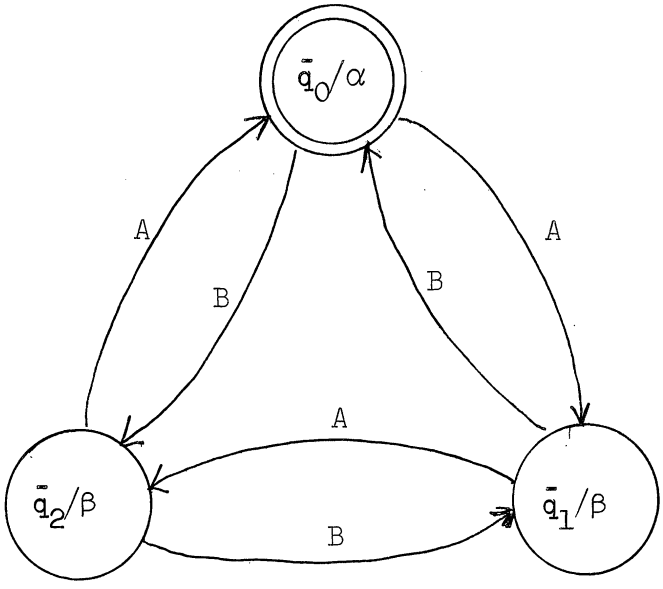
Theorem 1 (Krohn-Rhodes)

Let f and g be machines. Then $f \mid g$ if and only if $NF(f) \mid NF(g)$.

This theorem states that the capability of g to simulate f is determined by the properties of the function monoids S_f and S_g, and the equivalence relations ρ_f and ρ_g. Before entering into the proof of Theorem 1 let us consider an example.

Example 2

Let f be the function represented by the following sequential machine:



$f: \{A,B\}^* \rightarrow \{\alpha,\beta\}$

It is clear by inspection that the transition monoid of this machine is C_3, the cyclic group of order 3. Since the machine is reduced, $S_f \cong C_3$. The multiplication table for S_f is as follows:

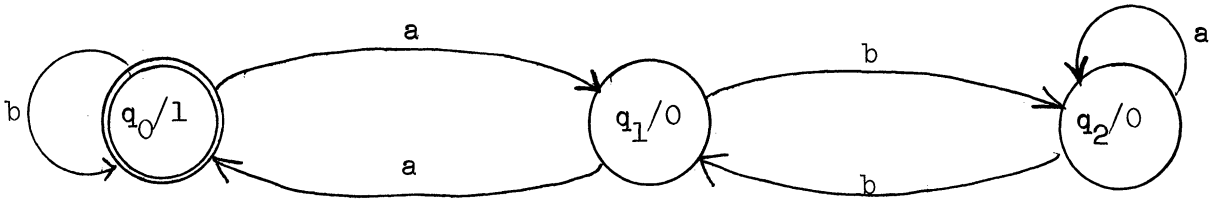
	[e]	[A]	[B]
[e]	[e]	[A]	[B]
[A]	[A]	[B]	[e]
[B]	[B]	[e]	[A]

Since $f(e) = \alpha$,

and $f(A) = f(B) = \beta$,

the equivalence relation ρ_f has the equivalence classes [e] , [A], [B] .

Let g be the function represented by the reduced machine that was considered in Example 1.



As we found in Example 1, the function monoid S_g is isomorphic to the symmetric group S_3, which has the subgroup C_3. The subgroup S of S_g isomorphic to C_3 has the following multiplication table.

	[e]	[ba]	[ab]
[e]	[e]	[ba]	[ab]
[ba]	[ba]	[ab]	[e]
[ab]	[ab]	[e]	[ba]

Also,

$$g(e) = 1,$$

$$g(ab) = g(ba) = 0;$$

$$[ab]\rho_g[ba],\ [ab]\not\rho_g[e].$$

Therefore, the following isomorphism θ from S onto S_f satisfies the condition $s_1\rho_g s_2 \Longrightarrow \theta(s_1)\rho_f\,\theta(s_2)$: $\theta([e]) = [e]$, $\theta([ba]) = A$, $\theta([ab]) = B$. Therefore, $NF(f) \mid NF(g)$. On the other hand, it is also true that $f \mid g$. The required homomorphism is given by

$$H(A) = ba,$$

$$H(B) = ab,$$

and the function h is given by

$$h(1) = \alpha,$$

$$h(0) = \beta.$$

In proving Theorem 1, it will be useful to associate with a function

$$f\colon A^* \to B,$$

another function

$$F\colon S_f^* \to B.$$

The function F is represented by the following sequential machine (which may have an infinite input alphabet):

$$(S_f,\ S_f,\ B,\ [e],\ \bar{\lambda},\ \bar{\delta}),$$

where

$$\bar{\lambda}(s,t) = s * t,$$

and, if $s = [x]$,

$$\bar{\delta}(s) = f(x).$$

Then

$$F([e]) = f(e),$$

and, in general,

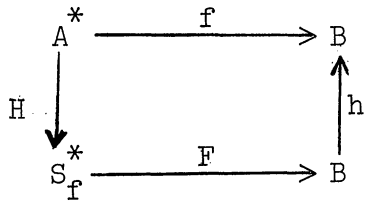
$$F([x_1][x_2]\ldots[x_n]) = f(x_1x_2\ldots x_n).$$

Note that $(S_f^*,\cdot)$ and $(S_f,*)$ are not the same monoid. Thus, for example, [e] is the identity element of S_f, but is a word of length 1 in S_f^*, whose identity element is e.

Lemma 5

Let f be a function from A^* to B, and let F be the associated function from S_f^* to B. Then $f \mid F$ and $F \mid f$.

PROOF: First, we show that $f \mid F$. The diagram showing the domains and ranges of the required homomorphism and function is as follows:



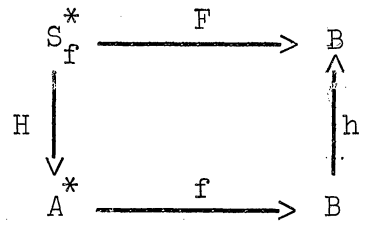
For all $a \in A$, let $H(a) = [a]$.

Then $H(a_1a_2\ldots a_n) = [a_1][a_2]\ldots[a_n]$,

and $F([a_1][a_2]\ldots[a_n]) = f(a_1a_2\ldots a_n)$.

If h is the identity function then f(x) is identically equal to hFH(x), as required.

It is equally easy to show that $F \mid f$. The diagram of mappings for this case is as follows:



The required homomorphism H is determined by the equation $H([x]) = x$, where x is an arbitrary, but fixed, element of [x]. Then

$$F([x_1]\cdot[x_2]\cdot\ldots\cdot[x_n]) = f(x_1x_2\ldots x_n)$$
$$= f(H([x_1][x_2]\ldots[x_n])).$$

If h is the identity function, then F is identically equal to hfH. This completes the proof of Lemma 5.

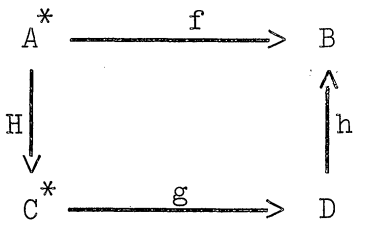
PROOF: (of Theorem 1)

$$f \mid g \iff NF(f) \mid NF(g).$$

We begin by showing that

$$f \mid g \implies NF(f) \mid NF(g).$$

If $f \mid g$, then $f = hgH$. The diagram of mappings is as follows:



Let S, a subset of S_g, be defined as follows:

$$S = \{[H(x)],\ x \in A^*\};$$

note that '[ ]' denotes 'equivalence class of the relation φ_g''. To prove that S is a semigroup (and therefore a subsemigroup of S_g) we

must show that $[H(x)]*[H(y)] \in S$.

But
$$[H(x)]*[H(y)] = [H(x)\cdot H(y)]$$
$$= [H(xy)].$$

Therefore, S is closed under *, the multiplication operation in S_f.

Now, to show that $NF(f) \mid NF(g)$, we must construct a suitable homomorphism from S onto S_f. First we show that

$$[H(x)] = [H(y)] \Longrightarrow [x] = [y]:$$

$$[H(x)] = [H(y)] \Longrightarrow \forall\, r \in C^* \quad \forall\, s \in C^*,$$

$$g(rH(x)s) = g(rH(y)s) \Longrightarrow \forall\, v \in A^* \quad \forall\, w \in A^*,\ g(H(v)H(x)H(w)) = g(H(v)H(y)H(w)$$

$$\Longrightarrow \forall\, v \quad \forall\, w,\ g(H(vxw)) = gH(vyw)$$

$$\Longrightarrow \forall\, v \quad \forall\, w, hgH(vxw) = hgH(vyw)$$

$$\Longrightarrow x\ \varphi\ y$$

$$\Longrightarrow [x] = [y].$$

In view of this result, the mapping

$$\theta\colon [H(x)] \to [x]$$

is well defined, and is a mapping onto S_f. Also

$$\theta([H(x)] * [H(y)]) = \theta([H(x)\ H(y)])$$
$$= \theta([H(xy)])$$
$$= [xy],$$

and

$$\theta([H(x)]) * \theta([H(y)]) = [x] * [y]$$
$$= [xy].$$

Therefore, θ is a homomorphism. Finally, we must show that, for $[H(x)] \in S$ and $[H(y)] \in S$,

$$[H(x)]\rho_g[H(y)] \Longrightarrow [x]\ \rho_f\ [y].$$

This is done as follows:

$$\begin{aligned}[H(x)]\ \rho_g\ [H(y)] &\Longrightarrow g(H(x)) = g(H(y)) \\ &\Longrightarrow hgH(x) = hgH(y) \\ &\Longrightarrow f(x) = f(y) \\ &\Longrightarrow [x]\ \rho_f\ [y].\end{aligned}$$

To sum up, S is a subsemigroup of S_g, and there is a homomorphism θ from S onto S_f such that

$$s_1\ \rho_g\ s_2 \Longrightarrow \theta(s_1)\ \rho_f\ \theta(s_2).$$

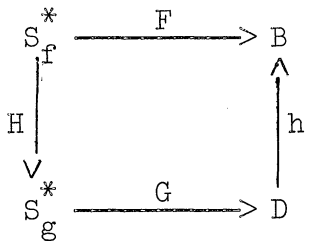
The second half of the proof requires us to show that

$$NF(f) \mid NF(g) \Longrightarrow f|g.$$

By Lemma 5 and the transitivity of the "divides" relation, an equivalent statement is:

$$NF(f) \mid NF(g) \Longrightarrow F \mid G,$$

where G is the function from S_g^* to D associated with g. The domains and ranges of the mappings to be constructed are shown in the following diagram.



Since $NF(f) \mid NF(g)$, there is a subsemigroup S of S_g, and a homomorphism

$$\theta\colon S \xrightarrow{\text{onto}} S_f,$$

such that

$$s_1 \rho_g s_2 \Longrightarrow \theta(s_1) \rho_f \theta(s_r).$$

The required homomorphism H from $(S_f^*, \cdot)$ to $(S_g^*, \cdot)$ is determined by the action of H on the elements of S_f.

Let H be chosen so that, for all $s \in S_f$,

$$H(s) = \bar{s},$$

where

$$\theta(\bar{s}) = s.$$

Such a choice is possible, since θ is onto.

Let $s_1 s_2 \ldots s_n$ be a word of S_f^*. Then

$$F(s_1 s_2 \ldots s_n) = F(s_1 * s_2 * \ldots * s_n),$$

and

$$\begin{aligned} GH(s_1 s_2 \ldots s_n) &= G(\bar{s}_1 \bar{s}_2 \ldots \bar{s}_n) \\ &= G(\bar{s}_1 * \bar{s}_2 * \ldots * \bar{s}_n). \end{aligned}$$

Since θ is a homomorphism,

$$\begin{aligned} \theta(\bar{s}_1 * \bar{s}_2 * \ldots * \bar{s}_n) &= \theta(\bar{s}_1) * \theta(\bar{s}_2) * \ldots * \theta(\bar{s}_n) \\ &= s_1 * s_2 * \ldots * s_n. \end{aligned}$$

Thus, h can be constructed if, for all $\bar{s} \in S_g$, the value of $F(\theta(\bar{s}))$ can be predicted from the value of $G(\bar{s})$. This is equivalent to the statement that

$$G(\bar{s}) = G(\bar{t}) \Longrightarrow F(\theta(\bar{s})) = F(\theta(\bar{t})).$$

This is true, since

$$G(\bar{s}) = G(\bar{t}) \implies \bar{s}\ \rho_g\ \bar{t}$$

$$\implies \theta(\bar{s})\ \rho_f\ \theta(\bar{t})$$

$$\implies F(\theta(\bar{s})) = F(\theta(\bar{t})).$$

This completes the proof of Theorem 1, and with it our discussion of simulation. The Krohn-Rhodes Normal Form will appear again later, in our discussion of the decomposition of sequential machines.

9. REGULAR EVENTS AND REPRESENTABLE FUNCTIONS

In Chapter 7 we proved that a function f is representable by a finite-state sequential machine if and only if the right congruence $R_{\equiv_f}$ is of finite rank, or, equivalently, if and only if the number of distinct functions f_x is finite. In the present section we introduce regular events and regular expressions, and thereby obtain another characterization of representable functions. Much of the material we cover can be found in: J. A. Brzozowski [7] and D. Arden [5].

To begin with, we shall consider a restricted class of finite-state machines, the finite automata, or finite-state acceptors. A finite automaton has only two output symbols, A and R. A sequence x is said to be accepted if res(x) = A, and rejected otherwise. It is convenient to specify a finite automaton as a quintuple

$$\mathcal{A} = (A_I, Q, q_0, F, \lambda)$$

where $F \subset Q$, and $\delta(q) = A$ if and only if $q \in F$;
the set F is called the set of final states. Note that the word x is accepted if and only if $rp(x) \in F$.

An event over A_I^* is a subset of A_I^*. The finite automaton represents the event P if P is the set of words accepted by $\mathcal{A}$. An event is representable if and only if some finite automaton represents it.

The characterization of representable functions in terms of right congruences yields, as an immediate consequence, a characterization of representable events. For any event P over A_I^*, and any word $x \in A_I^*$, $D_x(P)$, the derivative of P with respect to x, is defined as follows:

$$D_x(P) = \{y \mid xy \in P\}.$$

Lemma 1

The event P is representable if and only if the number of distinct derivatives $D_x(P)$, as x ranges over A_I^*, is finite.

PROOF: By definition, P is representable if and only if the following function f(x) is representable by a finite automaton:

$$f(x) = \begin{cases} A \text{ if } x \in P \\ \\ R \text{ otherwise.} \end{cases}$$

But, for x and $y \in A_I^*$, the functions f_x and f_y are identical if and only if, for all w,

$$xw \in P \iff yw \in P;$$

i.e., if and only if

$$D_x(P) = D_y(P).$$

Therefore, the number of distinct functions $f_x(w)$ is finite if and only if the number of distinct derivatives $D_x(P)$ is finite.

The main result of the present section is that the representable events over A_I^* are precisely the regular events over A_I^*, which we now proceed to define. In the definition, we shall use the following operations on events:

union $\quad P \cup Q = \{x \mid x \in P \text{ or } x \in Q\}$

concatenation $\quad P \cdot Q = \{xy \mid x \in P \text{ and } y \in Q\}$

n-th power $\quad P^n$, where n is a nonnegative integer, is defined recursively as follows:

$P^0 = \{e\}$; for $n > 0$, $P^n = P^{n-1} \cdot P$.

iteration $\quad P^* = \bigcup_{n=0}^{\infty} P^n = \{e\} \cup P \cup P^2 \cup \ldots$.

Let A be a finite alphabet. The definition of regular event over A^* is recursive.

i) the empty set φ is regular;

ii) the set $\{e\}$ is regular;

iii) for each $a \in A$, $\{a\}$ is regular;

iv) if P and Q are regular, then $P \cup Q$ and $P \cdot Q$ are regular;

v) if P is regular, then P^* is regular;

vi) no event is regular unless its being so follows from (i) ... (v).

Regular events may be denoted by well-formed formulas constructed from the constant events φ, $\{e\}$, and $\{a\}$, $a \in A$, together with the binary operators v and $\cdot$, and the unary operator *; such formulas are called regular expressions. In writing regular expressions, certain conventions are useful. Any set consisting of a single word may be written without braces, and the '$\cdot$' between concatenated events may be omitted. Thus, instead of $\{a\}\cdot\{b\}\cdot\{c\cdot d\}$ we may write abcd.

The operation * has precedence over $\cdot$ and $\cup$, and $\cdot$ has precedence over v; thus: $P^* \cup (Q\cdot R)$ may be written as $P^* \cup QR$.

The following identities, valid for all events P, Q, and R, are direct consequences of the definitions:

$$P \cup P = P$$

$$P \cup Q = Q \cup P$$

$$P \cup (Q \cup R) = (P \cup Q) \cup R$$

$$P(QR) = (PQ)R$$

$$PQ \cup PR = P(Q \cup R)$$

$$PR \cup QR = (P \cup Q)R$$

$$(P \cup Q)^* = (P^* \cup Q^*)^* = (P^* Q^*)^*$$

$$Pe = eP = P$$

$$P\varphi = \varphi P = \varphi$$

$$\varphi^* = e$$

$$\varphi \cup R = R$$

The associative laws enable us to mit parentheses according to the usual convention:

$$(P \cup Q) \cup R = P \cup Q \cup R.$$

It is useful to acquire facility in translating verbal descriptions of events into regular expression notation, when possible. For example, the event consisting of those sequences of a's and b's which end in b, followed by an odd number of a's, followed by an even number of b's is

$$(a \cup b)^* b(aa)^* a(bb)^*.$$

The event consisting of those sequences of a's and b's which either do not include three consecutive a's, or have two consecutive b's since the last three consecutive a's, is

$$(e \cup (a \cup b)^* bb)(b \cup ab \cup aab)^*(e \cup a \cup aa).$$

The importance of regular events is underscored by the following thecrem, which is the main result of the present section.

Theorem 1

An event P over A^* is representable if and only if it is regular.

As by-products of the proof of Theorem 1, we shall obtain two useful algorithms. The first of these, given a finite automaton representing the event P, produces a regular expression for P. The second, given a regular expression for P, produces a finite automaton representing P. The proof of Theorem 1 consists of the proofs of corollaries 1 and 5.

We begin by proving that every representable event is regular. Let P be the event represented by the finite automaton

$$\mathcal{a} = \{A_I, Q, q_0, F, \lambda\},$$

where

$$A_I = \{a_1, a_2, \ldots, a_m\},$$

and

$$Q = \{q_0, q_1, \ldots, q_{n-1}\}.$$

For $j=0,1,\ldots,n-1$, let

$$X_j = \{x \mid rp_{\mathcal{a}}(x) = q_j\}.$$

Then

$$P = \bigcup_{\{j \mid q_j \in F\}} X_j.$$

In order to prove that P is regular, it is sufficient to show that, for each j, X_j is a regular event.

For any two indices i and j in the set $\{0,1,\ldots,n-1\}$, let α_{ij} be the following regular event:

$$x \in \alpha_{ij} \iff x \in A_I$$

and

$$\lambda(q_i,x) = q_j.$$

In other words, α_{ij} is the set of input symbols which cause a transition of $\mathcal{A}$ from q_i to q_j. Now, if $x \in A_I^*$ and $a \in A_I$,

$$rp(xa) = \lambda(rp(x),a)$$

therefore

$$rp(xa) = q_j \iff \text{for some } i,\ rp(x) = q_i \text{ and } a \in \alpha_{ij}.$$

Accordingly, the events X_j are related by the following system of simultaneous equations

$$X_0 = e \cup \bigcup_{i=0}^{n-1} X_i\alpha_{i0}$$

(1)
$$X_j = \bigcup_{i=0}^{n-1} X_i\alpha_{ij}, \quad j=1,2,\ldots,n-1.$$

We shall prove that the system (1) has a unique solution, and that, in this solution, each event X_j is regular. The proof requires the following lemma.

Lemma 2 (Arden)

Let A and B be events such that $e \notin A$. Then the equation (2)

$$X = XA \cup B$$

has the unique solution

$$X = BA^*.$$

PROOF: Let X_0 be a solution of (2). By induction on n, $BA^n \subset X_0$ for all $n \geq 0$; therefore, $BA^* \subset X_0$. Assuming that

$$X_0 \neq BA^*,$$

let w be a shortest word of X_0 not in BA^*. Then

$$w \in X_0 A \cup B,$$

and since $w \notin B$, $w \in X_0 A$; i.e., $W = yz$, where $y \in X_0$ and $z \in A$. Since $e \notin A$,

$$lg(z) \geq 1,$$

and

$$lg(y) < lg(w).$$

By the minimality of lg(w), $y \in BA^*$; but then

$$w \in (BA^*)A \subset BA^*.$$

This contradiction establishes that

$$X_0 = BA^*.$$

Theorem 2

For i=1,2,...,n, and j=1,2,...,n, let a_{ij} be a regular event not including the null sequence e, and let b_i be a regular event. Then the system of equations

$$Y_j = \bigcup_{i=1}^{n} Y_i \, a_{ij} \vee b_j, \qquad j=1,2,\ldots,n$$

has a unique solution, and each of the events Y_j in this solution is regular.

PROOF: The proof is by induction on n.

Basis. n=1. By Lemma 2, $Y_1 = b_1 a_{11}^*$, which is regular.

Induction step. Suppose the result holds for k=n-1. Using Lemma 2, we may solve for Y in terms of the other variables:

$$Y_1 = ((\bigcup_{i=2}^{n} Y_i a_{i1}) \cup b_1) a_{11}^*. \tag{3}$$

Substituting in the other equations,

$$Y_j = \bigcup_{i=2}^{n} Y_i (a_{ij} \cup a_{i1} a_{11}^* a_{ij}) \cup (b_j \cup b_1 a_{11}^* a_{ij}), \quad j=2,3,\ldots,n.$$

Each event

$$a_{ij} \cup a_{i1} a_{11}^* a_{ij}$$

is regular and does not contain the null sequence, and each event

$$b_j \cup b_1 a_{11}^* a_{1j}$$

is regular. To see this, one uses the facts that the regular events are closed under $\cdot$, $\cup$, and $*$, and that $e \in AB$ if and only if $e \in A$ and $e \in B$. Therefore the induction hypothesis applies, and, for $j=2,3,\ldots,n$, each event Y_j is a uniquely determined regular event; by (3), Y_1 is also a uniquely determined regular event.

Corollary 1

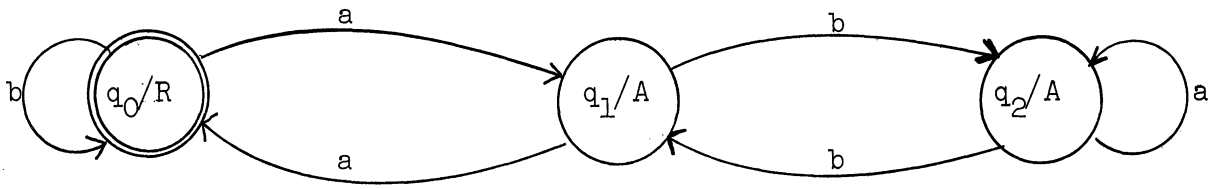
Every representable event is regular.

PROOF: Theorem 2 applies to the system (1); therefore, each event X_j is regular.

The proof of Theorem 2 suggests that a system such as (1) can be solved by elimination. This technique is applied in the following example.

Example 1

Consider the following finite automaton:



$$X_0 = e \cup X_0 b \cup X_1 a$$

$$X_1 = X_0 a \cup X_2 b$$

$$X_2 = X_1 b \cup X_2 a$$

Solving the first equation for X_0:

$$X_0 = (e \cup X_1 a)b^*$$

Substituting in the second equation:

$$X_1 = (e \cup X_1 a)b^* a \cup X_2 b = b^* a \cup X_1 ab^* a \cup X_2 b.$$

Solving the third equation for X_2:

$$X_2 = X_1 ba^*$$

Substituting in the equation for X_1:

$$X_1 = b^* a \cup X_1 ab^* a \cup X_1 ba^* b$$

Solving for X_1:

$$X_1 = b^* a(ab^* a \cup ba^* b)^*$$

$$X_2 = b^* a(ab^* a \cup ba^* b)^* ba^*$$

$$P = X_1 \cup X_2 = b^* a(ab^* a \cup ba^* b)(e \cup ba^*).$$

Before going on to prove that every regular event is realizable, let us derive some further applications of Theorem 2. A nondeterministic finite automaton $\mathcal{N}$ is a quintuple

$$\mathcal{N} = (A_I, Q, Q_0, F, L),$$

where

$$Q = (q_0, q_1, \ldots, q_{n-1}),$$

$$Q_0 \subset Q, \quad F \subset Q,$$

and L is a function from $Q \times A_I$ to subsets of Q. The set Q_0 is the set of initial states, and F is the set of final states. A nondeterministic automaton may be interpreted as having a number of choices for its "move". When it is in a given state q and receives an input a, it may go to any state in the set L(q,a); of course, L(q,a) may in some cases be the empty set. A word

$$x = a_0 a_1 \cdots a_{n-1}$$

is accepted by $\mathcal{N}$ if there exists a sequence of states $q^{(0)}, q^{(1)}, \ldots, q^{(n)}$ such that $q^{(0)} \in Q_0$, $q^{(n)} \in F$, and, for i=0,1,...,n-1,

$$q^{(i+1)} \in L(q^{(i)}, a_i).$$

We shall prove that the set of words accepted by a nondeterministic finite automaton is regular. For j=0,1,...,n, let X_j be the set of input words that can cause the automaton to reach q_j from some state in Q_0; then $e \in X_j$ if and only if $q_j \in Q_0$, and the word

$$x = a_0 a_1 \cdots a_n$$

is an element of X_j if and only if there exists a sequence of states $q^{(0)}, q^{(1)}, \ldots, q^{(n)}$ such that $q^{(0)} \in Q_0$,

$$q^{(n)} = q_j,$$

and for i=0,1,...,n-1,

$$q^{(i+1)} \in L(q^{(i)}, a_i).$$

Then

$$P = \bigcup_{\{j \mid q_j \in F\}} X_j .$$

Let

$$\alpha_{ij} = \{a_\ell \mid a_\ell \in A_I \text{ and } q_j \in L(q_i, a)\}.$$

Then the following system of equations holds:

$$\text{if } q_j \in Q_0,\ X_j = e \cup \bigcup_{i=0}^{n-1} X_i \alpha_{ij}$$

$$\text{if } q_j \notin Q_0,\ X_j = \bigcup_{i=0}^{n-1} X_i \alpha_{ij}.$$

Theorem 2 applies to this system of equations, and asserts that each set X_j is regular; therefore, P is regular. Thus, we have proved the following result.

Corollary 2

The set of words accepted by a nondeterministic finite automaton is regular.

Up to now, we have taken the point of view that a sequential machine computes a function from A_I^* to A_0; i.e., that it maps each input sequence into an output symbol. It is also possible to consider sequential machines as transducers, which map sequences to sequences. This point of view is generally applied to the class of generalized sequential machines, a variant of the Mealy machines. Nevertheless, for the sake of uniformity we shall continue to consider the Moore model. Let

$$\mathcal{M} = (A_I, Q, A_0, q_0, \lambda, \delta)$$

be a finite-state machine, and let the function seq(x) from A_I^* to A_0^* be defined as follows:

$$\begin{aligned} \text{seq}(e) &= \text{res}(e) \\ &= \delta(q_0); \end{aligned}$$

if $x = a_0, a_1, \ldots, a_{n-1}$, then

$$\text{seq}(x) = \text{res}(e)\text{res}(a_1)\ldots\text{res}(a_{n-1}).$$

For $j=0,1,\ldots,n-1$, let

$$Z_j = \{\text{seq}(x) \mid \text{rp}(x) = q_j\}.$$

Also, let

$$S_j = \{q_i \mid \exists\, a_\ell \ni \lambda(q_i, a_\ell) = q_j\}.$$

Then the following system of equations holds:

$$Z_0 = \text{seq}(e) \vee \left(\bigcup_{i \in S_0} Z_i\right) \delta(q_0)$$

$$Z_j = \left(\bigcup_{i \in S_j} Z_i\right) \delta(q_j), \quad j=1,2,\ldots,n-1.$$

Since this system of equations satisfies the conditions of Theorem 2, each of the events Z_j is regular. Therefore,

$$\bigcup_{j=0}^{n-1} Z_j = \{\text{seq}(x) \mid x \in A_I^*\}$$

is regular. Thus, we have proved the following result.

Corollary 3

The set of all output sequences that a given finite-state machine can produce is a regular set.

Using Corollary 3, we shall now obtain a stronger result. Let $\mathcal{M}$ be the machine introduced above, and let $P \subset A_I^*$ be the event represented by the finite automaton

$$\bar{Q} = (A_I, \bar{Q}, \bar{q}_0, \bar{F}, \lambda).$$

Let $\mathcal{N}$ be the finite-state machine

$$\mathcal{N} = (A_I, Q \times \bar{Q}, A_O, (q_0,\bar{q}_0), \lambda', \delta'),$$

where

$$\lambda'((q,\bar{q}),a) = (\lambda(q,a), \bar{\lambda}(\bar{q},a)),$$

and

$$\delta'(q,\bar{q}) = \delta(q).$$

Then, for any sequence x,

$$\text{seq}_{\mathcal{N}}(x) = \text{seq}_{\mathcal{M}}(x)$$

and

$$x \in P \iff \text{rp}_{\mathcal{N}}(x) = (q,\bar{q}),$$

where $\bar{q} \in \bar{F}$. Therefore,

$$\{\text{seq}_{\mathcal{M}}(x) \mid x \in P\} = \bigcup_{\{(q,\bar{q}) \mid \bar{q}\in\bar{F}\}} \{\text{seq}_{\mathcal{N}}(x) \mid \text{rp}_{\mathcal{N}}(x) = (q,\bar{q})\}.$$

By the argument given in proving Corollary 3, this set is regular. Therefore, we have proved the following result.

Corollary 4

Let $\mathcal{M}$ be a finite-state machine with input alphabet A_I, and let $P \subset A_I^*$ be a representable event. Then $\{\text{seq}_{\mathcal{M}}(x) \mid x\epsilon P\}$ is a regular event; i.e., the function $\text{seq}_{\mathcal{M}}(x)$ maps representable events onto regular events.

We shall next complete the proof of Theorem 1 by showing that every regular event is representable. Our method of proof will consist of showing that every regular event has only a finite number of derivatives.

It will prove useful to introduce the following operation on events:

$$\Delta(P) = \begin{cases} e \text{ if } e \in P \\ \\ \varphi \text{ if } e \notin P \end{cases}$$

The following identities, valid for all x and y in A*, all a $\in$ A, and all events P and Q over A, are direct consequences of the definitions of the operations $\Delta(P)$, $D_x(P)$, $P \cup Q$, $P \cdot Q$, and P^*.

$$\Delta(P \cup Q) = \Delta(P) \cup \Delta(Q)$$

$$\Delta(P \cdot Q) = \Delta(P) \cdot \Delta(Q)$$

$$\Delta(P^*) = e$$

$$\Delta(D_x(P)) = \begin{cases} e \text{ if } x \in P \\ \\ \varphi \text{ if } x \notin P \end{cases}$$

$$D_e(P) = P$$

$$D_{xy}(P) = D_y(D_x(P))$$

$$D_x(P \cup Q) = D_x(P) \cup D_x(Q)$$

$$D_x(PQ) = D_x(P)Q \cup \bigcup_{\{(w_1,w_2) \mid w_1w_2=x\}} \Delta(D_{w_1}(P))D_{w_2}(Q)$$

for $x \neq e$,

$$D_x(P^*) = \bigcup_{\{(w_1,w_2) \mid w_1w_2=x \text{ and } w_1 \in P^*\}} D_{w_2}(P)P^*$$

$$D_a(P \cup Q) = D_a(P) \cup D_a(Q)$$

$$D_a(PQ) = D_a(P)Q \cup \Delta(P)D_a(Q)$$

$$D_a(P^*) = D_a(P)P^*.$$

Most of these identities follow directly from the definitions. We shall prove the two most difficult ones.

(1) $$D_x(PQ) = D_x(P)Q \cup \bigcup_{\{(w_1,w_2) \mid w_1w_2=x\}} \Delta(D_{w_1}(P))D_{w_2}(Q).$$

PROOF: $$y \in D_x(PQ) \iff xy \in PQ.$$

It is convenient to distinguish the two ways in which xy can be an element of PQ.

i) $y = vw$, where $xv \in P$ and $w \in Q$;

ii) $x = w_1w_2$, where $w_1 \in P$ and $w_2y \in Q$.

The set of words v such that $xv \in P$ is $D_x(P)$. Therefore, the set of words y satisfying (i) is $D_x(P)Q$.

The set of words y such that $w_2y \in Q$ is $D_{w_2}(Q)$; also, $w_1 \in P$ if and only if $$\Delta(D_{w_1}(P)) = e.$$

Therefore, the set of words y satisfying (ii) is

$$\bigcup_{\{(w_1,w_2) \mid w_1w_2=x\}} \Delta(D_{w_1}(P))D_{w_2}(Q).$$

Taking the union of the two sets of words, we obtain (1).

(2) For $x \neq e$,

$$D_x(P^*) = \bigcup_{\{(w_1,w_2) \mid w_1w_2=x \text{ and } w_1 \in P^*\}} D_{w_2}(P)P^*.$$

PROOF: $y \in D_x(P^*) \iff xy \in P^*$

$\iff x=w_1w_2$ and $y=z_1z_2$

where $w_1 \in P^*$, $z_2 \in P^*$, and $w_2z_1 \in P$. Hence $z_1 \in D_{w_2}(P)$, and $z_1z_2 \in D_{w_2}(P)P^*$. Taking the union over all choices of (w_1,w_2) satisfying the required conditions, we obtain (2).

The following two lemmas establish certain properties common to all the regular events over an alphabet A. In both lemmas, the method of proof is to show, for every regular expression R, that the event it denotes has the required property. Each of the proofs is by induction on the number of occurrences of the operators $\cup$, $\cdot$, and * in the expression R; this number i(R) is called the index of R. The induction steps of the proofs make use of the fact that, if

$$i(R) = n > 0,$$

then either

$$R = P \cup Q,$$

or

$$R = P \cdot Q,$$

or

$$R = P^*,$$

where P and Q are regular expressions of index less than n.

Lemma 3

If R is regular, then, for all $x \in A^*$, $D_x(R)$ is a regular event.

PROOF:

Basis.

$$i(R) = 0.$$

Then $R = \varphi,$

$R = e,$

or $R = a_\ell,$

an element of A. Clearly, all the derivatives of these expressions are regular.

Induction step.

Let R be of index n, and take the induction hypothesis that the derivatives of events denoted by expressions of index less than n are regular. Either

$$R = P \cup Q,$$

$$R = P \cdot Q,$$

or $R = P^*.$

Inspecting the formulas for $D_x(P \cup Q)$, $D_x(P \cdot Q)$, and $D_x(P^*)$, we find that, in each case, $D_x(R)$ can be composed from regular expressions by a finite number of $\cdot$, $\cup$, and * operations. Therefore, $D_x(R)$ is regular.

Lemma 4

If R is regular, then R has only a finite number of distinct derivatives.

PROOF:

Basis. The result is obvious if $i(R) = 0$.

Induction step. Let R be of index n, and take the induction hypothesis that events denoted by expressions of index less than n have only a finite number of distinct derivatives.

If $$R = P \cup Q,$$

then, for any x,

$$D_x(R) = D_x(P) \cup D_x(Q).$$

Then, if P has p derivatives, and Q has q derivatives, R has at most pq derivatives.

If $$R = P \cdot Q,$$

then, for any x,

$$D_x(R) = D_x(P)Q \cup \bigcup D_{w_2}(Q),$$

where the union is taken over some of the derivatives of Q. The number of ways of choosing $D_x(P)$ is p, and the number of distinct unions of derivatives of Q is at most 2^q. Therefore, there are at most $p \cdot 2^q$ distinct events having the form required of $D_x(R)$.

If $$R = P^*,$$

then, for $x \neq e$,

$$D_x(R) = \bigcup D_{w_2}(P)P^*,$$

where the union is taken over some of the derivatives of P. The number of events expressible in this form is at most 2^p.

Corollary 5

Every regular event is representable.

PROOF: By Lemma 1, every event with a finite number of distinct derivatives is representable.

This corollary completes the proof of Theorem 1; henceforth, the class of regular events and the class of representable events may be identified with each other.

The states of the minimum-state finite automaton representing a given regular event R are in one-to-one correspondence with the distinct derivatives of R. Let $< D_x(R) >$ denote the state corresponding to $D_x(R)$. Then the transition function of the reduced machine is as follows

$$\lambda(< D_x(R) >, a) = < D_{xa}(R) > .$$

Also, $$< D_x(R) > \in F \iff \Delta(D_x(R)) = e,$$

which means that $x \in R$. It is easy to show that, if $< D_e(R) >$ is the initial state, then, for all x,

$$rp(x) = < D_x(R) > ,$$

so that x is accepted if and only if $x \in R$.

A first step in constructing a finite automaton representing R now suggests itself. Starting at the root of the free tree generated by A, and moving up level by level, label each node n_x with an expression for the event $D_x(R)$. The label for n_{xa} may be obtained from the label for n_x by means of the identity

$$D_{xa}(R) = D_a(D_x(R)).$$

If $D_y(R)$ is found to be equal to a derivative previously computed, do not branch out from the node n_y.

The difficulty with this process is that we have no procedure, as yet,

for testing whether two regular expressions, corresponding to derivatives $D_x(R)$ and $D_y(R)$, denote the same event. Nevertheless, the construction process can be made to work, provided that care is used concerning some computational details. First, we must assume that R is in such a form that no parentheses implied by associativity are omitted. The derivative $D_{xa}(R)$ is computed as $D_a(D_x(R))$, where the operation D_a on expressions is defined recursively as follows:

$$D_a(e) = \varphi,$$

$$D_a(\varphi) = \varphi,$$

$$D_a(b) = \varphi$$

if $b \neq a$,

$$D_a(a) = e,$$

$$D_a(PQ) = \begin{cases} (D_a(P)\cdot Q) \text{ if } \Delta(P) = \varphi \\ ((D_a(P)\cdot Q) \cup D_a(Q)) \text{ if } \Delta(P) = e, \end{cases}$$

$$D_a(P^*) = (D_a(P)\ P^*).$$

Let two expressions be called similar, or of the same similarity class, if each can be converted to the other solely by means of the following identities:

$$R \cup R = R$$

$$P \cup (Q \cup R) = (P \cup Q) \cup R$$

$$P \cup Q = Q \cup P$$

$$\varphi P = P\varphi = \varphi$$

$$eP = Pe = P.$$

It is clear that similar expressions denote the same event, and that it is effectively decidable whether two regular expressions are similar.

Theorem 3

Let R be regular. Then, among all the expressions for the derivatives of R, only a finite number of similarity classes occur.

PROOF: We use the same induction scheme as in Lemmas 3 and 4.

Basis. For $i(R) = 0$, the result is obvious.

Induction step. Let R be of index n, and take the induction hypothesis that the theorem holds for all expressions of index less than n. If

$$R = P \cup Q,$$

then, by induction on $lg(x)$, it follows that each expression $D_x(R)$ takes the form

$$D_x(P) \cup D_x(Q).$$

If

$$R = P \cdot Q,$$

then, by induction on $lg(x)$, each expression $D_x(R)$ is of the same similarity class as

$$(D_x(P) \cdot Q) \cup \bigcup D_{w_2}(Q),$$

where the union is taken over all words w_2 such that x is of the form $w_1 w_2$, where $w_1 \in P$. If

$$R = P^*$$

then, again by induction on $lg(x)$, each expression $D_x(R)$ is

similar to

$$\bigcup D_{w_2}(P)\ P^*$$

where the union is taken over all w_1w_2 such that $x = w_1w_2$ and $w_1 \in P^*$. In all three cases, the induction hypothesis is sufficient to show that the derivatives of R fall into a finite number of similarity classes.

Since the number of similarity classes of derivatives of R is finite, the process of labelling nodes of the free tree generated by A with derivatives must terminate, provided that no branches out of n_y are constructed if $D_y(R)$ is similar to some derivation previously computed. Once this construction process has been completed, an automaton representing the event R can easily be constructed. Let $[D_x(R)]$ denote the similarity class of $D_x'(R)$. The automaton is as follows:

$$\mathcal{A} = (A,\{[D_x(R)],\ x \in A^*\},\ [D_e(R)],\ F,\ \lambda),$$

where $[D_x(R)] \in F$ if and only if

$$\Delta(D_x(R)) = e,$$

and

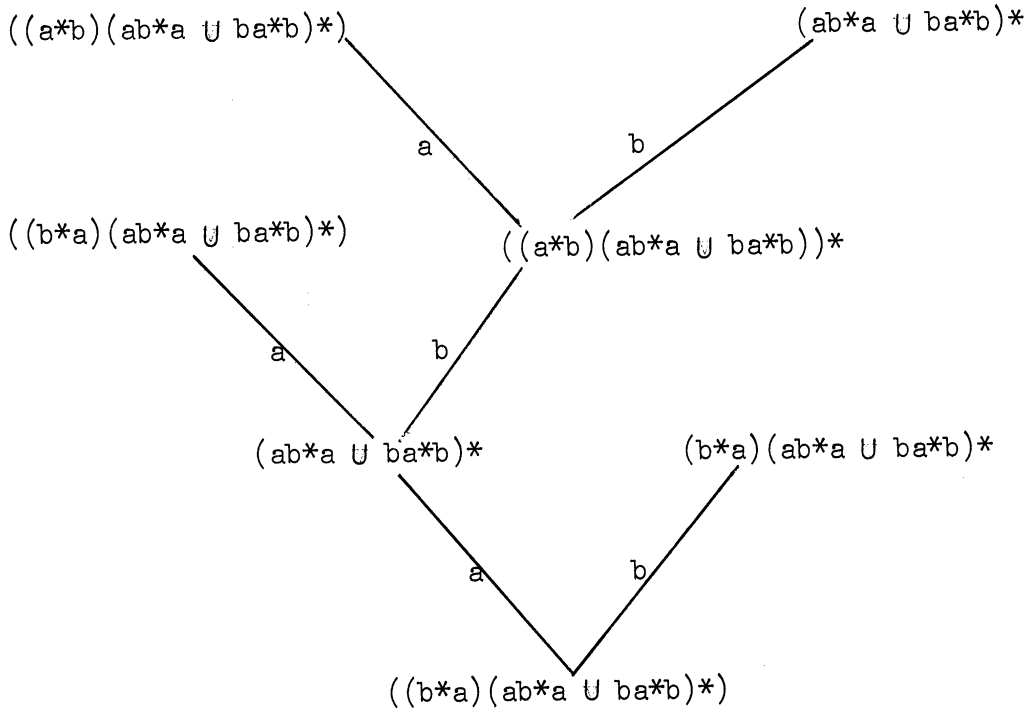
$$\lambda([D_x(R)],\ a) = [D_{xa}(R)].$$

Example 2.

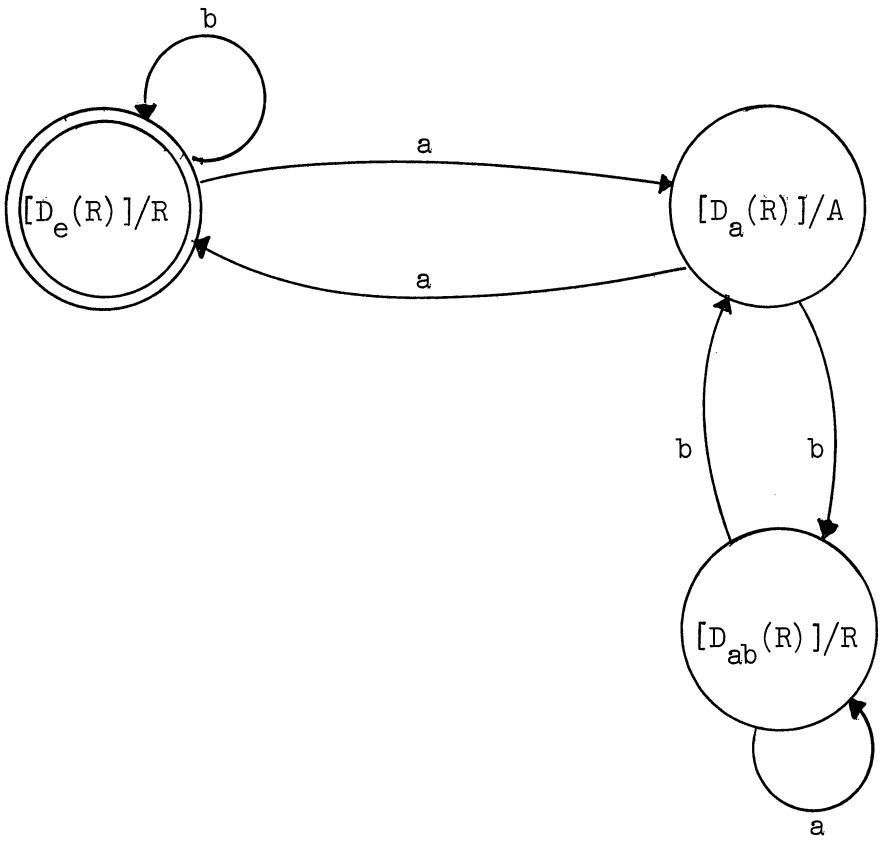
Suppose A = {a,b}, and

$$R = ((ba^*)(ab^*a \vee ba^*b)^*).$$

The labelled tree showing the calculation of derivatives is as follows:



A finite automaton representing R has the following state diagram.



In this particular case the algorithm leads to a reduced machine. This is not necessarily true in general, since the computation process may produce derivatives which denote the same event, but are not similar.

Now that we have a complete characterization of representable events, it is an easy matter to characterize representable functions.

Theorem 4

Let A_I and A_O be finite alphabets. The function

$$f: A_I^* \rightarrow A_O$$

is representable by a finite-state sequential machine if and only if, for each element $b \in A_O$,

$$\{x \mid f(x) = b\}$$

is regular.

PROOF: First we establish the necessity of the given condition. Suppose f is represented by the finite-state machine $\mathcal{M}$, with the set of states

$$Q = \{q_0, q_1, \ldots, q_{n-1}\}.$$

Then, by an argument previously given, for each j, the set

$$X_j = \{x \mid rp(x) = q_j\}.$$

is regular. Therefore,

$$\{x \mid f(x) = b\} = \bigcup_{\{j \mid \delta(q_j)=b\}} X_j,$$

which is a regular set. To show sufficiency, suppose each set $\{x \mid f(x) = b\}$ is regular. Since A_O is finite, for some sufficiently

large integer ℓ there is a one-to-one mapping g from A_0 onto a subset of $\{0,1\}^{\ell}$;

$$b \xrightarrow{g} (\alpha_1(b), \alpha_2(b), \ldots, \alpha_\ell(b)).$$

Then, for $1 \leq k \leq \ell$,

$$\{x \mid \alpha_k(f(x)) = 1\} = \bigcup_{\{b \mid \alpha_k(b)=1\}} \{x \mid f(x) = b\}.$$

This set is therefore a regular event, and by Theorem 1, there exists a finite automaton $\mathcal{A}^{(k)}$ such that

$$\text{res}\, \mathcal{A}_k(x) = \alpha_k(f(x)).$$

Now, consider the finite-state direct product machine

$$\mathcal{P} = \mathcal{A}^{(1)} \times \mathcal{A}^{(2)} \times \ldots \times \mathcal{A}^{(\ell)}.$$

Then, for all x,

$$\text{res}\, \mathcal{P}(x) = g(f(x)).$$

Replacing each output symbol s of an accessible state of $\mathcal{P}$ by the symbol $g^{-1}(s) \in A_0$, one obtains a finite-state machine $\mathcal{P}'$ representing f.

10. PAIR ALGEBRAS AND STATE ASSIGNMENT

Let

$$\bar{\mathcal{M}} = (A_I, \bar{Q}, A_O, \bar{q}_O, \bar{\lambda}, \bar{\delta})$$

be a reduced machine representing the function

$$f: A_I^* \to A_O.$$

Then if

$$\mathcal{M} = (A_I, Q, A_O, q_O, \lambda, \delta)$$

is a machine with all its states accessible which also represents f,

$$\Delta_{\mathcal{M}} = \Delta_{\bar{\mathcal{M}}},$$

and, because $\bar{\mathcal{M}}$ is reduced,

$$\perp_{\mathcal{M}} \leq \perp_{\bar{\mathcal{M}}}.$$

Therefore, there is a strong homomorphism from $\mathcal{M}$ onto $\bar{\mathcal{M}}$. Let the functions associated with this homomorphism be

$$h: Q \to \bar{Q}$$

and

$$\varphi: A_O \to A_O.$$

Then we have the identity

$$\text{res}_{\bar{\mathcal{M}}}(x) = \varphi(\text{res}_{\mathcal{M}}(x);$$

and, since $\mathcal{M}$ and $\bar{\mathcal{M}}$ both represent the same function, φ must be the identity function. Therefore, the function h satisfies the following equations:

i) $h(q_O) = \bar{q}_O$

ii) $h(\lambda(q,a)) = \bar{\lambda}(h(q),a)$

iii) $\bar{\delta}(h(q)) = \delta(q)$.

The existence of a function h satisfying these conditions is a necessary and sufficient condition for $\mathcal{M}$ to represent the same function as $\bar{\mathcal{M}}$. When such a function exists, we say that $\mathcal{M}$ realizes $\bar{\mathcal{M}}$. Although the motivation for this concept depends on the fact that $\bar{\mathcal{M}}$ is reduced, the definition makes sense whether $\bar{\mathcal{M}}$ is reduced or not.

The concept of "realization" is important in connection with the logical design of synchronous sequential circuits. A design specification for such a circuit may be given by a representable function

$$f: \quad A_I^* \rightarrow A_o.$$

Of course, since the input and output alphabets of sequential circuits are given as binary tuples, the set A_I in this case will be of the form $\{0,1\}^{(p)}$, and A_o will be of the form $\{0,1\}^{(r)}$. Let f be such a design specification, and let $\bar{\mathcal{M}}$ be a reduced machine representing the function f. Then any sequential circuit satisfying the design specification can be described by a machine $\mathcal{M}$ which realizes $\bar{\mathcal{M}}$. Moreover, since each internal state of a circuit is represented by a binary s-tuple, it follows that Q, the set of (accessible) states of $\mathcal{M}$, is a subset of $\{0,1\}^{(s)}$, for some s. The problem of specifying $\mathcal{M}$, which involves giving the "coding" of its states as binary s-tuples, so that the resulting sequential circuit will have a simple structure is called the state assignment problem.

In this section we shall discuss the algebraic approach that Hartmanis and Stearns have taken to the state assignment problem [11,12]. A finite-state machine

$$\mathcal{M} = (A_I, Q, A_o, q_0, \lambda, \delta)$$

is state-assigned if

$$Q \subseteq R_1 \times R_2 \times \ldots \times R_s,$$

where $s > 1$, and each of the sets R_j has at least two elements. Then any state $q \in Q$ is an s-tuple $(r_1(q), \ldots, r_s(q))$. The transition function

$$\lambda\colon Q \times A_I \to Q$$

induces partial functions (undefined for some domain elements)

$$\lambda_j\colon R_1 \times R_2 \times \ldots \times R_R \times A_I \to R_j, \quad j=1,2,\ldots,s,$$

where

$$\lambda_j(r_1(q), r_2(q), \ldots, r_s(q), a) = r_j(\lambda(q,a)).$$

A state-assigned machine $\mathcal{M}$ is said to exhibit "reduced state dependence" if the values of some of the functions λ_j can be determined from the input a, together with only a proper subset of the set of arguments $\{r_1, r_2, \ldots, r_s\}$. In the case where $\mathcal{M}$ is a state-assigned realization of $\bar{\mathcal{M}}$, and each set

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

the existence of reduced state dependence means that, in a sequential circuit corresponding to $\mathcal{M}$, some of the "next state" outputs of the combinational part (inputs to latches) can be determined independently of some of the "present-state" inputs (outputs of latches). When a state assignment is made which exploits the possibilities for reduced dependence, the complexity of the resulting combinational circuit is usually considerably less than it would otherwise be.

The concept of reduced dependence in state assignments is closely allied with fundamental ideas about number representations. Thus, for

any integer $b \geq 2$, the base b number representation is a "state assignment" of the integers which allows reduced dependence in the formation of sums and products. For example, the k least significant bits of a sum can be determined from the k least significant bits of the two addends.

The following example shows a connection between the concept of reduced dependence and another kind of number representation, the residue number system, whose application in computers has been studied by Garner, Svoboda, and others.

Example 1.

Consider a sequential machine $\mathcal{M}$ for which

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

$$\bar{Q} = A_o = \{\bar{q}_0, \bar{q}_1, \ldots, \bar{q}_{209}\},$$

and, for all j,

$$\lambda(\bar{q}_j, a) = \bar{q}_{j+1 (\mathrm{mod}\ 210)},$$

and,

$$\delta(\bar{q}_j) = \bar{q}_j.$$

Then the following state assignment is possible:

$$\bar{q}_j = (r_1(\bar{q}_j), r_2(\bar{q}_j), r_3(\bar{q}_j), r_4(\bar{q}_j)),$$

where the four coordinates are the residues of j modulo 2, 3, 5, and 7, respectively. With this assignment, the state transitions of the machine can be determined separately in each coordinate:

$$\lambda_1(r_1, r_2, r_3, r_4, a) = r_1 + 1 \pmod 2$$

$$\lambda_2(r_1, r_2, r_3, r_4, a) = r_2 + 1 \pmod 3$$

$$\lambda_3(r_1, r_2, r_3, r_4, a) = r_3 + 1 \pmod 5$$

$$\lambda_4(r_1, r_2, r_3, r_4, a) = r_4 + 1 \pmod 7$$

We shall investigate the properties of a machine $\bar{\mathcal{M}}$ which govern the existence of state-assigned realizations $\mathcal{M}$ with reduced dependence. Given such a realization $\mathcal{M}$, with

$$Q \subseteq R_1 \times R_2 \times \ldots \times R_s,$$

each subset

$$S \subseteq \{1,2,\ldots,s\}$$

determines an equivalence relation ρ_S over Q as follows:

$$q_1 \rho_S \; q_2 \iff \text{for all } j \in S,\ r_j(q_1) = r_j(q_2).$$

In what follows, it will be convenient to speak interchangeably of equivalence relations over a set Q and of the partitions of Q induced by such equivalence relations. An ordered pair (σ,τ) of equivalence relations over Q is called a <u>partition pair</u> if the following implication holds:

$$q_1 \; \sigma \; q_2 \implies \forall a, \quad (q_1,a) \; \tau \; (q_2,a).$$

Note that (σ,σ) is a partition pair if and only if σ is an equivalence relation with S.P. In terms of partition pairs, the following characterization of reduced state dependence may be given.

<u>Theorem 1</u>

Let S and T be subsets of $\{1,2,\ldots,s\}$. Then the following conditions are equivalent:

1) for all $k \in T$, the value of the function λ_k is determined completely by the arguments a and $\{r_j \mid j \in S\}$;
2) (ρ_S, ρ_T) is a partition pair.

PROOF: Condition (1) means that whenever, for all $j \in S$,

$$r_j(q_1) = r_j(q_2),$$

$$\lambda_k(q_1,a) = \lambda_k(q_2,a)$$

for all $k \in T$. This is equivalent to the statement that, if $q_1 \rho_S q_2$, then, for all a,

$$\lambda(q_1,a) \rho_T (q_2,a);$$

and this means precisely that (ρ_S,ρ_T) is a partition pair. This completes the proof.

Now we have seen that the existence of partition pairs (ρ_S,ρ_T) determines the existence of reduced state dependence in a state-assigned realization $\mathcal{M}$ of $\bar{\mathcal{M}}$. Let us next consider the properties of $\bar{\mathcal{M}}$ which determine the existence of reduced dependence in its realizations. Suppose that $\mathcal{M}$ realizes $\bar{\mathcal{M}}$, and that (σ,τ) is a partition pair of $\mathcal{M}$. Let the equivalence classes associated with σ be $B_1,B_2,\ldots,B_t$, and let the equivalence classes associated with τ be $C_1,C_2,\ldots,C_u$. Then, for any $j \in \{1,2,\ldots,t\}$, and for any $a \in A_I$,

$$\lambda(B_j,a) = \{\lambda(q,a) \mid q \in B_j\}$$

is contained in some set $C_{k(j,a)}$. Let h be the homomorphism from $\mathcal{M}$ onto $\bar{\mathcal{M}}$, and let

$$h(B_j) = \{h(q) \mid q \in B_j\}.$$

Then, for any $j \in \{1,2,\ldots,t\}$,

$$\bar{\lambda}(h(B_j),a) = \{\bar{\lambda}(\bar{q},a) \mid \bar{q} \in h(B_j)\}$$

$$= \{\bar{\lambda}(h(q), a) \mid q \in B_j\}.$$

Because h is a homomorphism, this set is equal to $\{h(\lambda(q,a)) \mid q \in B_j\}$,

which is contained in $h(C_{k(j,a)})$. Thus, the collections $\{h(B_j)\}$ and $\{h(C_k)\}$ have the property that the image of each set $h(B_j)$ under an input is contained in some set $h(C_k)$. Also, since h is onto

$$\bigcup_j h(B_j) = \bigcup_k h(C_k) = \bar{Q}.$$

In the particular case where h is one-to-one, so that $\mathcal{M}$ and $\bar{\mathcal{M}}$ are isomorphic, each of the collections $\{h(B_j)\}$ and $\{h(C_k)\}$ is a partition of $\bar{Q}$.

The foregoing discussion suggests that the definition of partition pair be extended to "overlapping partitions", as follows. A collection $\{D_j,\ j=1,2,\ldots,t\}$ of subsets of $\bar{Q}$ is a <u>cover</u> of $\bar{Q}$ if

$$\bigcup_j D_j = \bar{Q}.$$

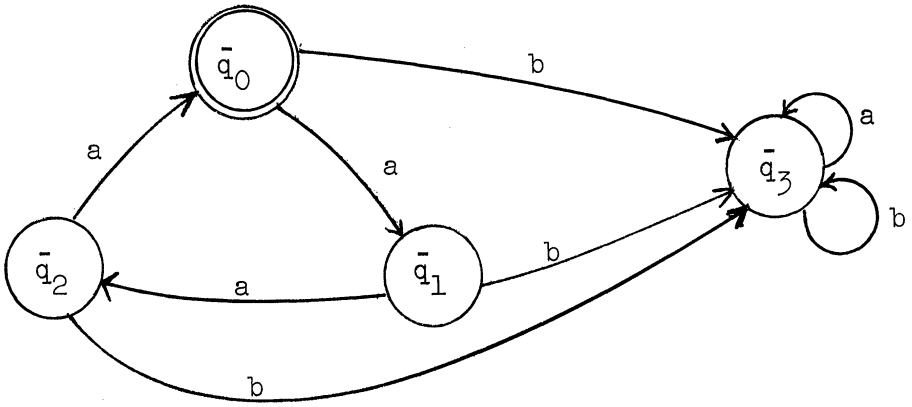
The ordered pair of covers $(\{D_j\}, \{E_k\})$ is a <u>cover pair</u> (with respect to $\bar{\mathcal{M}}$) if, for all j, and for all $a \in A_I$,

$$\bar{\lambda}(D_j,a) = \{\bar{\lambda}(\bar{q},a) \mid \bar{q} \in D_j\}$$

is a subset of some set C_k. A cover φ is said to have the <u>substitution property</u> (S.P.) if (φ,φ) is a cover pair.

The following example indicates the use of covers with S.P. in the construction of state assignments.

Example 2.



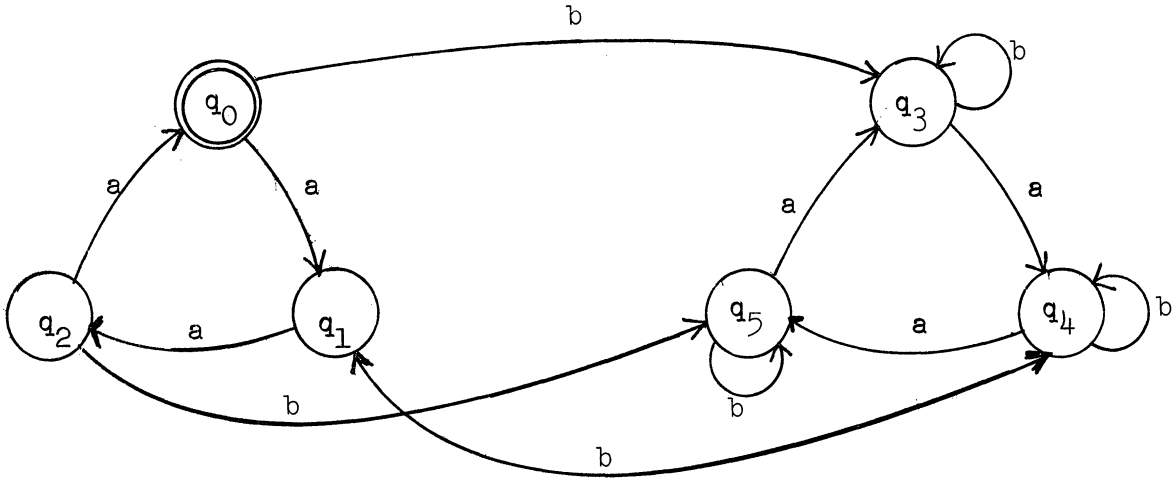
This machine $\bar{\mathcal{M}}$ has the following two covers with S.P.:

$$\varphi\colon \overline{012}\ \overline{3}$$

and

$$\theta\colon \overline{03}\ \overline{13}\ \overline{23}.$$

The first of these is a partition with S.P., but the second is not a partition. By making three "copies" of state 3, we can obtain a machine $\mathcal{M}$ which realizes $\bar{\mathcal{M}}$, and has partitions with S.P. φ' and θ', such that the homomorphism from $\mathcal{M}$ onto $\bar{\mathcal{M}}'$ maps the blocks of φ' onto the blocks of φ, and the blocks of θ' onto the blocks of θ.



$\varphi' = \overline{012}\ \overline{345}$ $\qquad$ $\theta' = \overline{03}\ \overline{14}\ \overline{25}$

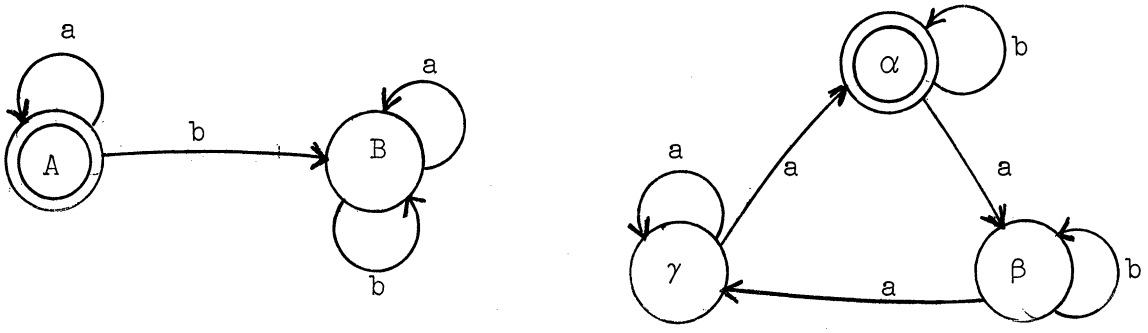
The homomorphism h is as follows:

$$\begin{aligned} h(q_0) &= \bar{q}_0, \\ h(q_1) &= \bar{q}_1, \\ h(q_2) &= \bar{q}_2, \\ h(q_3) &= h(q_4) \\ &= h(q_5) \\ &= \bar{q}_3. \end{aligned}$$

We can now construct a state assignment for $\mathcal{M}$ in which φ' is the partition associated with the equivalence relation $\rho_{\{1\}}$, and θ' is associated with $\rho_{\{2\}}$:

q	$r_1(q)$	$r_2(q)$
q_0	A	α
q_1	A	β
q_2	A	γ
q_3	B	α
q_4	B	β
q_5	B	γ

Because $\rho\{1\}$ and $\rho\{2\}$ are equivalence relations with S.P., reduced dependence exists, and as far as its state transitions are concerned, $\mathcal{M}$ can be represented as a pair of machines operating in parallel, one keeping track of the coordinate r_1, and the other keeping track of r_2, as follows.



We next present a theorem which shows how the existence of partition pairs in the realizations of $\bar{\mathcal{M}}$ is dictated by the cover pairs for $\bar{\mathcal{M}}$. For any cover φ, and any state $\bar{q}$, let $n_\varphi(\bar{q})$ denote the number of blocks of φ in which $\bar{q}$ occurs.

Theorem 2

Let

$$(\varphi,\theta) = (\{D_j\}, \{E_k\})$$

be a cover pair for $\bar{\mathcal{M}}$. Then there exists a realization $\mathcal{M}$ of $\bar{\mathcal{M}}$ with $\Sigma_i \max[n_\varphi(\bar{q}_i), n_\theta(\bar{q}_i)]$ states, having a partition pair $(\{A_j\}, \{B_k\})$ such that, for all j, $h(A_j) = D_j$ and, for all k, $h(B_k) = E_k$.

PROOF: For each state $\bar{q} \in \bar{Q}$, $\mathcal{M}$ will have $\max(n_\varphi(\bar{q}), n_\theta(\bar{q}))$ states whose image under the homomorphism h is $\bar{q}$. Call these states $\bar{q}_A$, $\bar{q}_B$, ... Then enough "copies" of each state $\bar{q}$ are available to permit the construction of partitions $\{A_j\}$ and $\{B_k\}$ of Q, the set of states of $\mathcal{M}$, such that, for all j,

$$h(A_j) = D_j$$

and, for all k,

$$h(B_k) = E_k.$$

For each pair (j,a) choose $k(j,a)$ such that

$$\lambda(D_j,a) \subset E_{k(j,a)}.$$

The transition function of $\mathcal{M}$ is now constructed as follows: if $\bar{q}_P \in A_j$, then $\bar{q} \in D_j$,

$$\lambda(\bar{q},a) \in E_{k(j,a)}$$

and, for some R,

$$[\lambda(\bar{q},a)]_R \in B_{k(j,a)},$$

then set

$$\lambda(\bar{q}_P,a) = [\bar{\lambda}(\bar{q},a)]_R.$$

Then

$$\lambda(A_j,a) \subset B_{k(j,a)},$$

so that $(\{A_j\}, \{B_k\})$ is a partition pair. Also, taking $(\bar{q}_0)_A$ as the initial state of $\mathcal{M}$, and setting

$$\delta(\bar{q}_P) = \delta(\bar{q})$$

for all states $\bar{q}_P$ of M, we find that all the conditions for a homomorphism are met, so that $\mathcal{M}$ realizes $\bar{\mathcal{M}}$.

Example 3.

Let $\bar{\lambda}$ be given by the following table:

$\bar{\lambda}$		A_I	
		a	b
$\bar{Q}$	1	1	4
	2	4	2
	3	4	3
	4	3	3

Then the ordered pair (φ,θ) = $(\{\overline{123},\ \overline{34},\ \overline{234}\},\ \{\overline{14},\ \overline{234}\})$ is a cover pair. Performing the construction given in the proof of Theorem 2, the following realization $\mathcal{M}$ of $\overline{\mathcal{M}}$ is obtained.

λ		A_I	
		a	b
Q	1_A	1_A	4_B
	2_A	4_A	2_A
	3_A	4_A	3_A
	3_B	4_B	3_A
	4_A	3_B	3_A
	2_B	4_B	2_A
	3_C	4_B	3_A
	4_B	3_C	3_A

The following is a partition pair (σ,τ) for $\mathcal{M}$:

$(\{\overline{1_A\ 2_A\ 3_A},\ \overline{3_B\ 4_A},\ \overline{2_B\ 3_C\ 4_B}\},\ \{\overline{1_A\ 4_A},\ \overline{2_A\ 2_B\ 3_A\ 3_B\ 3_C\ 4_B}\})$.

The homomorphism from $\mathcal{M}$ onto $\overline{\mathcal{M}}$ is as follows:

$1_A \to 1$, $2_A \to 2$, $2_B \to 2$, $3_A \to 3$, $3_B \to 3$, $3_C \to 3$, $4_A \to 4$, $4_B \to 4$. The images of the blocks of σ are the blocks of φ, and the images of the blocks of τ are the blocks of θ.

Since the cover pairs of $\bar{\mathcal{M}}$ determine its state-assigned realizations with reduced dependence, it is of interest to consider the algebraic structure of the set of all cover pairs of $\bar{\mathcal{M}}$. We shall also be interested in the structure of the set of partition pairs of $\bar{\mathcal{M}}$, which determine the existence of reduced dependence in state-assigned machines isomorphic to $\bar{\mathcal{M}}$.

Let $\{D_j\}$ and $\{E_k\}$ be covers, and let $\{F_\ell\}$ and $\{G_m\}$ be collections of subsets of $\bar{Q}$ such that each of the sets F_ℓ is a subset of some set D_j, and each of the sets G_m is contained in a set E_k. Then it is easy to check that either all of the following are cover pairs, or none are: $(\{D_j\}, \{E_k\})$, $(\{D_j\}, \{E_k\} \cup \{G_m\})$, $(\{D_j\} \cup \{F_\ell\}, \{E_k\})$, and $(\{D_j\} \cup \{F_\ell\}, \{E_k\} \cup \{G_m\})$. Thus, in characterizing the cover pairs of $\bar{\mathcal{M}}$, we may as well consider only those covers in which no set is included in another. Accordingly, we are led to the following definitions. A <u>set system</u> is a collection $\{D_1, D_2, \ldots, D_t\}$ of subsets of $\bar{Q}$ such that

$$\bigcup_{j=1}^{t} D_j = \bar{Q},$$

and, if

$$D_{j_1} \subset D_{j_2},$$

then $j_1 = j_2$. If $\{S_i\}$ is any collection of subsets of $\bar{Q}$, let Max $(\{S_i\})$ denote the collection consisting of those sets in $\{S_i\}$ that are not properly contained in any other set in $\{S_i\}$. Then, if $\{D_j\}$ is a cover, Max $(\{D_j\})$ is a set system. A <u>systems pair</u> is a cover pair in which both covers are set systems.

The set systems associated with $\bar{Q}$ may be partially ordered as follows: $\varphi_1 \leq \varphi_2$ if every block of φ_1 is contained in a block of φ_2.

Lemma 1

The set systems associated with $\bar{Q}$ form a distributive lattice.

PROOF: The lattice operations are defined as follows:

$$\text{glb}(\varphi_1,\varphi_2) = \varphi_1 \cdot \varphi_2 = \text{Max } \{B \cap B' \mid B \in \varphi_1 \text{ and } B' \in \varphi_2\}$$

$$\text{lub}(\varphi_1,\varphi_2) = \varphi_1 + \varphi_2 = \text{Max } \{B \mid B \in \varphi_1 \text{ or } B \in \varphi_2\}.$$

We omit the simple verification that these operations define the greatest lower bound and least upper bound of φ_1 and φ_2, and that the distributive laws hold:

$$\varphi_1 \cdot (\varphi_2 + \varphi_3) = (\varphi_1 \cdot \varphi_2) + (\varphi_1 \cdot \varphi_3)$$

$$\varphi_1 + (\varphi_2 \cdot \varphi_3) = (\varphi_1 + \varphi_2) \cdot (\varphi_1 + \varphi_3).$$

The universal lower bound 0 for the finite lattice of set systems has each element of $\bar{Q}$ in a block by itself, and the universal upper bound $I = \{\bar{Q}\}$.

Lemma 2

If (φ_1, φ_1') and (φ_2, φ_2') are systems pairs, then $(\varphi_1 \cdot \varphi_2, \varphi_1' \cdot \varphi_2')$ and $(\varphi_1 + \varphi_2, \varphi_1' + \varphi_2')$ are systems pairs.

PROOF: Any element of $\varphi_1 \cdot \varphi_2$ is of the form $B \cap C$, where B is a set contained in φ_1, and C is contained in φ_2. Since (φ_1, φ_1') and (φ_2, φ_2') are systems pairs, it follows that, for any $a \in A_I$, there exist sets $B' \in \varphi_1'$ and $C' \in \varphi_2'$ such that

$$\bar{\lambda}(B,a) \subset B',$$

and

$$\bar{\lambda}(C,a) \subset C'.$$

Therefore,

$$\bar{\lambda}(B \cap C,\ a) \subset B' \cap C',$$

and, since $B' \cap C'$ is contained in some block of $\varphi_1' \cdot \varphi_2'$, the conditions for $(\varphi_1 \cdot \varphi_2,\ \varphi_1' \cdot \varphi_2')$ to be a systems pair are satisfied.

Any element $B \in \varphi_1 + \varphi_2$ is either an element of φ_1 or an element of φ_2; then

$$\bar{\lambda}(B,a) \subset B',$$

where B' is either an element of φ_1' or an element of φ_2'. Accordingly, B' is a subset of some block of $\varphi_1' + \varphi_2'$, and $\bar{\lambda}(B,a)$ is contained in this block, so that the conditions for $(\varphi_1 + \varphi_2,\ \varphi_1' + \varphi_2')$ to be a systems pair are satisfied.

Lemma 3

For any set systems φ and φ', (φ, I) and $(0, \varphi')$ are systems pairs.

Hartmanis and Stearns have introduced an abstract algebraic structure which captures the essential properties of the set of all systems pairs. Let L_1 and L_2 be finite lattices, and let Δ be a subset of $L_1 \times L_2$. Then Δ is a pair algebra on $L_1 \times L_2$ if

(a) $(x_1,y_1) \in \Delta$ and $(x_2,y_2) \in \Delta$

$$\Rightarrow (x_1 \cdot x_2\ \ y_1 \cdot y_2) \in \Delta$$

and $(x_1 + x_2,\ y_1 + y_2) \in \Delta$.

(b) If I is the universal upper bound of L_2 and 0 is the universal lower bound of L_1, then for any $x \in L_1$, $(x, I) \in \Delta$, and for any $y \in L_2$, $(0,y) \in \Delta$.

Theorem 3

Let L be the lattice of set systems of $\bar{Q}$, and let

$$\Delta \subset L \times L$$

be the collection of all systems pairs for $\mathcal{M}$.

Then Δ is a pair algebra.

The proof of this result is a direct consequence of Lemmas 2 and 3.

Before going on to study the properties of pair algebras, let us establish that the set of partition pairs associated with a machine $\mathcal{M}$ is a pair algebra. Let σ and σ' be partitions of Q. Then $\sigma \leq \sigma'$ if and only if every block of σ is contained in a block of σ'.

Lemma 4

The partitions of Q form a lattice.

PROOF: The lattice operations are defined as follows. The blocks of

$$\text{glb}(\sigma_1,\sigma_2) = \sigma_1 \cdot \sigma_2$$

are the set intersections of the blocks of σ_1 with the blocks of σ_2. The states q_1 and q_2 are in the same block of

$$\text{lub}(\sigma_1,\sigma_2) = \sigma_1 + \sigma_2$$

if and only if there is a sequence $z_1,z_2,\ldots,z_n$ of elements of Q such that $z_1=q_1$, $z_n=q_2$, and for i=1,2,...,n-1, z_i and z_{i+1} are either in

the same block of σ_1 or in the same block of σ_2. The verification that the operations just defined yield the greatest lower bound and least upper bound of σ_1 and σ_2 is essentially the same as the corresponding step in the proof that the right congruences over A_I^* form a lattice (Theorem 5-3).

The following lemmas will establish that the partition pairs of $\mathcal{M}$ form a pair algebra.

Lemma 5

If (σ_1,τ_1) and (σ_2,τ_2) are partition pairs, then $(\sigma_1 \cdot \sigma_2, \tau_1 \cdot \tau_2)$ and $(\sigma_1 + \sigma_2, \tau_1 + \tau_2)$ are partition pairs.

PROOF: Every block of $\sigma_1 \cdot \sigma_2$ is of the form $B \cap C$, where B is a block of σ_1 and C is a block of σ_2. Then

$$\lambda(B,a) \subseteq B',$$

and

$$\lambda(C,a) \subseteq C',$$

where B' is a block of τ_1, and C' is a block of τ_2; thus

$$\lambda(B \cap C, a) \subseteq B' \cap C',$$

and

$$B' \cap C'$$

is a block of $\tau_1 \cdot \tau_2$. This completes the proof that $(\sigma_1 \cdot \sigma_2, \tau_1 \cdot \tau_2)$ is a partition pair.

If q_1 and q_2 are in the same block of $\sigma_1 + \sigma_2$, then there is a

sequence $z_1, z_2, \ldots, z_n$ such that

$$z_1 = q_1,$$

$$z_n = q_2,$$

and for i=1,2,...,n-1, z_i and z_{i+1} are either in the same block of σ_1 or in the same block of σ_2. For any given input symbol a, consider the sequence $(\lambda(z_1,a), \lambda(z_2,a), \ldots, \lambda(z_n,a))$. If $z_i \ \sigma_1 \ z_{i+1}$, then

$$\lambda(z_i,a) \ \tau_1 \ \lambda(z_{i+1},a),$$

and if $z_i \ \sigma_2 \ z_{i+1}$, then

$$\lambda(z_i,a) \ \tau_2 \ \lambda(z_{i+1},a),$$

since (σ_1,τ_1) and (σ_2,τ_2) are partition pairs. Since

$$z_i = q_1$$

and

$$q_n = q_2,$$

we have proved that, if q_1 and q_2 are in the same block of $\sigma_1 + \sigma_2$, then for any a, $\lambda(q_1,a)$ and $\lambda(q_2,a)$ are in the same block of $\tau_1 + \tau_2$. This completes the proof.

The universal upper bound I of the lattice of partitions of Q has a single block containing all the elements of Q, and the universal lower bound 0 has each state in a block by itself.

Lemma 6

Let σ and τ be partitions of φ. Then (σ, I) and $(0, \tau)$ are partition pairs.

Combining Lemmas 5 and 6, we may state the following theorem.

Theorem 4

The partition pairs associated with a machine $\mathcal{M}$ form a pair algebra.

We shall next derive some properties of pair algebras in general. Once this has been done, we shall consider how some of the results apply to the class of systems pairs and the class of partition pairs. In what follows we assume that Δ is a pair algebra on $L_1 \asymp L_2$.

Lemma 7

If $(x,y) \in \Delta$ and $x' \leq x$, then $(x',y) \in \Delta$;

if $(x,y) \in \Delta$ and $y \leq y'$, then $(x,y') \in \Delta$.

PROOF: Since $(x,y) \in \Delta$ and $(x',I) \in \Delta$,

$$(x \circ x',\ y \circ I) = (x',y) \in \Delta.$$

Since $(x,y) \in \Delta$ and $(0,y') \in \Delta$,

$$(x + 0,\ y + y') = (x,y') \in \Delta.$$

For any set S of lattice elements, let ΠS denote the greatest lower bound of S, and ΣS, the least upper bound of S. For $x \in L_1$, let $m(x)$ be defined as

$$\Pi\{y_i \mid (x,y_i) \in \Delta\},$$

and for $y \in L_2$, let $M(y)$ be defined as

$$\Sigma\{x_i \mid (x_i,y) \in \Delta\}.$$

The interesting properties of pair algebras all seem to involve these operators. In deriving these properties, we can cut our work in half by noting a certain duality. If L is a lattice with ordering relation

$\leq$, then a dual lattice L^D is obtained by taking the elements of L with the ordering relation $\geq$ defined as follows:

$$y \geq x \iff x \leq y.$$

Then the $\cdot$ operation in L is the + operation in L^D, and the + operation in L is the $\cdot$ operation in L^D. Universal upper and lower bounds in L, if they exist, are respectively universal lower bounds in L^D. If Δ is a pair algebra on $L_1 \asymp L_2$, then a dual pair algebra Δ^D on

$$L_2^D \asymp L_1^D$$

is obtained as follows:

$$(y,x) \in \Delta^D \iff (x,y) \in \Delta.$$

Let the operations m and M in Δ^D be denoted m^D and M^D, to distinguish them from the corresponding operations in Δ, which we continue to denote as m and M. Then the functions m^D and M are identical, and the functions M^D and m are identical. Thus, any theorem T about the class of all pair algebras applies to both Δ and Δ^D, and when T, as it applies to Δ^D, is translated into a statement about Δ, a dual theorem T^D is obtained. The properties given in Table 1 are listed in dual pairs.

(i)	$(x,m(x)) \in \Delta$	$(M(y),y) \in \Delta$
(ii)	$x_1 \leq x_2 \Longrightarrow m(x_1) \leq m(x_2)$	$y_1 \leq y_2 \Longrightarrow M(y_1) \leq M(y_2)$
(iii)	$m(x_1+x_2) = m(x_1) + m(x_2)$	$M(y_1 \cdot y_2) = M(y_1) \cdot M(y_2)$
(iv)	$m(x_1 \cdot x_2) \leq m(x_1) \cdot m(x_2)$	$M(y_1) + M(y_2) \leq M(y_1 + y_2)$
(v)	$m(x) \leq y \Longleftrightarrow (x,y) \in \Delta$	$x \leq M(y) \Longleftrightarrow (x,y) \in \Delta$
(vi)	$x \leq M(m(x))$	$m[M(y)] \leq y$
(vii)	$m\{M[m(x)]\} = m(x)$	$M\{m[M(x)]\} = M(x)$

TABLE 1

We shall prove that the first of each dual pair of properties holds.

(i) Let $y_1,y_2,\ldots,y_n$ be the elements y_i of L_2 such that $(x,y_i) \in \Delta$.

Since $(x,y_1) \in \Delta$ and $(x,y_2) \in \Delta$,

$$(x \cdot x,\ y_1 \cdot y_2) = (x,\ y_1 \cdot y_2) \in \Delta.$$

Repeating this argument n-1 times,

$$(x, y_1 \cdot y_2 \cdot \ldots \cdot y_n) = (x,m(x)) \in \Delta.$$

(ii) Since $(x_2,\ m(x_2)) \in \Delta$ and $x_1 \leq x_2$,

$$(x_1,\ m(x_2)) \in \Delta;$$

since $m(x_1) \leq y$ whenever $(x_1,y) \in \Delta$,

$$m(x_1) \leq m(x_2).$$

(iii) Since $(x_1,\ m(x_1)) \in \Delta$ and $(x_2,\ m(x_2)) \in \Delta$,

$$(x_1 + x_2,\ m(x_1) + m(x_2)) \in \Delta;$$

therefore, $m(x_1 + x_2) \leq m(x_1) + m(x_2)$.

On the other hand, since $x_1 \leq x_1 + x_2$ and $x_2 \leq x_1 + x_2$,

$m(x_1) \leq m(x_1 + x_2)$ and $m(x_2) \leq m(x_1 + x_2)$;

therefore

$$m(x_1) + m(x_2) \leq m(x_1 + x_2).$$

Combining the two results,

$$m(x_1) + m(x_2) = m(x_1 + x_2).$$

(iv) Since $(x_1, m(x_1)) \in \Delta$ and $(x_2, m(x_2) \in \Delta$,

$$(x_1 \cdot x_2, m(x_1) \cdot m(x_2)) \in \Delta;$$

therefore

$$m(x_1 \cdot x_2) \leq m(x_1) \cdot m(x_2).$$

(v) If $(x,y) \in \Delta$, then $m(x)$ is a lower bound for y. Conversely, if $m(x) \leq y$, then since $(x, m(x)) \in \Delta$, we know by Lemma 7, that $(x,y) \in \Delta$.

(vi) Since $(x, m(x)) \in \Delta$, the result (v), in dual form, ensures that

$$x \leq M(m(x)).$$

(vii) Since $x \leq M(m(x))$,

$$m(x) \leq m(M(m(x)).$$

But, by (vi), in dual form, taking y as $m(x)$,

$$m(M(m(x)) = m(x).$$

It is interesting to note that the functions m and M establish a Galois connection between the lattices L_1 and L_2^D. In general, a Galois connection between two partially ordered sets S and T is a pair of functions

$$\sigma: S \xrightarrow{\text{into}} T$$

and

$$\tau: T \xrightarrow{\text{into}} S$$

such that:

(a) $x_1 \leq x_2 \Rightarrow \sigma(x_2) \leq \sigma(x_1)$ $\qquad (x_1, x_2 \in S)$

(b) $y_1 \leq y_2 \Rightarrow \tau(y_2) \leq \tau(y_1)$ $\qquad (y_1, y_2 \in T)$

(c) $x \leq \tau\ \sigma(x)$ $\qquad (x \in S)$

(d) $y \leq \sigma\ \tau(y)$ $\qquad (y \in T)$

It is evident that, with appropriate allowance for dualization of L_2, properties (a) and (b) correspond to our identity (ii), and (c) and (d) correspond to (vi).

An element $(x,y) \in \Delta$ is called an Mm-pair if

$$y = m(x) \quad \text{and} \quad x = M(y).$$

By virtue of (vii), the following statements hold:

for any $x \in L_1$, $\quad (Mm(x), m(x))$ is an Mm-pair,

and

for any $y \in L_2$, $\quad (M(y), mM(y))$ is an Mm-pair.

As the following useful lemma shows, the Mm-pairs associated with a pair algebra Δ determine the structure of Δ completely.

Lemma 8

The ordered pair (x,y) is an element of Δ if and only if there is an Mm-pair (x',y') such that $x \leq x'$ and $y' \leq y$.

PROOF: The sufficiency of the condition is obvious:

$$(x',y') \in \Delta \Rightarrow (x,y) \in \Delta.$$

To prove necessity simply note that, if $(x,y) \in \Delta$, either $(Mm(x), m(x))$ or $(M(y), mM(y))$ can serve as the required Mm-pair (x',y').

If (x_1,y_1) and (x_2,y_2) are Mm-pairs then, clearly,

$$x_1 \leq x_2 \Rightarrow y_1 \leq y_2.$$

Therefore, the Mm-pairs are partially ordered in a natural way:

$$(x_1,y_1) \leq (x_2,y_2)$$

if and only if $x_1 \leq x_2$.

Lemma 9

The set of Mm-pairs forms a lattice in which

$$\text{glb}((x_1,y_1), (x_2,y_2)) = (x_1 \cdot x_2, m(x_1 \cdot x_2))$$

and

$$\text{lub}((x_1,y_1), (x_2,y_2)) = (M(y_1 + y_2), y_1 + y_2).$$

PROOF: Since

$$x_1 = M(m(x_1))$$

and

$$x_2 = M(m(x_2)),$$

$$x_1 \cdot x_2 = M(m(x_1) \cdot m(x_2)).$$

Since

$$m(x_1 \cdot x_2) \leq m(x_1) \cdot m(x_2),$$

$$M(m(x_1 \cdot x_2)) \leq M(m(x_1) \cdot m(x_2)),$$

so that

$$M(m(x_1 \cdot x_2)) \leq x_1 \cdot x_2.$$

On the other hand, since

$$(x_1 \cdot x_2, m(x_1 \cdot x_2)) \in \Delta,$$

$$x_1 \cdot x_2 \leq M(m(x_1 \cdot x_2)).$$

Therefore

$$x_1 \cdot x_2 = M(m(x_1 \cdot x_2)),$$

and $(x_1 \cdot x_2,\ m(x_1 \cdot x_2))$ is an Mm-pair, and a lower bound for both (x_1,y_1) and (x_2,y_2); it is also the greatest lower bound since

$$(x_3,y_3) \leq (x_1,y_1) \text{ and } (x_3,y_3) \leq (x_2,y_2)$$

implies

$$x_3 \leq x_1 \cdot x_2.$$

By a dual proof, it is verified that

$$\text{lub}((x_1,y_1),\ (x_2,y_2)) = (M(y_1 + y_2),\ y_1 + y_2).$$

Let Δ be a pair algebra on $L \times L$. An element $x \in L$ is <u>self-sufficient</u> (with respect to Δ) if and only if $(x,x) \in \Delta$. Then x is self-sufficient if and only if

$$m(x) \leq x \leq M(x).$$

In particular, the elements 0 and I are self-sufficient.

A nonempty subset R of a lattice L is a <u>sublattice</u> of L if

$$a,b \in R \Rightarrow a \cdot b \in R \text{ and } a + b \in R,$$

where $\cdot$ and $+$ are the meet and join operations for L. Note that it is possible for R to be a lattice, and yet not be a sublattice of L. For example, the partitions of an n-element set S form a lattice, but for $n \geq 3$, the partitions of S do not form a sublattice of the lattice of set systems on S.

<u>Lemma 10</u>

The self-sufficient elements with respect to Δ form a sublattice of L.

PROOF: If x and y are self-sufficient then $(x,x) \in \Delta$ and $(y,y) \in \Delta$. Therefore, $(x\cdot y, x\cdot y) \in \Delta$, and $(x + y, x + y) \in \Delta$, so that $x\cdot y$ and $x + y$ are self-sufficient.

Let us now apply some of the foregoing general results to two particular pair algebras: Π, the pair algebra of partition pairs of a finite-state machine $\mathcal{M}$, and $\equiv$, the pair algebra of systems pairs of $\mathcal{M}$. It is immediate that the self-sufficient elements of Π are the partitions with S.P., and, analogously, that the self-sufficient elements for $\equiv$ are the set systems with S.P.: a set system $\{B_j\}$ has S.P. if, for all j, and for every input symbol a, there is a k such that

$$\bar{\lambda}(B_j,a) \subset B_k.$$

Let us consider how the functions m and M are determined for Π. The equivalence relation $m(\sigma)$ is to be chosen as "fine" as possible, subject to the condition that $(\sigma,m(\sigma))$ must be a partition pair. Thus, we are led to the following two-step specification of $m(\sigma)$:

(i) Let the symmetric, reflexive relation $\mu(\sigma)$ be defined as follows: $\bar{q}_1 \mu(\sigma) \bar{q}_2$ if and only if $\bar{q}_1 = \bar{q}_2$ or there exist $\bar{q}_i$ and $\bar{q}_j \in \bar{Q}$, and $a \in A_I$, such that

$$\bar{q}_i \sigma \bar{q}_j, \quad \bar{\lambda}(\bar{q}_i,a) = \bar{q}_1,$$

and

$$\bar{\lambda}(\bar{q}_j,a) = \bar{q}_2;$$

(ii) $m(\sigma)$ is the transitive closure of $\mu(\sigma)$; i.e. $\bar{q}_1 \, m(\sigma) \, \bar{q}_2$ if and only if there is a sequence of states $z_1,z_2,\ldots,z_n$ such that $\bar{q}_1 = z_1$, $\bar{q}_2 = z_n$, and, for $i=1,2,\ldots,n-1$,

$$z_i \, \mu(\sigma) \, z_{i+1}.$$

It is clear that the equivalence relation specified by this construction is equal to $m(\sigma)$; for two states are equivalent in this relation if and only if the definition of a partition pair, together with the condition of transitivity, implies that these states are equivalent in any equivalence relation τ such that (σ,τ) is a partition pair.

The function $M(\tau)$ associated with the pair algebra Π is also easy to specify:

$$\bar{q}_1 \; M(\tau) \; \bar{q}_2$$

if and only if, for all a,

$$\bar{\lambda}(\bar{q}_1,a) \; \tau \; \bar{\lambda}(\bar{q}_2,a).$$

This equivalence relation is clearly the coarsest of all the equivalence relations σ such that (σ,τ) is a partition pair.

Example 4.

Consider a machine $\mathcal{M}$ with

$$A_I = \{a_1,a_2\},$$
$$\bar{Q} = \{1,2,3,4,5,6,7\},$$

and the transition function $\bar{\lambda}$ specified by the following table.

$\bar{\lambda}$		A_I	
		a_1	a_2
$\bar{Q}$	1	6	5
	2	6	2
	3	3	5
	4	1	2
	5	2	1
	6	4	7
	7	4	7

Let the equivalence relation σ be specified by the following partition: $\{\overline{1}\ \overline{23}\ \overline{4}\ \overline{567}\}$. The images of the blocks of this partition under the inputs a_1 and a_2 form the following set of blocks: $\{\overline{6}\ \overline{36}\ \overline{1}\ \overline{24};$ $\overline{5}\ \overline{25}\ \overline{2}\ \overline{17}\}$. Then

$$\bar{q}_i\ \mu(\sigma)\ \bar{q}_j$$

if and only if $\bar{q}_i$ and $\bar{q}_j$ occur together in one of these blocks. Therefore, $m(\sigma)$, the transitive closure of $\mu(\sigma)$, is given by the following partition: $\{\overline{17},\ \overline{245},\ \overline{36}\}$. Let us now compute $M(m(\sigma))$. Denoting the blocks 17, 245, 36 by A, B, and C, we can form a table which specifies, for each state $\bar{q}$ and input a, the block which includes $\bar{\lambda}(\bar{q},a)$. This table is as follows:

		A_I	
		Q_1	Q_2
$\bar{Q}$	1	C	B
	2	C	B
	3	C	B
	4	A	B
	5	B	A
	6	B	A
	7	B	A

Then two states are equivalent in $M(m(\sigma))$ if and only if they correspond to identical rows in the table. Therefore, $M(m(\sigma))$ is given by the partition $\{\overline{123}\ \overline{4}\ \overline{567}\}$, and the ordered pair $(\{\overline{123}\ \overline{4}\ \overline{567}\},\ \{\overline{17}\ \overline{245}\ \overline{36}\})$ is an Mm-pair.

The function $m(\sigma)$ associated with the pair algebra $\equiv$ is quite easy to specify: given

$$\sigma = \{B_j\},$$

$$m(\sigma) = \text{Max}\ (\{\bar{\lambda}(B_j)\} \cup 0).$$

The determination of $M(\tau)$ is a bit more complicated. Let

$$\tau = \{C_k\},$$

and suppose that

$$A_I = \{a_1, a_2, \ldots, a_m\};$$

for any m-tuple $(k_1, k_2, \ldots, k_m)$ of indices of blocks of τ, let

$$B_{k_1,k_2,\ldots,k_m} = \{\bar{q}_j \mid \forall\ i,\ \bar{\lambda}(\bar{q}_j, a_i) \in C_{k_i}\}$$

Then, if $\{B_{k_1,k_2,\ldots,k_m}\}$ is the collection of all sets obtained in this way, $M(\tau) = \text{Max}\{B_{k_1,k_2,\ldots,k_m}\}$.

Example 5.

We continue to consider the machine introduced in Example 4.

Let

$$\sigma = \{\overline{1234}\ \overline{4567}\}.$$

The images of the blocks of σ under the inputs a_1 and a_2 form the set $\{\overline{126}\ \overline{124}\ \overline{25}\ \overline{127}\}$. Therefore

$$m(\sigma) = \{\overline{136}\ \overline{124}\ \overline{25}\ \overline{127}\}.$$

Let the blocks of $m(\sigma)$ be denoted as follows:

$$\overline{136} = C_1,$$

$$\overline{124} = C_2,$$

$$\overline{25} = C_3,$$

$$\overline{127} = C_4.$$

As a first step in computing $M(m(\sigma))$, we form a table giving, for each state $\bar{q}$ and input symbol a, the indices of all blocks of $m(\sigma)$ which include $\bar{\lambda}(\bar{q}, a)$.

		A_I	
		a_1	a_2
$\bar{Q}$	1	1	3
	2	1	2,3,4
	3	1	3
	4	1,2,4	2,3,4
	5	2,3,4	1,2,4
	6	2	4
	7	2	4

From the table we see that

$$B_{1,3} = \overline{1234},$$

that

$$B_{2,4} = \overline{4567},$$

and that every set B_{k_1,k_2} is contained either in $B_{1,3}$ or in $B_{2,4}$. Therefore

$$M(m(\sigma)) = \{\overline{1234}, \overline{4567}\},$$

and the ordered pair

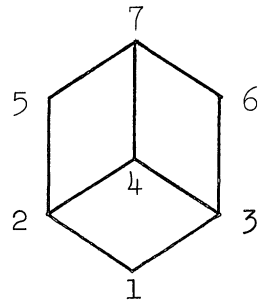
$$(\{\overline{1234}, \overline{4567}\}, \{\overline{136}, \overline{124}, \overline{25}, \overline{127}\})$$

is an Mm-pair in the pair algebra $\equiv$.

The next computational problem we consider is that of determining the lattices of Mm-pairs for the pair algebras π and $\equiv$ associated with a finite-state machine $\bar{m}$. For this purpose we shall need a few elementary lattice-theoretic concepts. An element a of a lattice L is meet-irreducible if a cannot be expressed as b·c, where b and c are different from a. Similarly, a is join-irreducible if a cannot be expressed as b+c, where b and c are different from a.

Example 6.

For the finite lattice shown below, the set of join-irreducible elements is {1,2,3,5,6}, and the set of meet-irreducible elements is {4,5,6,7}.



Lemma 11

Any element of a finite lattice L is a join of join-irreducible elements.

PROOF: Let a be a least element of L which is not a join of join-irreducible elements. Then a is not join-irreducible, so $a = b + c$, where $b \neq a$ and $c \neq a$. Clearly, b and c cannot be joins of join-irreducible elements. But if, for example, b is not such a join, then since $b \leq a$, the minimality of a is contradicted.

Corollary 1

Any element of a finite lattice L is a meet of meet-irreducible elements.

Now, suppose that

$$\Delta \subset L_1 \asymp L_2$$

is a pair algebra. Let the set of join-irreducible elements of L_1 be $\{a_1,\ldots,a_r\}$, and let the set of meet-irreducible elements of L_2

be $\{b_1,\ldots,b_s\}$. Then any element $x \in L_1$ is equal to

$$\sum_{\{a_i \mid a_i \leq x\}} a_i$$

and

$$m(x) = m\Big(\sum_{\{a_i \mid a_i \leq x\}} a_i\Big).$$

But, from identity (ii) of Table 1 it follows that

$$m\Big(\sum_{\{a_i \mid a_i \leq x\}} a_i\Big) = \sum_{\{a_i \mid a_i \leq x\}} m(a_i).$$

Thus, every element of L_2 of the form $m(x)$ is a join of elements of the form $m(a_i)$, where each element a_i is join-irreducible. By duality, any element $y \in L_2$ is equal to

$$\prod_{\{b_j \mid y \leq b_j\}} b_j,$$

and

$$M(y) = \prod_{\{b_j \mid y \leq b_j\}} M(b_j).$$

Two algorithms for computing the Mm-pairs associated with a pair algebra Δ now suggest themselves; the first of these makes use of the join-irreducible elements of L_1. The second algorithm, whose application to Δ is equivalent to the application of the first algorithm to Δ^D, uses the meet-irreducible elements of L_2. An outline of the first

algorithm follows:

(1) Determine the join-irreducible elements $a_1, a_2, \ldots, a_r$ of L_1;

(2) Find $m(a_1)$, $m(a_2)$, ..., $m(a_r)$;

(3) Form all the distinct joins of elements $m(a_i)$;

(4) For each element $\Sigma\, m(a_i)$, form the Mm-pair

$(M(m(\Sigma\, a_i)), m(\Sigma\, a_i))$.

This algorithm yields all the Mm-pairs contained in Δ. By Lemma 8, the Mm-pairs determine Δ completely.

To assess the usefulness of the two dual algorithms in determining the Mm-pairs of the pair algebras π and $\equiv$, let us characterize the join-irreducible and meet-irreducible elements of the lattice of partitions of a finite set $\bar{Q}$ with n elements, and the lattice of set systems associated with $\bar{Q}$; these lattices will be denoted, respectively, $P(\bar{Q})$ and $S(\bar{Q})$.

Let I and 0 denote the universal upper and lower bounds of $P(\bar{Q})$. Also, let σ_{ab} denote the partition with one block equal to $\{a,b\}$, and each other block containing only a single element. Finally, for any proper subset $R \subset \bar{Q}$, let σ_R denote the two-block partition having R as one block, and the complement of R in $\bar{Q}$ as the other. Thus,

$$\sigma_{ab} \neq \sigma_{\{a,b\}}.$$

Lemma 12

An element $\sigma \in P(\bar{Q})$ is join-irreducible if and only if $\sigma = 0$ or $\sigma = \sigma_{ab}$, where $a \in \bar{Q}$ and $b \in \bar{Q}$.

PROOF: 0 is join-irreducible, and the only elements σ such that $\sigma \leq \sigma_{ab}$ are 0 and σ_{ab}, so that σ_{ab} is join-irreducible. On the other hand, any partition σ is equal to

$$\sum_{\{\sigma_{ab} \mid \sigma_{ab} \leq \sigma\}} \sigma_{ab},$$

so that no elements except the specified ones are join-irreducible.

Lemma 13

An element $\sigma \in P(\bar{Q})$ is meet-irreducible if and only if $\sigma = I$, or $\sigma = \sigma_R$, where R is a proper subset of $\bar{Q}$.

PROOF: This proof parallels the previous one. I is meet-irreducible, and the only elements σ such that $\sigma_R \leq \sigma$ are I and σ_R, so that σ_R is meet-irreducible. On the other hand, any partition

$$\sigma = \prod_{\{\sigma_R \mid \sigma \leq \sigma_R\}} \sigma_R,$$

so that no elements except the specified ones are meet-irreducible.

Lemma 14

An element $\varphi \in S(\bar{Q})$ is join-irreducible if and only if φ contains at most one block with more than a single element.

PROOF: The universal lower bound of $S(\tilde{Q})$ is join-irreducible, and the join of any two incomparable elements of $S(\tilde{Q})$ necessarily contains at least two blocks with more than one element. On the other hand, any element φ of $S(\tilde{Q})$ is the join of all those elements θ such that $\theta \leq \varphi$ and θ has only one block with more than one element.

Lemma 15

An element $\varphi \in S(\tilde{Q})$ is meet-irreducible if and only if each block of φ has at least n-1 elements.

PROOF: Let φ contain a block B_j with less than n-1 elements, and let k_1 and k_2 be elements of $\tilde{Q}$ not in B_j. Let φ^1 be obtained from φ by replacing B_j with $B_j \cup \{k_1\}$, and let φ^2 be obtained by replacing B_j with $B_j \cup \{k_2\}$. Then

$$\varphi = \varphi^1 \cdot \varphi^2.$$

Therefore, in order for φ to be meet-irreducible, every block of φ must contain at least n-1 elements. On the other hand, let φ_1 and φ_2 be two incomparable set systems, each having n-1 elements in every block. Let B_{j_1} be a block of φ_1 not contained in φ_2, and let B_{j_2} be a block of φ_2 not in φ_1. Then $B_{j_1} \cap B_{j_2}$ is a block of $\varphi_1 \cdot \varphi_2$, and $B_{j_1} \cap B_{j_2}$ has n-2 elements. Therefore, any set system in which every block has at least n-1 elements is not a meet of incomparable set systems, and every such set system is therefore meet-irreducible. This completes the proof.

Thus, we see that $P(\bar{Q})$ has $\binom{n}{2}+1$ join-irreducible elements, and 2^{n-1} meet-irreducible elements; $S(\bar{Q})$ has 2^n-n join-irreducible elements, and 2^n-n meet-irreducible elements. Thus, it seems practical to obtain the Mm-pairs of the pair algebra π using the join-irreducible elements of $P(\bar{Q})$, but the determination of the Mm-pairs of $\equiv$, using either the join-irreducible or meet-irreducible elements of $S(\bar{Q})$, seems impractical except in the case of machines with a very small number of states.

Example 7.

Consider a machine $\bar{m}$ with

$$\bar{Q} = \{0,1,2,3\},$$

$$A_I = \{a_1,a_2,a_3\},$$

and the following transition function

		A_I		
		a_1	a_2	a_3
$\bar{Q}$	0	3	2	0
	1	3	0	2
	2	2	3	0
	3	2	1	2

$$m(0) = 0$$

$$m(\sigma_{01}) = \overline{02}\ \overline{1}\ \overline{3}$$

$$m(\sigma_{02}) = \overline{23}\ \overline{0}\ \overline{1}$$

$$m(\sigma_{03}) = I$$

$$m(\sigma_{12}) = \overline{023}\ \overline{1}$$

$$m(\sigma_{13}) = \overline{01}\ \overline{23}$$

$$m(\sigma_{23}) = \overline{02}\ \overline{13}$$

No additional partitions are obtained as joins of the given ones. The

Mm-pairs are as follows:

$(0,0)$, $(\overline{01}\ \overline{23},\ \overline{02}\ \overline{1}\ \overline{3})$, $(\overline{02}\ \overline{1}\ \overline{3},\ \overline{23}\ \overline{0}\ \overline{1})$, $(1,\ 1)$, $(\overline{012}\ \overline{3},\ \overline{023}\ \overline{1})$

$(\overline{02}\ \overline{13},\ \overline{01}\ \overline{23})$, and $(\overline{01}\ \overline{23},\ \overline{02}\ \overline{13})$.

A state-assigned machine isomorphic with $\mathcal{M}$, and with the set of states $R_1 \times R_2$, where

$$R_1 = R_2 = \{\alpha, \beta\},$$

can be so constructed that

$$\rho_{\{1\}} = \overline{01}\ \overline{23},$$

and

$$\rho_{\{2\}} = \overline{02}\ \overline{13},$$

using the following encoding of the elements of $\bar{Q}$:

$$0 \longleftrightarrow (\alpha,\alpha)$$

$$1 \longleftrightarrow (\alpha,\beta)$$

$$2 \longleftrightarrow (\beta,\alpha)$$

$$3 \longleftrightarrow (\beta,\beta).$$

Then $(\rho_{\{1\}}, \rho_{\{2\}})$ and $(\rho_{\{2\}}, \rho_{\{1\}})$ are both partition pairs. Accordingly, the second coordinate of the next state can be determined from the input symbol and the first coordinate of the present state; also, the first coordinate of the present state can be determined from the input and the second coordinate of the present state.

Even for this simple example, the determination of the Mm-pairs of $\equiv$ is arduous, and we omit it.

The next problem we consider is that of finding the self-sufficient elements of a pair algebra $\Delta \subset L \times L$. For any element $x \in L$, let

$$\hat{x} = \pi\{x_k \mid x \leq x_k \text{ and } (x_k, x_k) \in \Delta\};$$

then $\hat{x}$ is the least self-sufficient element having x as a lower bound. We shall give an algorithm for determining $\hat{x}$ given x, on the assumption that, given any element $y \in L$, m(y) can be computed. We begin by noting that an element y is self-sufficient if and only if

$$m(y) \leq y,$$

or, equivalently,

$$y + m(y) = y.$$

Now, let x be self-sufficient, and suppose $z \leq x$. Then

$$m(z) \leq m(x),$$

so that

$$z + m(z) \leq x + m(x) = x.$$

Then, unless z is self-sufficient,

$$z < z + m(z) \leq x.$$

Therefore, x may be computed by constructing a sequence $x^{(1)}, x^{(2)}, \ldots, x^{(h)}$, where

$$x^{(1)} = x,$$

$$x^{(j+1)} = x^{(j)} + m(x^{(j)}),$$

and h is the least value of j such that

$$x^{(j-1)} = x^{(j)};$$

such a value of h must occur, since

$$x^{(1)} \leq x^{(2)} \leq \ldots \leq x^{(j)} \leq x^{(j+1)} \leq \ldots,$$

and L is a finite lattice.

By a dual argument, $\bar{y}$, the greatest self-sufficient element less than or equal to y, may be obtained by applying the following iteration process:

$y^{(1)} = y$; for $j \geq 1$,

$$y^{(j+1)} = y^j \cdot M(y^{(j)}).$$

One special case of this iteration process was used in Section 7, where an algorithm was given for determining the indistinguishability relation associated with a sequential machine $\mathcal{M}$. Problem 7-5 characterizes this relation as the coarsest equivalence relation with S. P. which refines the relation ρ given by

$$q_1 \rho q_2 \iff \delta(q_1) = \delta(q_2).$$

The algorithm given in Section 7 determines $\bar{\rho}$ from ρ, using the iteration process given above, specialized to the case of the pair algebra π.

Example 8.

Let $\overline{\mathcal{M}}$ be a machine with $\overline{Q} = \{0, 1, 2, 3, 4, 5, 6, 7\}$, $A_I = \{a_1, a_2, a_3, a_4, a_5\}$, and the following transition function.

$\overline{Q}$ \ A_I

	a_1	a_2	a_3	a_4	a_5
0	1	0	4	7	2
1	0	1	5	6	3
2	3	2	5	5	0
3	2	3	4	4	1
4	4	5	2	3	6
5	5	4	3	2	7
6	6	7	3	1	4
7	7	6	2	0	5

We consider the pair algebra π, and choose $x = \sigma_{06}$. The computation of $\hat{x}$ is as follows.

j	$x^{(j)}$	$m(x^{(j)})$
1	$\overline{06}\ \overline{1}\ \overline{2}\ \overline{3}\ \overline{4}\ \overline{5}\ \overline{7}$	$\overline{0167}\ \overline{234}\ \overline{5}$
2	$\overline{0167}\ \overline{234}\ \overline{5}$	$\overline{0167}\ \overline{2345}$
3	$\overline{0167}\ \overline{2345}$	$\overline{0167}\ \overline{2345}$

The result of the iteration is that $\hat{x} = \overline{0167}\ \overline{2345}$.

For any join-irreducible element a_i of L, let $a_i^{(1)}$ denote a_i, and for $j \geq 1$, let

$$a_i^{(j+1)} = a_i^{(j)} + m(a_i^{(j)});$$

also, let $\hat{a}_i$ denote the least self-sufficient element greater than or equal to a_i. If

$$x = \sum_{\{i \mid a_i \leq x\}} a_i,$$

then

$$x^{(1)} = \sum_{\{i \mid a_i \leq x\}} a_i^{(1)}$$

$$x^{(2)} = x^{(1)} + m(x^{(1)})$$

$$= \sum_{\{i \mid a_i \leq x\}} a_i^{(1)} + m\Big(\sum_{\{i \mid a_i \leq x\}} a_i^{(1)} \Big)$$

$$= \sum_{\{i \mid a_i \leq x\}} a_i^{(1)} + \sum_{\{i \mid a_i \leq x\}} m(a_i^{(1)})$$

$$= \sum_{\{i \mid a_i \leq x\}} a_i^{(1)} + m(a_i^{(1)})$$

$$= \sum_{\{i \mid a_i \leq x\}} a_i^{(2)}.$$

Similarly, for all j,

$$x^{(j)} = \sum_{\{i \mid a_i \leq x\}} a_i^{(j)};$$

therefore,

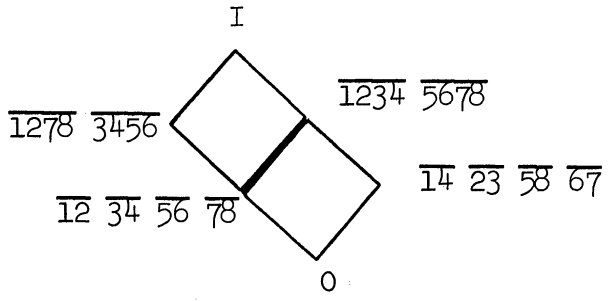
$$\hat{x} = \sum_{\{i \mid a_i \leq x\}} \hat{a}_i.$$

Since every self-sufficient element is equal to $\hat{x}$ for some x, the lattice of self-sufficient elements may be generated by taking all possible joins of elements $\hat{a}_i$.

Example 9. For the machine given in Example 8, the equivalence relations with S.P. of the form $\hat{a}_i$, where a_i is join-irreducible are determined as follows:

a_i	$\hat{a}_i$
0	0
$\sigma_{12}, \sigma_{34}, \sigma_{56}, \sigma_{78}$	$\overline{12}\ \overline{34}\ \overline{56}\ \overline{78}$
$\sigma_{14}, \sigma_{23}, \sigma_{58}, \sigma_{67}$	$\overline{14}\ \overline{23}\ \overline{58}\ \overline{67}$
$\sigma_{17}, \sigma_{18}, \sigma_{27}, \sigma_{28}, \sigma_{35}, \sigma_{36}, \sigma_{45}, \sigma_{46}$	$\overline{1278}\ \overline{3456}$
$\sigma_{13}, \sigma_{24}, \sigma_{57}, \sigma_{68}$	$\overline{1234}\ \overline{5678}$
$\sigma_{15}, \sigma_{16}, \sigma_{25}, \sigma_{26}, \sigma_{37}, \sigma_{38}, \sigma_{47}, \sigma_{48}$	I

In this case, no further equivalence relations with S.P. are obtained by taking joins of the elements $\hat{a}_i$. The lattice of equivalence relations with S.P. for the given machine is therefore given by the following diagram.



If Δ is a pair algebra over $L \times L$, then the operations m and M each map L into L. Let $m^{[n]}(x)$ be defined recursively in the usual way:

$$m^{[1]}(x) = m(x);$$

$$m^{[n]}(x) = m(m^{[n-1]}(x)),$$

$n=2,3,\ldots,$;

let $M^{[n]}(x)$ be defined similarly.

Lemma 16

For $n=1,2,\ldots,$ $\quad x \leq M^{[n]}(m^{[n]}(x))$.

PROOF: We proceed by induction on n. Property (vi) in Table 1 on page 144 establishes the result for n=1. Assuming as an induction hypothesis that the result is true for n=k, consider $M^{[k+1]}m^{[k+1]}(x)$, which is equal to $M^{[k]}(M(m(m^{[k]}(x))))$. But, again using property (vi),

$$m^{[k]}(x) \leq M(m(m^{[k]}(x)));$$

and, by repeated application of the dual form of property (ii),

$$M^{[k]}(m^{[k]}(x)) \leq M^{[k]}(M(m(m^{[k]}(x)))) = M^{[k+1]}(m^{[k+1]}(x)).$$

By the induction hypothesis,

$$x \leq M^{[k]}(m^{[k]}(x)) \leq M^{[k+1]}(m^{[k+1]}(x)).$$

This completes the proof.

The following corollary is obtained by duality.

Corollary 2

For $n=1,2,\ldots$ $\quad m^{[n]}(M^{[n]}(y)) \leq y.$

Corollary 3

$m^{[n]}(I) = 0$ if and only if $M^{[n]}(0) = I.$

PROOF: Suppose

$$m^{[n]}(I) = 0.$$

Since

$$I \leq M^{[n]}(m^{[n]}(I)),$$

$$I \leq M^{[n]}(0).$$

This is possible if

$$M^{[n]}(0) = I.$$

Therefore

$$m^{[n]}(I) = 0 \Longrightarrow M^{[n]}(0) = I.$$

By duality, the reverse implication also holds.

The result given in Corollary 3 is applicable to the problem of testing whether a transition system is n-definite. A transition system

$$\tau = (A_I, Q, q_0, \lambda)$$

is said to be n-definite if, for any two states q_1 and q_2, and any word y such that $lg(y) \geq n$,

$$\lambda(q_1,y) = \lambda(q_2,y).$$

For any word $y \in A_I^*$, let

$$L(y) = \{\lambda(q,y),\ q \in Q\}.$$

For k=0,1,2,..., . Let $C^{(k)}$ be the following set system:

$$C^{(k)} = \text{Max}\ (\{L(y) \mid \lg(y) = k\} \cup 0).$$

Then τ is n-definite if and only if

$$C^{(n)} = 0.$$

Also,

$$C^{(k+1)} = \text{Max}\ (\{L(ya) \mid a \in A_I \text{ and } \lg(y) = k\} \cup 0)$$

$$= \text{Max}\ (\{\lambda(L(y),a) \mid a \in A_I \text{ and } \lg(y) = k\} \cup 0).$$

Therefore,

$$C^{(k+1)} = m(C^{(k)}),$$

where 'm' is the m operation in the pair algebra $\equiv$. Since

$$C^{(0)} = I,$$

$$C^{(n)} = m^{[n]}(I).$$

Therefore, τ is n-definite if and only if

$$m^{[n]}(I) = 0.$$

Applying Corollary 3, we now immediately obtain a second characterization of n-definiteness; τ is n-definite if and only if

$$M^{[n]}(0) = I.$$

Let us review what has been accomplished so far in our discussion of the state assignment problem. First, we defined the relation " $\mathcal{M}$ realizes $\bar{\mathcal{M}}$ ", and defined a state-assigned machine $\mathcal{M}$ as one in which the state set Q is a subset of some Cartesian product

$$R_I \times R_2 \times \ldots \times R_s.$$

We observed that, in a state-assigned machine $\mathcal{M}$, the transition

function λ can be represented by s partial functions $\lambda_1,\lambda_2,\ldots,\lambda_s$, where $\lambda_k(r_1,r_2,\ldots,r_s,a)$ has its domain in

$$R_1 \times R_2 \times \ldots \times R_s \times A_I,$$

and has R_k as its range. For any set

$$S \subseteq \{1,2,\ldots,s\},$$

the equivalence relation ρ_S on Q was defined, and it was established that, given

$$S \subseteq \{1,2,\ldots,s\}$$

and

$$T \subseteq \{1,2,\ldots,s\},$$

all of the functions λ_k, $k \in T$, are determined by the arguments r_i, $i \in S$, together with the argument a, if and only if (ρ_S,ρ_T) is a partition pair.

Next, it was shown that, if (σ,τ) is a partition pair for $\mathscr{M}$, a realization of $\bar{\mathscr{M}}$, then the homomorphism

$$h: Q \to \bar{Q},$$

maps the blocks of σ onto the blocks of a cover φ, and the blocks of τ onto the blocks of a cover θ, such that (φ,θ) is a cover pair for $\bar{\mathscr{M}}$. Conversely, it was shown that, given a cover pair for $\bar{\mathscr{M}}$, one could construct a realization $\mathscr{M}$ of $\bar{\mathscr{M}}$ having a partition pair (σ,τ) such that the homomorphism h maps the blocks of σ onto the blocks of φ, and the blocks of τ onto the blocks of θ. Thus, the cover pairs of $\bar{\mathscr{M}}$ determine, and are determined by, the partition pairs for the realizations of $\bar{\mathscr{M}}$.

Systems pairs were then introduced, and the systems pairs for $\bar{\mathscr{M}}$

were shown to determine the cover pairs for $\bar{\mathcal{M}}$. With appropriate lattice ordering relations for the partitions of $\bar{Q}$ and the set systems associated with $\bar{Q}$, the sets of partition pairs and systems pairs for a machine $\bar{\mathcal{M}}$ were seen to form pair algebras. The properties of pair algebras were then studied. Of particular importance were the functions m(x) and M(y), the lattice of Mm-pairs, and the lattice of self-sufficient elements.

An approach to the determination of the Mm-pairs and the self-sufficient elements of a pair algebra was then given. These procedures appear to be fairly satisfactory in the case of the pair algebra Π of partition pairs, but unsatisfactory in the case of the pair algebra $\equiv$ of systems pairs.

We shall now consider the problem of constructing state assignments with reduced state dependence, once the partition pairs and systems pairs of $\bar{\mathcal{M}}$ have been determined. To simplify matters we shall restrict our attention to state-assigned realizations of $\bar{\mathcal{M}}$ which are isomorphic with $\bar{\mathcal{M}}$. Thus, if

$$\mathcal{M} = (A_I, Q, A_O, q_0, \lambda, \delta),$$

where

$$Q \subset R_1 \asymp R_2 \asymp \ldots \asymp R_s,$$

the homomorphism from $\mathcal{M}$ onto $\bar{\mathcal{M}}$ is determined by a function

$$h: Q \to \bar{Q}$$

which is one-to-one and onto. If ρ is a partition of Q, then h determines a partition $h(\rho)$ of $\bar{Q}$; the blocks of $h(\rho)$ are the images under h of the blocks of ρ. Clearly, (ρ_1,ρ_2) is a partition pair for $\mathcal{M}$ if and only if $(h(\rho_1), h(\rho_2))$ is a partition pair for $\bar{\mathcal{M}}$.

In order to obtain a well defined combinatorial problem, we shall further restrict consideration to p-minimal state assignments. For any integer $p \geq 2$, a p-minimal state assignment is one in which

$$Q \subseteq R_1 \times R_2 \times \ldots \times R_s,$$

where

(i) for each i, $|R_i| \leq p$, where '$|R_i|$' denotes the number of elements in R_i;

and

(ii) $s = \lceil \log_p |\bar{Q}| \rceil$, where $\lceil x \rceil$ denotes the least integer greater than or equal to x.

Thus, in particular, 2-minimal state assignments correspond to sequential circuits which have the least number of latches of any circuits associated with realizations of $\bar{\mathcal{M}}$.

Given a collection $\{(\theta_i, \varphi_i)\}$ of partition pairs for $\bar{\mathcal{M}}$, let

$$\{\sigma_1, \sigma_2, \ldots, \sigma_m\} = \bigcup_i \{\theta_i, \varphi_i\}.$$

Then, given $p \geq 2$, we may ask whether there exists a p-minimal state assignment which "uses" all the given partition pairs, in the following sense: for each index j, there is a set S_j such that

$$\sigma_j = h(\rho_{S_j}),$$

where, of course, ρ_{S_j} is the partition induced on Q by those coordinates of the state assignment with indices in S_j.

In order to discuss this question, we shall make a few definitions and elementary combinatorial remarks. Given a partition σ of a finite set, let $b(\sigma)$ denote the number of blocks of σ, and let $c(\sigma)$ denote the

largest number of elements in any block of σ. If $\sigma \leq \rho$, let ρ/σ denote the following partition of the set of blocks of σ: B_i and B_j are elements of the same block of ρ/σ if and only if they are subsets of the same block of ρ.

Example 10.

Suppose

$$\rho = \overline{1234}\ \overline{567}$$

and

$$\sigma = \overline{123}\ \overline{4}\ \overline{5}\ \overline{67}.$$

Then

$$b(\rho) = 2,$$

$$c(\rho) = 4,$$

$$b(\sigma) = 4,$$

$$c(\sigma) = 3.$$

Also, ρ/σ has the following two blocks: $\{\overline{123}\ \overline{4}\}$ and $\{\overline{5}\ \overline{67}\}$;

$$b(\rho/\sigma) = 2,$$

$$c(\rho/\sigma) = 2.$$

Lemma 17

Let σ and ρ be partitions of $\bar{Q}$, with $\sigma \leq \rho$. Then the minimum value of $b(\tau)$, over all partitions τ such that $\rho\tau = \sigma$, is $c(\rho/\sigma)$.

PROOF: Let τ be such that $\rho\tau = \sigma$. Then any two blocks of σ contained in the same block of ρ must be in distinct blocks of τ; therefore,

$$b(\tau) \geq c(\rho/\sigma).$$

To construct a partition τ such that

$$\rho\tau = \sigma$$

and

$$b(\tau) = c(\rho/\sigma),$$

order the elements (blocks of σ) of each block of ρ/σ. Then, for

$$1 \leq i \leq c(\rho),$$

let the i-th block of τ be the union of all those blocks of σ which are i-th elements of blocks of ρ/σ.

Corollary 4

Let σ be a partition of $\bar{Q}$. Then the minimum value of $b(\tau)$, over all partitions τ such that

$$\sigma\tau = 0,$$

is $c(\sigma)$.

We also remark, that, in any state-assigned machine, and for any set S of indices,

$$\rho_S = \prod_{i\epsilon S} \rho_{\{i\}}.$$

Since, for any two partitions σ and τ,

$$b(\sigma\tau) \leq b(\sigma)b(\tau),$$

$$b(\rho_S) \leq \prod_{i\epsilon S} b(\rho_{\{i\}}).$$

In the particular case of a p-minimal state assignment,

$$b(\rho_{\{i\}}) \leq p,$$

so that

$$b(\rho_S) \leq p^{|S|}.$$

Also, if $\bar{S}$ is the complement of S in the set $\{1,2,\ldots,s\}$, then

$$\rho_S \cdot \rho_{\bar{S}} = \rho_{\{1,2,\ldots,s\}} = 0.$$

Therefore

$$p^{|\bar{S}|} \geq b(\rho_{\bar{S}}) \geq c(\rho_S).$$

Accordingly,

$$\begin{aligned} b(\rho_S) \cdot c(\rho_S) &\leq p^{|S|} p^{|\bar{S}|} \\ &\leq p^{|S + \bar{S}|} \\ &\leq p^{\lceil \log_p |Q| \rceil}. \end{aligned}$$

If $\sigma = h(\rho_S)$,

then $b(\sigma) = b(\rho_S)$

and $c(\sigma) = c(\rho_S)$.

Therefore, unless

$$b(\sigma) \cdot c(\sigma) \leq p^{\lceil \log_p |Q| \rceil},$$

σ is not used in any p-minimal state assignment. Since $b(\rho_S) \leq p^{|S|}$ and $c(\rho_S) \leq p^{|\bar{S}|}$,

$$\lceil \log_p b(\rho_S) \rceil \leq |S|$$

and

$$\lceil \log_p c(\rho_S) \rceil \leq |\bar{S}|.$$

We shall show that these inequalities must hold as equations. Assuming

that one of the inequalities is strict, we have the following:

$$\lceil \log_p |\bar{Q}| \rceil = |S| + |\bar{S}|$$

$$> \lceil \log_p b(\rho_S) \rceil + \lceil \log_p c(\rho_S) \rceil .$$

Then

$$\lceil \log_p |\bar{Q}| \rceil - 1 \geq \lceil \log_p b(\rho_S) \rceil + \lceil \log_p c(\rho_S) \rceil ,$$

and

$$p^{\lceil \log_p |\bar{Q}| \rceil - 1} \geq p^{\lceil \log_p b(\rho_S) \rceil + \lceil \log_p c(\rho_S) \rceil}$$

$$\geq b(\rho_S) \cdot c(\rho_S)$$

$$\geq |\bar{Q}| .$$

But

$$\lceil \log_p |\bar{Q}| \rceil - 1 < \log_p |\bar{Q}| ,$$

so that a contradiction has been reached. Summing up, we have the following result.

Lemma 18

If σ is a partition of $\bar{Q}$ and $\mathcal{M}$ has a p-minimal state assignment such that $\sigma = h(\rho_S)$, then

$$|S| = \lceil \log_p b(\rho_S) \rceil$$

and

$$\lceil \log_p b(\sigma) \rceil + \lceil \log_p c(\sigma) \rceil = |\bar{Q}| .$$

Example 11.

Consider a sequential machine having the following transition function.

		A_I		
		a_1	a_2	a_3
$\tilde{Q}$	0	2	6	7
	1	3	6	1
	2	1	5	6
	3	0	4	2
	4	0	6	3
	5	0	2	4
	6	1	0	4
	7	5	2	6

This machine has the partition pairs

$$(\sigma_1,\sigma_2) = (\overline{01}\ \overline{26}\ \overline{34}\ \overline{57};\ \overline{05}\ \overline{17}\ \overline{46}\ \overline{23})$$

and

$$(\sigma_2,\sigma_3) = (\overline{05}\ \overline{17}\ \overline{46}\ \overline{23};\ \overline{3547}\ \overline{0126}).$$

Let us determine whether $\tilde{\mathcal{M}}$ has a two-minimal state assignment such that, for appropriate sets S_i,

$$\sigma_i = h(\rho_{S_i}),$$

i=1,2,3. Since

$$\lceil \log_2 |Q| \rceil = \lceil \log_2 8 \rceil = 3,$$

the assignment is such that

$$Q \subset R_1 \asymp R_2 \asymp R_3,$$

where, for i=1,2,3, $|R_i| = 2$.

To check that the conditions of Lemma 18 are satisfied, we make the following tabulation.

i	$\lceil S_i \rceil = \lceil \log_2 b(\sigma_i) \rceil$	$\lceil \log_2 c(\sigma_i) \rceil$	$\lceil \log_2 b(\sigma_i) \rceil + \lceil \log_2 c(\sigma_i) \rceil$
1	2	1	3
2	2	1	3
3	1	2	3

The condition that

$$\lceil \log_2 b(\sigma_i) \rceil + \lceil \log_2 c(\sigma_i) \rceil = 3$$

is satisfied for all i, and the quantities $|S_i|$ have been determined. Before continuing this example, we need the following combinatorial lemma, which is a generalization of the well-known principle of inclusion and exclusion.

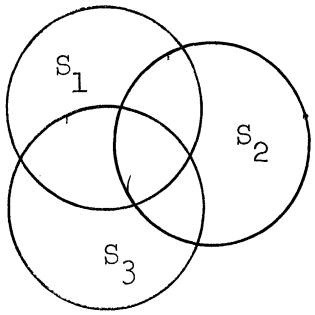
Lemma 19

Let $S_1, S_2, \ldots, S_m$ be subsets of a finite set U, and let T be a nonempty subset of $\{1,2,\ldots,m\}$. Let $\bar{S}_j$ denote the complement of S_j in U, and let $\bar{T}$ denote the complement of T in $\{1,2,\ldots,m\}$. Then

$$\Big| \bigcap_{j \in T} S_j \cap \bigcap_{j \in \bar{T}} \bar{S}_j \Big| = \sum_{R \mid R \subset T} (-1)^{|R|+1} \Big| \bigcup_{j \in \bar{T} \cup R} S_j \Big|$$

Example 12.

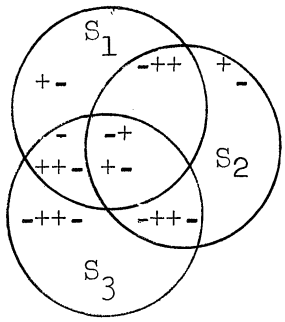
Let the sets S_1, S_2, and S_3 be represented on a Venn diagram, as follows.



Applying the lemma to the case $T = \{1,2\}$, we obtain

$$|S_1 \cap S_2 \cap \bar{S}_3| = -|S_3| + |S_1 \cup S_3| + |S_2 \cup S_3| - |S_1 \cup S_2 \cup S_3|.$$

This may be checked by putting a plus sign in a region of the Venn diagram for every set containing the region which occurs with coefficient +1, and a minus sign for every set containing the region which occurs with coefficient -1.



The plus and minus signs cancel out, except in the region $S_1 \cap S_2 \cap \bar{S}_3$, which has one more plus sign than minus sign.

Now we may apply Lemma 19 to the set of partitions introduced in Example 11.

Example 13.

We have already determined that

$$|S_1| = 2,$$

$$|S_2| = 2,$$

and $$|S_3| = 1.$$

Also $$\sigma_1 \cdot \sigma_2 = 0$$

$$= h(\rho_{S_1}) \cdot h(\rho_{S_2})$$

$$= h(\rho_{S_1 \cup S_2}).$$

Therefore $$|S_1 \cup S_2| = |\log_2 b(0)|$$

$$= 3.$$

Similarly, it is determined that

$$|S_1 \cup S_3| = 2,$$

$$|S_2 \cup S_3| = 3,$$

and $$|S_1 \cup S_2 \cup S_3| = 3.$$

Applying Lemma 19, we find that

$$|S_1 \cap S_2 \cap \bar{S}_3| = |\bar{S}_1 \cap S_2 \cap \bar{S}_3|$$

$$= |S_1 \cap \bar{S}_2 \cap S_3|$$

$$= 1,$$

and

$$|S_1 \cap S_2 \cap S_3| = |\bar{S}_1 \cap S_2 \cap S_3|$$

$$= |\bar{S}_1 \cap \bar{S}_2 \cap S_3|$$

$$= |S_1 \cap \bar{S}_2 \cap \bar{S}_3|$$

$$= 0.$$

With no loss of generality, we may index the sets R_1, R_2, and R_3 in any

2-minimal state assignment using σ_1, σ_2, and σ_3 so that

$$S_1 \cap S_2 \cap \bar{S}_3 = \{1\},$$

$$\bar{S}_1 \cap S_2 \cap \bar{S}_3 = \{2\},$$

and

$$S_1 \cap \bar{S}_2 \cap S_3 = \{3\}.$$

Then

$$S_1 = \{1,3\},$$

$$S_2 = \{1,2\},$$

and

$$S_3 = \{3\}.$$

Using the symbol τ_i to denote $h(\rho_{\{i\}})$, we obtain

$$\sigma_1 = h(\rho_{\{1,3\}})$$

$$= h(\rho_{\{1\}})h(\rho_{\{3\}})$$

$$= \tau_1 \cdot \tau_3,$$

$$\sigma_2 = \tau_1 \cdot \tau_2, \quad \text{and} \quad \sigma_3 = \tau_3.$$

Then $$\sigma_1 = \tau_1 \cdot \tau_3 \leq \tau_1,$$

and

$$\sigma_2 = \tau_1 \cdot \tau_2 \leq \tau_1,$$

therefore,

$$\sigma_1 + \sigma_2 = \overline{0157}\ \overline{2346} \leq \tau_1,$$

and either

$$\tau_1 = \overline{0157}\ \overline{2346} \quad \text{or} \quad \tau_1 = I.$$

The latter case is clearly impossible, since the necessary condition of Lemma 18 would be violated. Forming the product $\tau_2 \cdot \tau_3$, we obtain σ_1;

if σ_1 were not obtained, it would be known that no 2-minimal state assignment using $\{\sigma_1,\sigma_2,\sigma_3\}$ exists. Finally, choosing for τ_2 a two-block partition such that

$$\tau_1 \cdot \tau_2 = \sigma_2,$$

we obtain

$$\tau_2 = \overline{0546}\ \overline{1237}.$$

This determines the state assignment completely. The function

$$h: R_1 \times R_2 \times R_3 \to \bar{Q}$$

is as follows:

r_1	r_2	r_3	$h(r_1,r_2,r_3)$
0	0	0	5
0	0	1	0
0	1	0	7
0	1	1	1
1	0	0	4
1	0	1	6
1	1	0	3
1	1	1	2

The function h determines the transition function for $\mathcal{M}$, the desired state-assigned realization of $\mathcal{M}$, as follows.

		A_I		
		a_1	a_2	a_3
Q	001	111	101	010
	011	110	101	011
	111	011	000	101
	110	001	100	111
	100	001	101	110
	000	001	111	100
	101	011	001	100
	010	000	111	101

It is easily verified that $(\rho_{\{1,3\}}, \rho_{\{1,2\}})$ and $(\rho_{\{1,2\}}, \rho_{\{3\}})$ are partition pairs for $\mathcal{M}$. Thus, the transition function λ of $\mathcal{M}$

determines transition functions λ_1, λ_2, λ_3 for the individual coordinates such that

$$\lambda_1 = \lambda_1(r_1, r_3, a)$$

$$\lambda_2 = \lambda_2(r_1, r_3, a)$$

$$\lambda_3 = \lambda_3(r_1, r_2, a).$$

11. LOOP-FREE DECOMPOSITIONS OF SEQUENTIAL MACHINES

We shall next turn our attention to a special class of state-assigned realizations called loop-free decompositions. A loop-free decomposition of $\bar{\mathcal{M}}$ may be reviewed as a realization of $\bar{\mathcal{M}}$ by an interconnected set

$$\{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_s\}$$

of machines such that, for any k, the inputs to $\mathcal{N}_k$ consist of the external inputs, together with the outputs of

$$\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_{k-1}.$$

The main theorem of this section is due to Krohn and Rhodes [13] and Zeiger [19]. It asserts that every machine has a loop-free decomposition into two-state identity-reset machines and machines whose function monoids are simple groups.

Let $\bar{\mathcal{M}} = (A_I, \bar{Q}, A_O, \bar{q}_0, \bar{\lambda}, \bar{\delta})$ be a finite-state sequential machine. A <u>loop-free decomposition</u> of $\bar{\mathcal{M}}$ is a state-assigned realization $\mathcal{M}$ of $\bar{\mathcal{M}}$ such that

$$\mathcal{M} = (A_I, Q, A_O, \bar{q}_0, \bar{\lambda}, \bar{\delta}),$$

there exist finite sets $R_1, R_2, \ldots, R_s$ such that

$$Q \subseteq R_1 \times R_2 \times \ldots \times R_s,$$

and for $k=1,2,\ldots,s$,

$$\rho_{\{1,2,\ldots,k\}}$$

is a partition with S.P.. Such a decomposition is <u>strict</u> if $\mathcal{M}$ is isomorphic with $\bar{\mathcal{M}}$. A loop-free decomposition of $\bar{\mathcal{M}}$ is said to be a <u>parallel decomposition</u> if, for each k, $\rho_{\{k\}}$ is a partition with S.P..

It follows that, in a loop-free decomposition, each coordinate transition function λ_k is of the form $\lambda_k(r_1,\ldots,r_k,a)$. Accordingly, the first k coordinates of a loop-free decomposition $\mathcal{M}$ determine a sequential machine

$$\mathcal{P}_k = (A_I, Q_k, Q_k, (r_1(q_0),r_2(q_0),\ldots,r_k(q_0)), \mu_k, \epsilon_k),$$

where

$$Q_k \subseteq R_1 \times R_2 \times \ldots \times R_k,$$

$$\mu_k((r_1,r_2,\ldots,r_k),a) = (\lambda_1(r_1,a),\lambda_2(r_1,r_2,a),\ldots,\lambda_k(r_1,r_2,\ldots,r_k,a)),$$

and ϵ_k is an identity function. If $\mathcal{T}(\mathcal{M})$ is the transition system derived from $\mathcal{M}$ by disregarding outputs, and $\mathcal{T}(\mathcal{P}_k)$ is similarly derived from $\mathcal{P}_k$, then $\mathcal{T}(\mathcal{P}_k)$ is isomorphic with

$$\mathcal{T}(\mathcal{M})/\rho_{\{1,2,\ldots,k\}}.$$

If $\mathcal{M}$ is a strict realization of $\bar{\mathcal{M}}$, then $\mathcal{T}(\mathcal{P}_k)$ is also isomorphic with

$$\mathcal{T}(\bar{\mathcal{M}})/h(\rho_{\{1,2,\ldots,k\}}).$$

It is also convenient to introduce the machines

$$\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k,$$

which are the <u>components</u> of the loop-free decomposition;

$$\mathcal{N}_k = (A_I \times Q_{k-1}, R_k, R_k, r_k(q_0), \omega_k, \varphi_k),$$

where

$$\omega_k(r_k,(a,r_1,r_2,\ldots,r_{k-1})) = \lambda_k(r_1,r_2,\ldots,r_k,a)$$

and φ_k is an identity function. If the loop-free decomposition is

parallel, then ω_k is determined solely by r_k and a. In this case, we define the parallel component

$$\mathcal{N}'_k = (A_I, R_k, R_k, r_k(q_0), \omega'_k, \varphi_k),$$

where

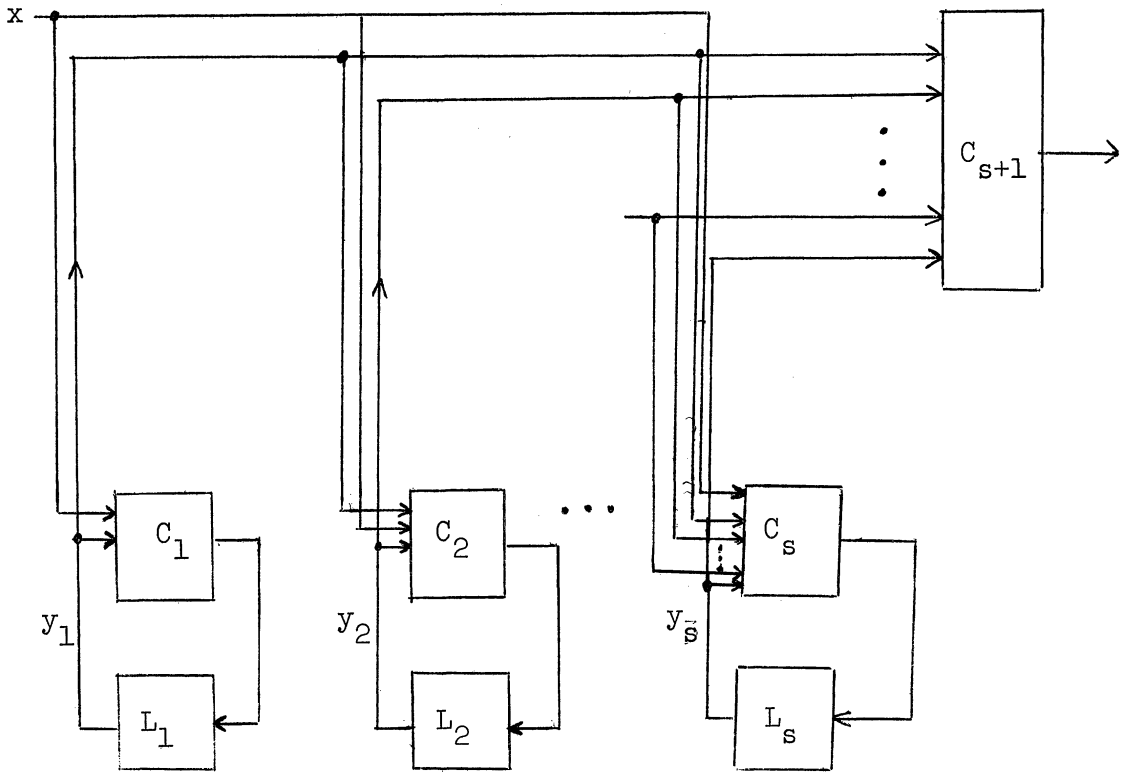
$$\omega'_k(r_k,a) = \lambda_k(r_k,a);$$

in this case, $\tau(\mathcal{N}'_k)$ is isomorphic with $\tau(\mathcal{M})/\rho_{\{k\}}$.

If the input alphabet of $\overline{\mathcal{M}}$ is $\{0,1\}^{(p)}$, the output alphabet is $\{0,1\}^{(r)}$, and $\overline{\mathcal{M}}$ has a loop-free decomposition in which, for each k,

$$R_k = \{0,1\}^{(\ell_k)},$$

then there is a sequential circuit corresponding to $\overline{\mathcal{M}}$ which has the following form.



Each component $\mathcal{N}_k$ is represented by a subcircuit consisting of a combinational part C_k and a set of latches L_k. The output is computed in the combinational circuit C_{s+1}. The symbol x denotes a "bundle" of p binary signals representing an element of the input alphabet. Similarly, z represents an output r-tuple, and for each k, y_k is an ℓ_k-tuple giving the state of the component $\mathcal{N}_k$.

It follows from the definitions of a loop-free decomposition, and of the machines $\mathcal{P}_k$ and $\mathcal{N}_k$, that, for

$$0 < k_1 < k_2 \leq s,$$

$\mathcal{P}_{k_2}$ has a loop-free decomposition with components

$$\mathcal{P}_{k_1}, \mathcal{N}_{k_1+1}, \ldots, \mathcal{N}_{k_2}.$$

Also, decompositions may be iterated, as follows. Suppose that $\mathcal{M}$ has a loop-free decomposition with components

$$\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_s,$$

and, for some k, $\mathcal{N}_k$ has a loop-free decomposition $\mathcal{R}$ with components

$$\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_t.$$

Let the set of states of $\mathcal{A}_i$ be S_i, and let h be the homomorphism from the state set of $\mathcal{R}$ onto R_k, the state set of $\mathcal{N}_k$. Then $\mathcal{M}$ has a loop-free decomposition with the ordered set of components

$$\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_{k-1},$$
$$\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_t,$$
$$\hat{\mathcal{N}}_{k+1}, \ldots, \hat{\mathcal{N}}_s.$$

The component $\hat{\mathcal{N}}_p$ is a modification of $\mathcal{N}_p$ determined as follows:

$$\hat{\mathcal{N}}_p = (A_I \times Q_{k-1} \times S_1 \times S_2 \times \cdots \times S_t \times R_{k+1} \times \cdots \times R_{p-1}, R_p, R_p, r_p(q_0)\ \omega_p, \varphi_p),$$

where

$$\hat{\omega}_p(r_p, (a, r_1, \ldots, r_{k-1}, s_1, \ldots, s_t, r_{k+1}, \ldots r_{p-1}) = \omega_p(r_p, (a, r_1, \ldots, r_{k-1}, h(s_1, \ldots, s_t), r_{k+1}, \ldots, r_{p-1}).$$

It will be convenient to note for future reference that the function monoid of $\hat{\mathcal{N}}_p$ is the same as that of $\mathcal{N}_p$, and that $\hat{\mathcal{N}}_p$ has the same number of states as $\mathcal{N}_p$.

If $\mathcal{M}$ is a loop-free decomposition of $\bar{\mathcal{M}}$, then $\mathcal{M}$ has the following chain of partitions with S.P.:

$$\rho_{\{1\}} \geq \rho_{\{1,2\}} \geq \rho_{\{1,2,3\}} \geq \cdots \geq \rho_{\{1,2,\ldots,s\}} = 0.$$

If the set system

$$\mathrm{Max}[h(\rho_{\{1,2,\ldots,k\}})]$$

is denoted by σ_k, then $\bar{\mathcal{M}}$ has the following chain of set systems with S.P.:

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_s = 0.$$

The strict loop-free decompositions of $\bar{\mathcal{M}}$ may be obtained by inspection of its lattice of partitions with S.P. . In the case of a strict loop-free decomposition, each element of the chain $\sigma_1, \sigma_2, \ldots, \sigma_s$ is, of course, a partition with S.P. . Also, if the partition $h(\rho_{\{k\}})$ is denoted τ_k, then the following system of equations holds:

$$\sigma_1 = \tau_1,$$

and for k=2,...,s

$$\sigma_k = \sigma_{k-1} \cdot \tau_k.$$

Conversely, given a chain of partitions with S.P.,

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_s = 0,$$

one can clearly construct a sequence of partitions $\tau_1, \tau_2, \ldots, \tau_n$ such that, for all k,

$$\prod_{i=1}^{k} \tau_i = \sigma_k;$$

by Lemma 10-17, τ_k can be chosen so that

$$b(\tau_k) = c(\sigma_{k-1}/\sigma_k).$$

A state-assigned realization $\mathcal{M}$ of $\bar{\mathcal{M}}$ can then readily be constructed such that $\mathcal{M}$ is isomorphic with $\bar{\mathcal{M}}$ and, for all k,

$$h(\rho_{\{k\}}) = \tau_k.$$

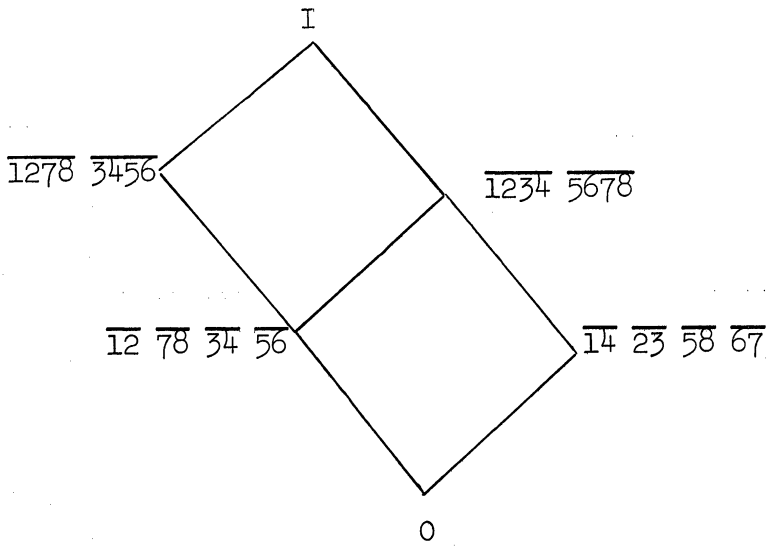
It then follows that, for all k,

$$h(\rho_{\{1,2,\ldots,k\}}) = \sigma_k,$$

so that $\rho_{\{1,2,\ldots,k\}}$ is a partition with S.P. for $\mathcal{M}$. Thus $\mathcal{M}$ is a strict loop-free decomposition of $\bar{\mathcal{M}}$.

Example 1.

For the machine $\bar{\mathcal{M}}$ considered in Examples 10-7 and 10-8, the lattice of partitions with S.P. is as follows.



Let us consider the following chain of partitions with S.P.:

$$\sigma_1 = \overline{1278}\ \overline{3456};$$

$$\sigma_2 = \overline{12}\ \overline{34}\ \overline{56}\ \overline{78};$$

$$\sigma_3 = 0.$$

Then

$$\tau_1 = \overline{1278}\ \overline{3456},$$

and we may choose τ_2 and τ_3 as follows:

$$\tau_2 = \overline{1234}\ \overline{5678},$$

$$\tau_3 = \overline{1357}\ \overline{2468}.$$

The following state assignment is such that, for k=1,2,3,

$$h(\rho_{\{k\}}) = \tau_k.$$

r_1	r_2	r_3	$h(r_1,r_2,r_3)$
0	0	0	1
0	0	1	2
1	0	0	3
1	0	1	4
1	1	0	5
1	1	1	6
0	1	0	7
0	1	1	8

The transition function λ for the state-assigned machine $\mathcal{M}$ is determined from the transition function $\bar{\lambda}$ for $\bar{\mathcal{M}}$, using the basic identity characteristic of homomorphisms:

$$\bar{\lambda}(h(q),a) = h(\lambda(q,a)).$$

The table for the function λ is as follows:

λ		A_I				
		a_1	a_2	a_3	a_4	a_5
Q	000	001	000	110	011	100
	001	000	001	111	010	101
	100	101	100	111	111	000
	101	100	101	110	110	001
	110	110	111	100	101	010
	111	111	110	101	100	011
	010	010	011	101	001	110
	011	011	010	100	000	111

It can now be checked that $\rho_{\{1\}}$ and $\rho_{\{1,2\}}$ have S.P. . Also, since τ_2 has S.P., $\rho_{\{2\}}$ has S.P., so that $\mathcal{M}/\rho_{\{1,2\}}$ has a parallel decomposition with parallel components isomorphic to $\mathcal{M}/\rho_{\{1\}}$ and $\mathcal{M}/\rho_{\{2\}}$.

We turn our attention next to the proof of a basic theorem about loop-free decompositions. The statement and proof given here are essentially due to Zeiger [19]; the original statement and proof are due to Krohn and Rhodes [13].

A sequential machine

$$\mathcal{M} = (A_I, Q, A_O, q_0, \lambda, \delta)$$

is an identity-reset machine if, for every input a, one of the following holds:

(i) $\lambda(q,a) = q$, for all q,

(ii) there is a state $q' \in Q$ such that, for all q, $\lambda(q,a) = q'$.

Theorem 1

Every finite-state machine has a loop-free decomposition in which each of the components $\mathcal{M}_k$ satisfies one of the following two conditions:

a) $\mathcal{M}_k$ is a two-state identity-reset machine;

b) the function monoid of $\mathcal{M}_k$ is a simple group.

In proving this theorem, we shall use some concepts and results from elementary group theory. Let G be a finite group with multiplication operation $\cdot$. A subset H of G is a subgroup of G if H is closed under the operation $\cdot$. If H is a subgroup of G then H contains 1, the identity element of G; also, if $a \in H$, then $a^{-1} \in H$. If G is a finite group, the order of G is the number of elements of G. If H is a subgroup of G, then for any element $g \in G$, the set

$$gH = \{gh \mid h \in H\}$$

is called the right coset of g relative to H. Left cosets are defined similarly. Any two right cosets associated with H have the same number of elements, and two right cosets are either identical or disjoint. The number of right cosets is called the index of H in G. These statements also hold for left cosets. Clearly, the order of G is the product of the order of H and the index of H in G.

A subgroup H is called normal if and only if, for all g, gH = Hg. An equivalent statement is that, for all g,

$$g^{-1}Hg = \{g^{-1}hg \mid h \in H\}$$

is equal to H. If H is normal, then the equivalence relation

$$g_1 \sim g_2 \iff g_1H = g_2H$$

is a two-sided congruence over G, and determines a homomorphism

$$G \to G/H.$$

The factor monoid G/H is a group, and is called the factor group of G modulo H. All two-sided congruences over G are induced by normal subgroups and therefore all homomorphic images of G are groups of the form G/H. A group G is simple if the only normal subgroups of G are {1} and G itself.

A composition series for G is a sequence

$$G = G_1 \supseteq G_2 \supseteq \cdots \supseteq G_s = \{1\},$$

where, for i=1,2,...,s-1, G_{i+1} is a normal subgroup G_i, and G_{i+1} is maximal; i.e., there is no normal subgroup H of G such that

$$G_i \supseteq H \supseteq G_{i+1},$$

$$G_i \neq H,$$

and

$$G_{i+1} \neq H.$$

In a composition series, all the factor groups G_i/G_{i+1} are simple. The Jordan-Hölder Theorem states that, if G has two composition series

$$G = G_1 \supseteq G_2 \supseteq \cdots \supseteq G_s = \{1\}$$

and

$$G = H_1 \supseteq H_2 \supseteq \cdots \supseteq H_t = \{1\},$$

then there is a one-to-one correspondence between the sets

$$\{G_i/G_{i+1}, \quad i=1,2,\ldots,s-1\}$$

and

$$\{H_i/H_{i+1}, \quad i=1,2,\ldots,t-1\},$$

such that corresponding factor groups are isomorphic. This implies, in

particular, that all composition series for G are of the same length. The factor groups associated with a composition series for G are called the composition factors of G.

A finite-state sequential machine with states S and transition function μ is a permutation-reset machine if, for every input symbol a, the function μ_a defined by the equation,

$$\mu_a(s) = \mu(s,a)$$

is either a permutation of S or a constant function, independent of s.

PROOF: (of Theorem 1):

As a first step, we shall construct, for an arbitrary finite-state machine

$$\bar{\mathcal{M}} = (A_I, \bar{Q}, A_o, \bar{q}_0, \bar{\lambda}, \bar{\delta}),$$

a loop-free decomposition

$$\mathcal{M} = (A_I, Q, A_o, q_0, \lambda, \delta)$$

in which all components are permutation-reset machines. If $|\bar{Q}|$ is n, then $\mathcal{M}$ will have n-1 components, so that

$$Q \subseteq R_1 \times R_2 \times \ldots \times R_{n-1};$$

also, for k=1,2,...,n-1, R_k will be equal to {1,2,...,n-k}.

The state assignment underlying this construction can be represented schematically by a tree. Part of the tree for the case n=4 is shown in the following figure. In the general case, the root of the tree is labelled '123...n', and each vertex at distance k from the root (also called a k-th level vertex) is labelled with a set of n-k distinct elements of {1,2,...,n}. Each k th-level vertex v is connected to

k (k+1)-th level vertices, labelled with the distinct (n-(k+1))-tuples obtained by deleting an element from the label of v; the edges from v to these vertices are numbered 1,2,...,n-k, in an arbitrary order.

The states of the realization $\mathcal{M}$ are in one-to-one correspondence with the leaves of the tree (the (n-1)-th level vertices); the name of a state is given by the sequence of labels of edges in the path from the root to the associated leaf, and the image of a state of $\mathcal{M}$ under the homomorphism h from $\mathcal{M}$ to $\overline{\mathcal{M}}$ is the label of the corresponding leaf. Thus, in the tree of Figure 1,

$$h(1,3,2) = 3.$$

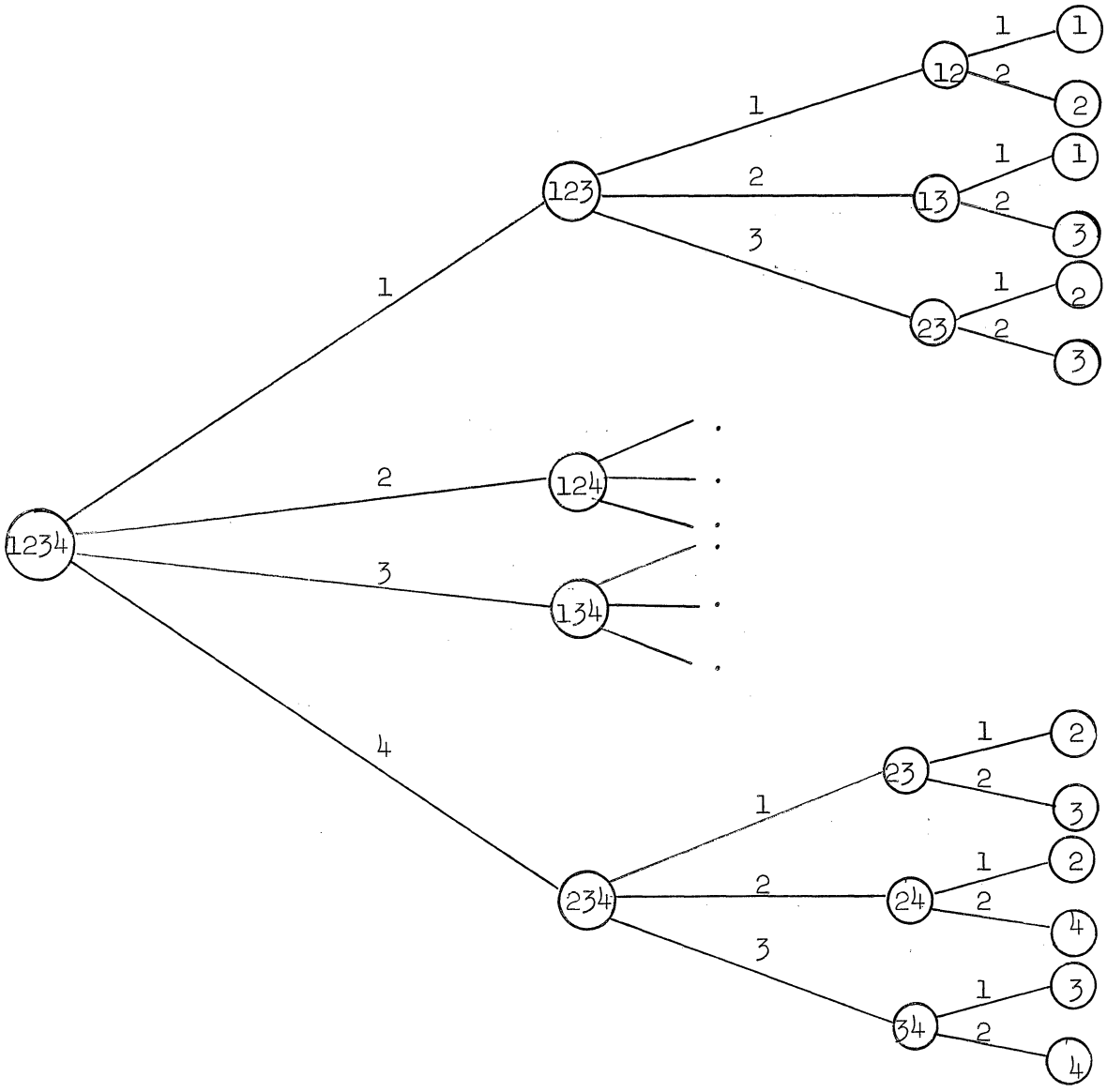


Figure 1

Let $[\alpha_1,\ldots,\alpha_k]$ be the block of $\rho_{\{1,2,\ldots,k\}}$ consisting of all states of $\mathcal{M}$ whose first k coordinates are $\alpha_1,\ldots,\alpha_k$. Then $h([\alpha_1,\ldots,\alpha_k])$ is given by the label of the unique k-th level node connected to the root by a path with $\alpha_1,\alpha_2,\ldots,\alpha_k$ as its sequence of edge labels. Consequently, the subsets of $\bar{Q}$ are obtained by deleting precisely one element from $h([\alpha_1,\ldots,\alpha_k])$ are the following:

$$h([\alpha_1,\ldots,\alpha_k 1]),\ h([\alpha_1,\ldots,\alpha_k 2]),\ \ldots,\ h([\alpha_1,\ldots,\alpha_k,n-k]).$$

The construction of the transition function λ of the realization will be recursive. The functions

$$\lambda_1(r_1,a),\ \lambda_2(r_1,r_2,a),\ \ldots,\ \lambda_k(r_1,r_2,\ldots,r_k,a)$$

will be defined successively. It will be convenient to recall that the function μ_k is defined as follows:

$$\mu_k(r_1,r_2,\ldots,r_k,a) = (\lambda_1(r_1,a),\lambda_2(r_1,r_2,a),\ldots,\lambda_k(r_1,r_2,\ldots,r_k,a)).$$

The function λ must satisfy the identity

$$(1)\quad h(\lambda(q,a)) = \bar{\lambda}(h(q),a).$$

From this it follows that, if $S \subseteq Q$, then

$$(2)\quad h(\lambda(S,a)) = \bar{\lambda}(h(S),a),$$

where

$$\lambda(S,a) = \{\lambda(q,a) \mid q \in S\}$$

and

$$h(S) = \{h(q) \mid q \in S\}.$$

But, if $\mu_k(\alpha_1,\ldots,\alpha_k,a) = (\beta_1,\ldots,\beta_k)$, then

$$\lambda(h([\alpha_1,\alpha_2,\ldots,\alpha_k]),a) \subseteq [\beta_1,\ldots,\beta_k],$$

so that

$$h[\lambda([\alpha_1,\alpha_2,\ldots,\alpha_k]),a)] \subseteq h([\beta_1,\ldots,\beta_k]).$$

Accordingly, the function μ_k must satisfy the following condition.

$$(3)\quad \begin{cases} \text{if } \mu_k(\alpha_1,\alpha_2,\ldots,\alpha_k,a) = (\beta_1,\beta_2,\ldots,\beta_k), \text{ then} \\ \\ \bar{\lambda}(h([\alpha_1,\alpha_2,\ldots,\alpha_k]),a) \subseteq h([\beta_1,\ldots,\beta_k]). \end{cases}$$

In the case k = n-1, this condition is equivalent to (1).

We shall now construct the functions $\lambda_1,\lambda_2,\ldots,\lambda_{n-1}$ so that (3) holds for all k and each component is a permutation-reset machine. The construction of λ_1 is illustrative of the general method. For $a \in A_I$, two cases are possible:

(1) $\bar{\lambda}(\bar{Q},a)$ is a proper subset of $\bar{Q}$;

(2) $\bar{\lambda}(\bar{Q},a) = \bar{Q}$.

Case 1.

In this case, there is a set $h([\alpha])$ such that

$$\bar{\lambda}(\bar{Q},a) \subseteq h([\alpha]).$$

For all $j \in R_1$, set

$$\lambda_1(j,a) = \alpha.$$

Then (3) is not violated and the input a causes a reset to the state α in the component $\mathcal{N}_1$.

Case 2.

Because the input a permutes the states of $\bar{Q}$, it follows that there is a unique permutation π of R_1 such that, for j=1,2,...,n,

$$\bar{\lambda}(h[j]) = h[\pi(j)].$$

For all $j \in R_1$, set

$$\lambda_1(j,a) = \pi(j).$$

Then (3) is not violated, and the input a permutes the states of $\mathcal{N}_1$.

Suppose now that μ_k satisfies (3), that the components

$$\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k$$

are permutation-reset, and that μ_{k+1} is to be constructed. If

$$\mu_k(\alpha_1,\alpha_2,\ldots,\alpha_k,a) = (\beta_1,\beta_2,\ldots,\beta_k),$$

then two cases can occur:

(1) $\bar{\lambda}(h[\alpha_1,\alpha_2,\ldots,\alpha_k])$ is a proper subset of $h([\beta_1,\beta_2,\ldots,\beta_k])$.

(2) $\bar{\lambda}(h([\alpha_1,\alpha_2,\ldots,\alpha_k]),a)$ is equal to $h([\beta_1,\beta_2,\ldots,\beta_k])$.

The construction of $\mu_{k+1}(\alpha_1,\alpha_2,\ldots,\alpha_k,\alpha_{k+1},a)$ in these two cases is as follows.

Case 1.

In this case, there exists β^*_{k+1} such that

$$\bar{\lambda}(h([\alpha_1,\alpha_2,\ldots,\alpha_k]),a) \subseteq h[\beta_1,\beta_2,\ldots,\beta^*_{k+1}].$$

For all coordinates α_{k+1}, set $\lambda_{k+1}(\alpha_1,\alpha_2,\ldots,\alpha_{k+1},a)$ equal to β^*_{k+1}. Then (3) is not violated and the action of the component $\mathcal{N}_{k+1}$ under the input $(a,\alpha_1,\ldots,\alpha_k)$ is to reset to the state β^*_{k+1}.

Case 2.

In this case, there is a unique permutation π of R_{k+1} such that, for all j,

$$\bar{\lambda}(h([\alpha_1,\alpha_2,\ldots,\alpha_k,j]),a) = h([\beta_1,\beta_2,\ldots,\beta_k,\pi(j)]).$$

Setting $\lambda_{k+1}(\alpha_1,\alpha_2,\ldots,\alpha_k,j,a)$ equal to $\pi(j)$, we find that condition (3) is not violated, and that the input $(\alpha_1,\alpha_2,\ldots,\alpha_k,a)$ permutes the states of $\mathcal{N}_{k+1}$.

Thus, we have given an inductive construction of a transition function λ such that all components are permutation-reset, and such that the identity

$$\bar{\lambda}(h[(\alpha_1,\alpha_2,\ldots,\alpha_{n-1}],a) = h(\lambda([\alpha_1,\alpha_2,\ldots,\alpha_{n-1})],a))$$

holds. By appropriate choice of the initial state and the output function of $\mathcal{M}$, the remaining conditions for a realization may be satisfied. Thus, we have shown how to produce a loop-free decomposition of $\bar{\mathcal{M}}$ into permutation-reset components.

Example 1.

Let $\bar{\mathcal{M}}$ be a four-state machine with the following transition function.

$\bar{\lambda}$		A_I	
		a	b
$\bar{Q}$	1	2	2
	2	4	3
	3	4	4
	4	2	1

We shall compute $\lambda([1,2,2],a)$. First, since

$$\bar{\lambda}(\bar{Q},a) \subseteq h([4]) = \{1,2,3\},$$

we set

$$\lambda_1(j,a) = 4$$

for all j; in particular,

$$\lambda_1(1,a) = 4.$$

Since

$$\bar{\lambda}(h([1]),a) = \bar{\lambda}(\{1,2,3\},a) = \{2,4\} = h([4,2]),$$

we set

$$\bar{\lambda}_2(1,j,a) = 2$$

for all j. In particular,

$$\lambda_2(1,2,a) = 2.$$

Since

$$\bar{\lambda}(h([1,2],a)) = h([4,2]),$$

for each j there is a unique element $\pi(j)$ such that

$$\bar{\lambda}(h([1,2,j]),a) = h([4,2,\pi(j)]).$$

In particular,

$$\pi(2) = 2,$$

so that

$$\lambda_3(1,2,2,a) = 2,$$

and

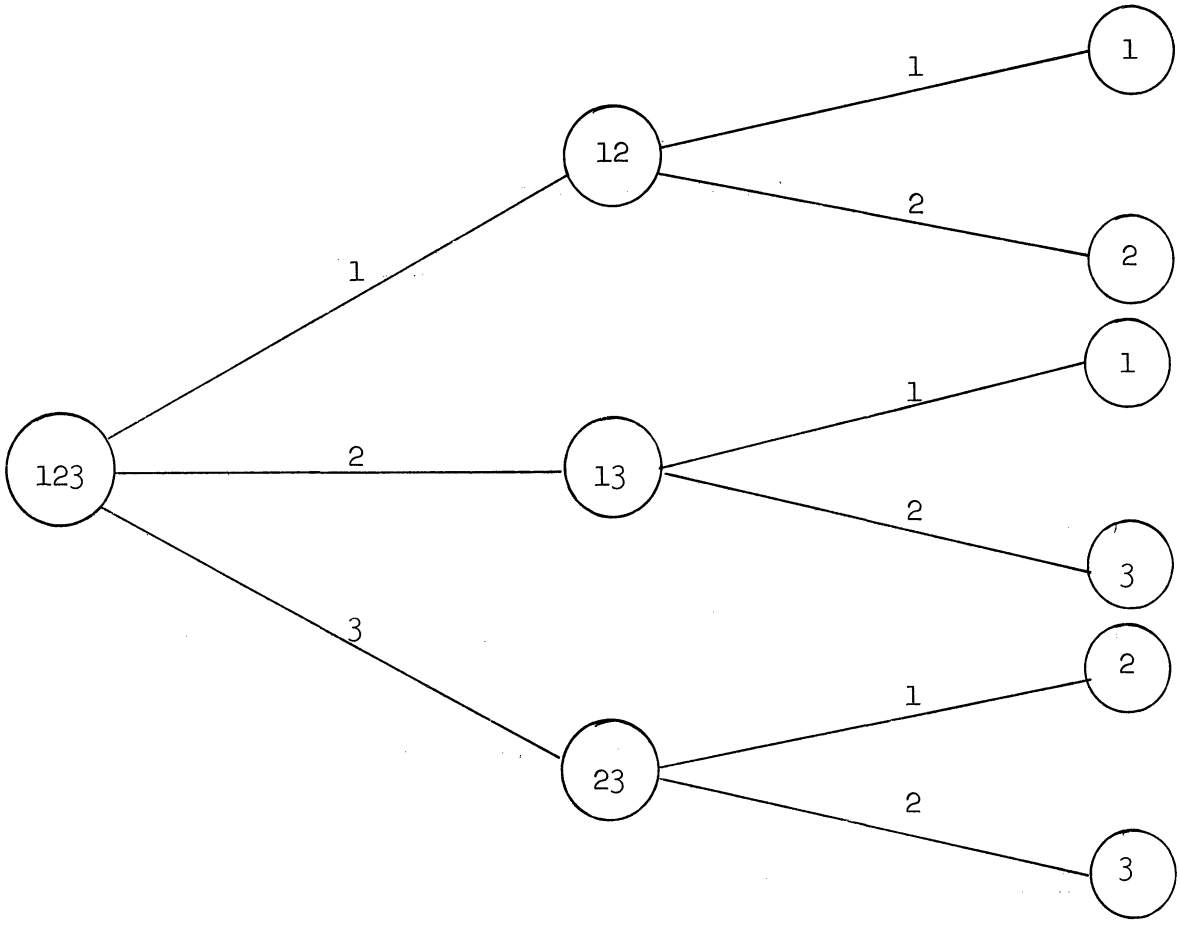
$$\lambda((1,2,2),a) = (4,2,2).$$

Example 2.

Let $\bar{\mathcal{M}}$ be a three-state machine with the following transition function.

$\bar{\lambda}$		A_I	
		a	b
$\bar{Q}$	1	2	3
	2	3	1
	3	3	2

The state assignment to be used is exhibited by the following tree.



$$\bar{\lambda}(\bar{Q},a) = \{2,3\} = h([31]).$$

Therefore

$$\lambda_1(j,a) = 3$$

for all j.

$$\bar{\lambda}(\bar{Q},b) = \bar{Q}\ .$$

Therefore, the input b permutes the states of $\mathcal{N}_1$, as follows:

$$1 \to 2,$$

$$2 \to 3,$$

$$3 \to 1.$$

The table for λ_1 is as follows.

λ_1		A_I	
		a	b
R_1	1	3	2
	2	3	3
	3	3	1

The function λ_2 is then determined as follows.

λ_2		$A_I \times R_1$					
		a_1	a_2	a_3	b_1	b_2	b_3
R_2	1	1	1	2	2	2	1
	2	2	2	2	1	1	2

The next step in proving Theorem 1 is to show that any permutation-reset machine has a loop-free decomposition into two components: a permutation machine, and an identity-reset machine. A finite-state machine with state set S and transition function μ is a <u>permutation machine</u> if every function μ_a is a permutation.

Let

$$\bar{\mathcal{M}} = (A_I, \bar{Q}, A_o, \bar{q}_0, \lambda, \bar{\delta})$$

be a permutation-reset machine. Let

$$P \subseteq A_I$$

be determined as follows: $a \in P$ if and only if $\bar{\lambda}_a$ is a permutation of $\bar{Q}$ (of course, for $x \in A_I^*$,

$$\bar{\lambda}_x(\bar{q}) = \bar{\lambda}(\bar{q},x)).$$

If P is empty, then $\bar{\mathcal{M}}$ is a reset machine, and does not have to be decomposed further at this step. Otherwise, let G be the group of permutations of $\bar{Q}$ generated by $\{\lambda_a \mid a \in P\}$, and let $\cdot$ be the multiplication operation

of G. We shall construct a loop-free realization $\mathcal{M}$ of $\overline{\mathcal{M}}$ in which $\mathcal{N}_1$, the first component, is a permutation machine with the set of states G, and $\mathcal{N}_2$, the second component, is an identity-reset machine with $\bar{Q}$ as its set of states.

In specifying λ, the transition function of $\mathcal{M}$, it will be convenient to represent the identity

$$h(\lambda(q,a)) = \bar{\lambda}(h(q),a)$$

by a commutative diagram, of the following form:

$$\begin{array}{ccc} Q & \xrightarrow{h} & \bar{Q} \\ {\scriptstyle \lambda_a}\downarrow & & \downarrow{\scriptstyle \bar{\lambda}_a} \\ Q & \xrightarrow{h} & \bar{Q}. \end{array}$$

In this case,

$$Q = G \times \bar{Q},$$

and the homomorphism h maps the ordered pair

$$(\Gamma, q) \in G \times \bar{Q}$$

onto the element

$$\Gamma(q) \in \bar{Q}.$$

In specifying λ, two cases must be distinguished.

Case 1.

$a \in P$. Then $\bar{\lambda}_a$ is a permutation. The specification of λ_a is shown in the following commutative diagram.

$$\begin{array}{ccc} (\Gamma, q) & \longrightarrow & \Gamma(q) \\ {\scriptstyle \lambda_a}\downarrow & & \downarrow{\scriptstyle \bar{\lambda}_a} \\ (\bar{\lambda}_a \cdot \Gamma, q) & \xrightarrow{h} & \bar{\lambda}_a \cdot \Gamma(q) \end{array}$$

Thus, in this case,

$$\lambda((\Gamma,q),a) = (\bar{\lambda}_a \cdot \Gamma, q).$$

Case 2.

$a \notin P$. Then $\bar{\lambda}_a$ is a reset, mapping every state onto some state r. The specification of λ_a is shown in the following diagram.

$$\begin{array}{ccc} (\Gamma,q) & \xrightarrow{h} & \Gamma(q) \\ \lambda_a \downarrow & & \downarrow \bar{\lambda}_a \\ (\Gamma,\Gamma^{-1}(r)) & \xrightarrow{h} & r \ . \end{array}$$

The loop-free decomposition of $\bar{\mathcal{M}}$ may now be described as follows. If the first component $\mathcal{N}_1$ is in the state Γ, and receives an input $a \in P$ it goes to the state $\bar{\lambda}_a \cdot \Gamma$; thus, the input a produces a permutation of G, the set of states of $\mathcal{N}_1$. If $\mathcal{N}_1$ is in the state Γ, and receives an input $a \notin P$, it remains in the state Γ; thus, the input a produces the identity permutation of G. If the second component $\mathcal{N}_2$ is in the state q and receives the input (a, Γ), where $a \in P$, it remains in the state q. If $\mathcal{N}_2$ is in the state q and receives the input (a, Γ), where $a \notin P$, it resets to the state $\Gamma^{-1}(r)$, where

$$r = \bar{\lambda}(\bar{q}, a)$$

for all $\bar{q}$. Thus, $\mathcal{N}_2$ is an identity-reset machine.

Example 3.

Let $\bar{\mathcal{M}}$ have the transition function $\bar{\lambda}$ given by the following table.

$\bar{\lambda}$		A_I		
		a_1	a_2	a_3
$\bar{Q}$	1	2	2	1
	2	3	1	1
	3	1	3	1

$\bar{\mathcal{M}}$ is a permutation-reset machine, and the group of permutations generated by λ_{a_1} and λ_{a_2} is S_3, the symmetric group of degree 3. Let the elements of this group be assigned names as follows:

a: $\begin{pmatrix}123\\123\end{pmatrix}$ b: $\begin{pmatrix}123\\132\end{pmatrix}$ c: $\begin{pmatrix}123\\213\end{pmatrix}$

d: $\begin{pmatrix}123\\231\end{pmatrix}$ e: $\begin{pmatrix}123\\312\end{pmatrix}$ f: $\begin{pmatrix}123\\321\end{pmatrix}$.

For example, the permutation e maps 1 into 3, 2 into 1, and 3 into 2. The transition function for the state-assigned machine $\mathcal{M}$ is as follows.

λ		A_I		
		a_1	a_2	a_3
Q	a,1	d,1	c,1	a,1
	a,2	d,2	c,2	a,1
	a,3	d,3	c,3	a,1
	b,1	c,1	d,1	b,1
	b,2	c,2	d,2	b,1
	b,3	c,3	d,3	b,1
	c,1	f,1	a,1	c,2
	c,2	f,2	a,2	c,2
	c,3	f,3	a,3	c,2
	d,1	e,1	b,1	d,3
	d,2	e,2	b,2	d,3
	d,3	e,3	b,3	d,3
	e,1	a,1	f,1	e,2
	e,2	a,2	f,2	e,2
	e,3	a,3	f,3	e,2
	f,1	b,1	e,1	f,3
	f,2	b,2	e,2	f,3
	f,3	b,3	e,3	f,3

The first component of $\mathcal{M}$ has the following transition function:

$\mathcal{N}_1$

λ_1		A_I		
		a_1	a_2	a_3
Q_1	a	d	c	a
	b	c	d	b
	c	f	a	c
	d	e	b	d
	e	a	f	e
	f	b	e	f

The second component performs the identity mapping under any input (a_1,Γ) or (a_2,Γ); for the other inputs, it performs resets, as follows.

$\mathcal{N}_2$

λ_2		$A_I \times Q_1$					
		a_3,a	a_3,b	a_3,c	a_3,d	a_3,e	a_3,f
R_2	1	1	1	2	3	2	3
	2	1	1	2	3	2	3
	3	1	1	2	3	2	3

Suppose a permutation-reset machine $\mathcal{M}$ has been decomposed, in the manner just described, into two components: a permutation machine $\mathcal{N}_1$ and an identity-reset machine $\mathcal{N}_2$. Let z be an element of A_I^*, and let x be the subsequence of z obtained by deleting those elements of z not in P (Recall that $a \in P$ if and only if $\bar{\lambda}_a$ is a permutation.). Then, for all $\Gamma \in G$,

$$\lambda_1(\Gamma,z) = \lambda_1(\Gamma,x),$$

since each deleted input leaves the state of $\mathcal{N}_1$ unchanged. Therefore, every function $(\lambda_1)_z(\Gamma)$ is identical with some function $(\lambda_1)_x(\Gamma)$, where $x \in P^*$. If x and y are elements of P^*, then, for all $\Gamma \in G$,

$$\lambda_1(\Gamma,x) = \bar{\lambda}_x \cdot \Gamma,$$

and

$$\lambda_1(\Gamma, y) = \bar{\lambda}_y \Gamma .$$

Accordingly, the functions $(\lambda_1)_x$ and $(\lambda_1)_y$ are identical if and only if $\bar{\lambda}_x$ and $\bar{\lambda}_y$ are identical. It follows, then, that the transition monoid of $\mathcal{N}_1$ is (abstractly) isomorphic to the group G of permutations $\bar{\lambda}_x$, $x \in P^*$. Since $\mathcal{N}_1$ is reduced, the function monoid of $\mathcal{N}_1$ is also isomorphic to G. As we shall see later, further decomposition of this permutation machine will be possible unless G is simple.

The second component $\mathcal{N}_2$ in the decomposition of $\mathcal{M}$ is an identity-reset machine. It is immediate that, in an identity-reset machine, every partition of the set of states has S.P. . Therefore, any strict state-assigned realization of $\mathcal{N}_2$ in which the states are binary s-tuples will be a parallel decomposition of $\mathcal{N}_2$ into two-state components; it is easy to see that these components are necessarily identity-reset.

At this point we have established that every finite-state machine has a loop-free decomposition in which all components are two-state machines or permutation machines. Moreover, each permutation machine is of the form

$$\bar{\mathcal{P}} = (A, G, G, 1, \bar{\lambda}, \bar{\delta}),$$

where the finite group G is the transition (and function) monoid of $\mathcal{P}$, 1 is the identity element of G, $\bar{\delta}$ is the identity function, and for each $a \in A$, there is an element $\Gamma_a \in G$ such that for all $\Gamma \in G$,

$$\bar{\lambda}(\Gamma, a) = \Gamma_a \cdot \Gamma .$$

Suppose that G has a proper normal subgroup N (i.e., $N \neq G$ and $N \neq \{1\}$). Then G is the union of the right cosets associated with N;

i.e.,

$$G = N \cup y_2N \cup \ldots \cup y_kN,$$

where k is the index of N in G. Let the set $\{1,y_2,\ldots,y_k\}$ of coset representatives be denoted by Y. Then $\mathcal{P}$ has a strict loop-free decomposition $\mathcal{M}$ with the two components $\mathcal{N}_1$ and $\mathcal{N}_2$, such that the function monoid of $\mathcal{N}_1$ is isomorphic to G/N, and the function monoid of $\mathcal{N}_2$ is N. The set of states of $\mathcal{M}$ is $Y \times N$. The homomorphism

$$h: Y \times N \to N,$$

and λ, the transition function of $\mathcal{M}$, are displayed in the following commutative diagram.

$$\begin{array}{ccc} (y,n) & \xrightarrow{\ h\ } & y\cdot n \\ \downarrow{\scriptstyle \lambda_a} & & \downarrow{\scriptstyle \bar{\lambda}_a} \\ w,w^{-1}\Gamma_a\cdot y\cdot n & \xrightarrow{\ h\ } & \Gamma_a\cdot y\cdot n. \end{array}$$

Here, w is the representative of the coset containing $\Gamma_a\cdot y$, so that $\Gamma_a\cdot y = wn_1$, for some $n_1 \in N$, and

$$w^{-1}\cdot \Gamma_a\cdot y = n_1.$$

When $\mathcal{N}_1$ is in the state y and the input is a, the next state is the representative of the coset containing y·a. In other words, the input a, applied to $\mathcal{N}_1$, permutes coset representatives exactly as premultiplication by $[\Gamma_a]$, the coset of Γ_a, multiplies cosets in G/N. Also, since the elements of Γ_a, $a \in A$, generate G, the cosets $[\Gamma_a]$, $a \in A$, generate G/N. Therefore, the transition monoid (or, equivalently, the function monoid) of $\mathcal{N}_1$ is isomorphic with G/N.

It remains to be shown that the transition monoid of $\mathcal{N}_2$ is N. We have already seen that the action of the input (y,a) on $\mathcal{N}_2$ is to maps each state n to $n_1 \cdot n$, where

$$n_1 = w^{-1} \cdot \Gamma_a \cdot y,$$

an element of N. To complete the proof that the transition monoid of $\mathcal{N}_2$ is N, we shall show that, for every element $n' \in N$, there is an input sequence to $\mathcal{N}_2$ that maps each state n to $n' \cdot n$. Since the elements Γ_a, $a \in A$, generate G, there is a sequence $x \in A_I^*$ such that, for all $g \in G$,

$$\bar{\lambda}(g,x) = n' \cdot g.$$

Since the homomorphism h from $Y \times N$ onto G is one-to-one, it is an isomorphism, and

$$h^{-1}: G \to Y \times N$$

is also an isomorphism. Since, for all $g \in G$,

$$\bar{\lambda}(g,x) = n' \cdot g,$$

it follows that

$$\lambda(h^{-1}(g),x) = h^{-1}(n' \cdot g).$$

Setting g=n, where n is an arbitrary element of N, and noting that

$$h^{-1}(n) = (1,n),$$

we find that

$$\lambda((1,n),x) = (1,n'n).$$

Therefore, there exists an input sequence to $\mathcal{N}_2$ mapping each state $n \in N$ to n'n. This completes the proof that the transition monoid (and function monoid) of $\mathcal{N}_2$ is N.

Example 4.

Consider a permutation machine $\mathcal{P}$ having the following transition function:

$\mathcal{P}$

λ		A_I	
		a_1	a_2
Q	0	1	2
	1	2	3
	2	3	0
	3	0	1

The transition monoid G of $\mathcal{P}$ is the cyclic group of order 4, which can be represented by the set of integers {0,1,2,3}, with group multiplication the same as addition modulo 4. Then

$$\Gamma_{a_1} = 1,$$

and

$$\Gamma_{a_2} = 2.$$

The group G has the normal subgroup

$$N = \{0,2\},$$

which partitions G into the cosets {0,2} and {1,3}. Let Y, the collection of coset representatives, be {0,1}. The transition function for $\mathcal{M}$ is:

$\mathcal{M}$

λ		A_I	
		a_1	a_2
Q	(0,0)	1,0	0,2
	(0,2)	1,2	0,0
	(1,0)	0,2	1,2
	(1,2)	0,0	1,0

The transition functions for the components $\mathcal{M}_1$ and $\mathcal{M}_2$ are as follows:

$\mathcal{M}_1$:

λ_1		A_I	
		a_1	a_2
Q_1	0	1	0
	1	0	1

$\mathcal{M}_2$:

λ_2		$A_I \times Q_1$			
		$a_1,0$	$a_1,1$	$a_2,0$	$a_2,1$
R_2	0	0	2	2	2
	2	2	0	0	0

We have now specified a procedure by which a machine with state set G and function monoid G can be decomposed into two components: $\mathcal{M}_1$, with state set in one-to-one correspondence with G/N and the function monoid G/N, and $\mathcal{M}_2$, with state set N and function monoid N, where N is a normal subgroup of G. The procedure can be iterated until all the components remaining are permutation machines whose groups are simple. Although various choices of a decomposition are possible at each step of this iterative process, the collection of simple groups ultimately obtained is uniquely determined by G; the simple groups which occur are the composition factors of G.

Thus we have completed the proof of Theorem 1. Every finite-state machine has a loop-free decomposition in which every component is either a two-state identity-reset machine or a permutation machine whose group is simple. By arguments that we shall not give, it is possible to specify exactly the simple groups that have to occur in any loop-free decomposition of a given finite-state machine $\bar{\mathcal{M}}$. Let $\mathcal{U}(\bar{\mathcal{M}})$ be the set consisting

of all the simple groups that are composition factors of subgroups of the function monoid of $\mathcal{M}$.

Theorem 2

Let $\mathcal{M}$ have a loop-free decomposition with components $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_s$. If $G \in \mathcal{H}(\mathcal{M})$, then $G \in \mathcal{H}(\mathcal{N}_i)$, for some i.

Theorem 3

$\mathcal{M}$ has a loop-free decomposition in which each component $\mathcal{N}_i$ is either a two-state identity-reset machine, or a permutation machine whose group is an element of $\mathcal{H}(\mathcal{M})$.

Corollary 1

$\mathcal{M}$ has a loop-free decomposition in which all components are two-state identity-reset machines if and only if $\mathcal{H}(\mathcal{M})$ contains only the group with one element.

The cyclic groups of prime order form an important class of simple groups. A group G is solvable if all its composition factors are cyclic of prime order. All finite abelian groups are solvable, and it has recently been shown by Feit and Thompson that all finite groups of odd order are solvable. Let p be a prime, and let $\mathcal{M}$ be a sequential machine whose function monoid is C_p, the cyclic machine $\mathcal{N}$ which realizes $\mathcal{M}$, for which the set of states is {0,1,...,p-1}, and for which each function

$$\lambda_a(q) = q + c(a) \pmod{p}.$$

Let such a machine be called a modulo p counter.

Corollary 2

$\mathcal{M}$ has a loop-free decomposition in which all components are two-state identity-reset machines and counters of prime modulos if and only if every subgroup of the function monoid of $\mathcal{M}$ is solvable.

12. TURING MACHINES AND RECURSIVELY UNSOLVABLE PROBLEMS

In this chapter we begin the study of classes of automata with greater capabilities than finite automata. Such a class C has the property that, with suitable definitions, every regular event is represented by an automaton in C. On the other hand, it will also be true that, with suitable definitions, every automaton that we consider will be capable of being simulated by a Turing machine; in fact, every digital, discrete-time, finitary automaton that has been conceived of up to now can be simulated by a Turing machine.

The fact that Turing machines seem to set an upper limit on the capabilities of automata would be sufficient reason for introducing them at this point. A second reason is that, as we begin to study more powerful automata, it will turn out that algorithms do not exist for solving certain problems of analysis. This was not the case in our study of sequential machines, where all the algorithms we sought turned out to exist. For example, we can determine algorithmically whether a given state of a sequential machine is accessible, whether the set of words accepted by a finite automaton is empty or infinite, whether two sequential machines represent the same function, and so forth. More complicated problems, such as deciding whether a sequential machine $\mathcal{M}$ exists such that the function $\text{seq}_{\mathcal{M}}(x)$ maps one given regular set onto another, also turn out to be solvable.

So long as the algorithms we sought could always be found, no general concept of "algorithm" had to be defined precisely; through experience, we have learned to recognize an algorithm when we see one. To show

that a certain problem is not decidable or not effectively solvable requires a higher degree of precision. The generally accepted definition of decidability hinges on the concept of a recursive set; and this concept can be developed in terms of Turing machines.

Turing machines and the concepts associated with recursive solvability are not exclusively the concern of automata theory; important undecidable problems occur in logic, number theory, and algebra. For this reason, we shall give only the development which is essential for our purposes, and refer the reader elsewhere (Davis [3], for example) for more information. We shall follow the formulation of Turing machines in terms of quadruples introduced by Post [34] and used in the book by Davis. Because we are more concerned in automata theory with recursive sets of words than with recursive sets of integers, our approach to the definition of a recursive set of words will be more direct than the usual ones (Davis, for example).

A Turing machine T consists of a control unit with a finite set of states Q (the internal states of T), together with a tape. The tape consists of a two-way infinite sequence of squares, each of which can contain any symbol from a finite alphabet A; A has a distinguished element a_o, the "blank" symbol. Initially, all but a finite number of squares of the tape contain a_o, and one square is being scanned by the control unit.

The operation of T is specified by a finite list of "commands" called quadruples. Each quadruple is of one of the following three forms:

Type 1 $\quad q_i\ a_j\ a_k\ q_\ell$

Type 2 $\quad q_i\ a_j\ R\ q_\ell$

Type 3 $\quad q_i\ a_j\ L\ q_\ell$.

For $(q_i,a_j) \in Q \times A$,

there is at most one quadruple containing q_i and a_j in the first two positions. When T is in state q_i and the square being scanned contains the symbol a_j, the command associated with (q_i,a_j) is executed. If no such command exists, T halts. The result of executing the command depends on its type, as follows:

Type 1

write a_k in the square being scanned, and enter state q_ℓ;

Type 2

enter state q_ℓ, and scan the square to the right of the present one;

Type 3

enter state q_ℓ, and scan the square to the left of the present one.

To make the definition of T quite precise, we introduce the concept of the complete states of T. A complete state of T is a triple

$$(q_i,x,y) \in Q \times A^* \times A^\dagger.$$

Such a triple corresponds to the situation in which the internal state is q_i, the tape contains the word xy, bounded on each side by an infinite sequence of copies of a_o, the "blank" symbol, and the square containing the first symbol of y is being scanned. Suppose

$$y = a_j y',$$

where $a_j \in A$. Then, if there is no quadruple with (q_i,a_j) in the first

two positions, the computation stops. If there is a quadruple of the form $q_i a_j a_k q_\ell$, then the next complete state is $(q_\ell, x, a_k y')$. If there is a quadruple of the form $q_i a_j R q_\ell$, two cases may occur:

1) if $y' \neq e$, then the next complete state is (q_ℓ, xa_j, y');

2) if $y' = e$, then the next complete state is (q_ℓ, xa_j, a_o).

If there is a quadruple of the form $q_i a_j L q_\ell$, two cases may occur:

(a) if $x = x'a_k$, $a_k \in A$, then the next complete state is $(q_\ell, x', a_k y)$;

(b) if $x = e$, then the next complete state is $(q_\ell, e, a_o y)$.

Thus, the initial complete state of a Turing machine determines whether the machine will ever stop, and what the final complete state will be if the machine does stop. Let P be a subset of A_I^*. The set P is recursively enumerable if there exists a Turing machine T which, given the initial complete state (q_0, e, x), halts if and only if $x \in P$; P is said to be recursive if there exists a Turing machine T having symbols a_1 and a_2 in its alphabet, such that, if T is given the initial complete state (q_0, e, x), it will halt with the final complete state (q_0, e, a_1) if $x \in P$, and will halt with the final complete state (q_0, e, a_2) if $x \notin P$. It can be shown that P is recursive if and only if both P and its complement in A^* are recursively enumerable.

The problem of deciding whether a given word x is an element of P is decidable, or recursively solvable, if P is recursive; otherwise this problem is undecidable, or recursively unsolvable.

We shall often speak about decidable or undecidable problems without introducing all the details that would be needed in order to specify the set P precisely. For example, consider the Post correspondence problem, which is usually stated as follows: Given two equally long ordered lists $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ of words over an alphabet A, to determine whether there exists a nonempty sequence of indices $i_1, i_2, \ldots, i_k$ such that the following two words are identical: $a_{i_1} a_{i_2} \ldots a_{i_k}$ and $b_{i_1} b_{i_2} \ldots b_{i_k}$. Post [33] has shown that, if A contains at least two elements, there is no algorithm for solving all instances of the Post correspondence problem.

This result will prove to be a basic tool for establishing undecidability results in automata theory. As the result is stated, however, it does not appear to have a precise meaning, since it does not seem to involve the recursiveness of a set of words. This can be rectified, for those who are fussy, by defining a format in which the data for any instance of the Post correspondence problem can be presented as an input word to a Turing machine. For example, let σ be a symbol not in A, and let the data be presented as a word over the alphabet

$$A \cup \{\sigma\},$$

as follows:

$$\sigma\, a_1\, \sigma\, a_2\, \sigma \ldots \sigma\, a_n\, \sigma\, b_1\, \sigma\, b_2\, \sigma \ldots \sigma\, b_n\, \sigma.$$

Then the unsolvability of the correspondence problem has the following precise formulation: the set consisting of all words in the above format, such that there exists a nonempty sequence $i_1, i_2, \ldots, i_k$ for which

$$a_{i_1} a_{i_2} \cdots a_{i_k} = b_{i_1} b_{i_2} \cdots b_{i_k},$$

is not recursive.

It may appear unsatisfactory that the statement of the result involves a particular data format. To see that this is not a serious objection, one should bear in mind that the result is invariant under any format conversion that can be effected by a Turing machine.

So far we have seen the use of Turing machines in formulating the concept of undecidability, or recursive unsolvability. In addition to this, Turing machines form the subject matter for a number of interesting undecidability results. For example, the problem of deciding whether a Turing machine, starting from the initial complete state (q_0, e, a_0), will halt, is recursively unsolvable.

In order to give a precise formulation of a question of decidability concerning Turing machines, we must reformulate that question as a question of the recursiveness of some set of words over a finite alphabet. One standard way of doing this is based on the technique of Gödel numbering. A Turing machine T is described by its set of quadruples, involving the symbols

$$R, L, a_0, a_1, \cdots, q_0, q_1, \cdots .$$

Let α be a function associating distinct positive odd integers greater than or equal to 3 with these symbols. For example, α might be chosen as follows:

$$\alpha(R) = 3,$$

$$\alpha(L) = 5,$$

$$\alpha(a_i) = 7 + 4i,$$

$$\alpha(q_i) = 9 + 4i.$$

Now, with each quadruple

$$S = (s_1, s_2, s_3, s_4)$$

we may associate an integer gn(S), the Gödel number of S, defined as

$$2^{\alpha(s_1)} \cdot 3^{\alpha(s_2)} \cdot 5^{\alpha(s_3)} \cdot 7^{\alpha(s_4)}.$$

Since we know how to obtain the unique prime factorization of an integer, and since the finite function α is one-to-one, there is an algorithm for recovering S from gn(S). Now, let the Turing machine T be specified by the quadruples $S_1, S_2, \ldots, S_n$, and let Pr(k) denote the k-th prime. Then a Gödel number for T is

$$\prod_{k=1}^{n} \Pr(k)^{gn(S_k)};$$

different Gödel numbers result from the n! different orderings of the quadruples. Clearly, there is an algorithm for recovering T from any of its Gödel numbers.

The word $1^{n+1}0$ over the alphabet {0,1} is said to be a representation of T if and only if n is a Gödel number for T. With this convention, we have a means of associating sets of Turing machines with sets of words, and thereby obtaining precisely stated undecidability questions and results. For example, the result that it is undecidable whether a Turing machine will eventually halt if presented with an

initially blank tape has the following precise formulation:

{x | x is a representation of T and T halts if its initial tape is blank}

is not recursive.

13. READ-ONLY AUTOMATA

In this section we discuss various classes of automata which can read their tapes, but cannot write on them. Many of the results to be described are due to Rabin and Scott [17]. Other contributions may be found in Minsky [30], as well as in Shepherdson [37], Moore [15], pp. 92-97, Rosenberg [36], Elgot and Rutledge [25], and Elgot and Mezei [24].

Within the restriction to read-only automata, there seem to be two ways to introduce a class of automata more powerful than finite automata:

i) allow the input tape to be read in either direction, rather than only left-to-right;

ii) provide more than one input tape.

Two-way automata

The first apparent generalization leads to the following definition. A two-way automaton $\mathcal{A}$ is a 6-tuple

$$\mathcal{A} = (A_I, Q, q_0, F, \lambda, D),$$

where the first five elements are defined in the same way as for a finite automaton (cf. Chapter 9), and D is a function from

$$Q \times A \text{ to } \{-1, +1\};$$

$D(q,a)$ specifies the direction of tape motion when $\mathcal{A}$ is in state q and the symbol a is read. Let

$$x = a_0 a_1 \cdots a_{n-1}$$

be an element of A_I^*. Then, associated with x, there is a unique sequence

$$(b_0,s_0), (b_1,s_1), \ldots, (b_r,s_r)$$

of elements of

$$\{-1, 0, 1, \ldots, n\} \times Q.$$

Intuitively, b_r gives the number of the square being scanned after r steps, and s_r denotes the state which occurs after r steps. The formal definitions are as follows:

$$(b_0, s_0) = (0, q_0);$$

for i=1,2,...,

$$(b_i, s_i) = (b_{i-1} + D(s_i, a_{b_i}), \lambda(s_i, a_{b_i})).$$

The sequence terminates if, for some r,

$$b_r = -1$$

or

$$b_r = n;$$

of course, the sequence may be infinite. The word

$$x = a_0 a_1 \cdots a_{n-1}$$

is <u>accepted</u> by $\mathcal{A}$ if the associated sequence terminates in a pair (b_r, s_r), where

$$b_r = n$$

and $s_r \in F$.

Surprisingly, two-way automata are no more powerful than finite automata, as the following theorem shows.

Theorem 1

Let $\mathcal{A}$ be a two-way automaton. Then $T(\mathcal{A})$, the set of words (tapes) accepted by $\mathcal{A}$, is regular.

The following proof of Theorem 1 is due to Shepherdson.

PROOF: It is sufficient to show that $T(\mathcal{A})$ has only a finite number of derivatives. Let σ be a symbol not in Q. We shall introduce two functions,

$$f: A_I^\dagger \to Q \cup \{\sigma\},$$

and

$$g: Q \times A_I^\dagger \to Q \cup \{\sigma\}.$$

Let

$$x = a_0 a_1 \cdots a_{n-1},$$

and consider the sequence

$$(b_0,s_0), \ldots, (b_r,s_r)$$

defined above. If the sequence fails to terminate, or if it terminates with

$$b_r = -1,$$

then

$$f(x) = \sigma;$$

if the sequence terminates with (b_r,s_r), where

$$b_r = n,$$

then

$$f(x) = s_r.$$

From the definition, it is clear that, if the word $xy \in A_I^*$ is presented to $\mathcal{A}$, and

$$f(x) = \sigma,$$

then xy is rejected without any symbol in the subsequence y ever being scanned; if

$$f(x) = s_r,$$

then $\mathcal{A}$ is in the state s_r when the first symbol of y is scanned for the first time. Let g(q,x) be defined in the same way, except that the starting point of the sequence is (n-1,q) instead of $(0,q_0)$. Suppose the word xy is presented to $\mathcal{A}$; if

$$g(q,x) = \sigma,$$

then, if $\mathcal{A}$ is in state q scanning the rightmost square of x (i.e., square n-1), then $\mathcal{A}$ will never again scan the leftmost square of y (i.e., square n); if

$$g(q,x) = r,$$

then, if $\mathcal{A}$ is in state q scanning square n-1, the next time it scans square n, it will be in state r.

Now, suppose x_1 and x_2 are elements of A_I^* such that

$$f(x_1) = f(x_2)$$

and the functions

$$g_{x_1}(q) = g(q,x_1)$$

and

$$g_{x_2}(q) = g(q,x_2)$$

are identical. Then, for any y,

$$x_1y \in T(\mathcal{A}) \iff x_2y \in T(\mathcal{A})$$

(I.e.,

$$D_{x_1}(T(\mathcal{A})) = D_{x_2}(T(\mathcal{A}))).$$

To see this, note that the computation of $\mathcal{A}$ when presented with the word xy alternates between phases in which symbols of x are read, and phases in which symbols of y are read; and, whether

$$x = x_1$$

or

$$x = x_2,$$

the results of the phases involving x will be the same. If

$$|Q| = n,$$

then there are only $(n+1)^{n+1}$ ways to specify the functions f(x) and $g_x(q)$, so that $T(\mathcal{A})$ has at most $(n+1)^{n+1}$ distinct derivatives. This completes the proof.

Two-tape automata

Since the provision of a single two-way tape does not augment the capabilities of finite-state machines (although it may result in the saving of states), we turn to an alternative possibility: a one-way two-tape read-only automaton. Such an automaton operates exactly as a finite automaton does, except that the state determines which of two input tapes is to be read next. Initially, tapes 1 and 2 contain the words $x_1\sigma$ and $x_2\sigma$, respectively, where $x_1 \in A_I^*$, and $x_2 \in A_I^*$, and $\sigma \notin A_I$; σ serves as an end marker which indicates that all the remaining squares on the tape are unmarked. The automaton stops when it tries to read an unmarked square beyond the end symbol σ. The ordered pair (x_1,x_2) is accepted if the automaton stops in a state which is an element of the designated set F of final states.

Formally, a two-tape automaton is a 7-tuple

$$\mathcal{A} = (A_I, Q, q_0, F, C_1, C_2, \lambda),$$

where A_I, Q, q_0, and F have the usual meanings, λ is a function

$$\lambda: Q \times (A_I \cup \{\sigma\}) \rightarrow Q,$$

and $\{C_1, C_2\}$ is a partition of Q. A complete state of $\mathcal{A}$ is a triple

$$(s,y,z) \in Q \times (e \cup A_I^*\sigma) \times (e \cup A_I^*\sigma);$$

s gives the internal state, and y and z give the words remaining to be read on the two tapes. The successor (s',y',z') of a complete state (s,y,z) is determined as follows: if $s \in C_1$ and

$$y = a\bar{y},$$

where

$$a \in A_I \cup \{\sigma\},$$

then

$$(s',y',z') = (\lambda(s,a),\bar{y},z);$$

if $s \in C_2$ and

$$z = a\bar{z},$$

where

$$a \in A_I \cup \{\sigma\},$$

then

$$(s',y',z') = (\lambda(s,a),y,\bar{z});$$

if $s \in C_1$ and

$$y = e,$$

or $s \in C_2$ and

$$z = e,$$

then (s,y,z) has no successor.

The operation of $\mathcal{A}$ when presented with the inputs $(x_1\sigma, x_2\sigma)$ is specified by a sequence

$$(s_0,y_0,z_0),\ (s_1,y_1,z_1),\ \ldots,\ (s_r,y_r,z_r)$$

of complete states such that

(a) $(s_0,y_0,z_0) = (q_0,x_1\sigma,x_2\sigma)$

(b) $(s_{i+1},y_{i+1},z_{i+1})$ is the successor of (s_i,y_i,z_i), $i=0,1,\ldots,r-1$.

(c) (s_r,y_r,z_r) has no successor.

The pair (x_1,x_2) is accepted if and only if $s_r \in F$. Let $T_2(\mathcal{A})$ be the set of pairs of tapes accepted by $\mathcal{A}$.

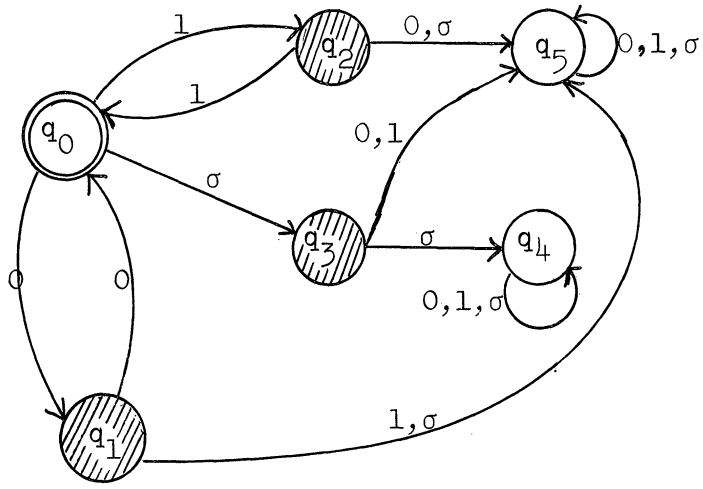
A two-tape automaton may be specified by a state diagram in which those states in C_2 are distinguished by being shaded.

Example 1.

For the following automaton $\mathcal{A}$,

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

and $T_2(\mathcal{A})$ is the set of all pairs (x,x) such that $x \in A_I^*$.



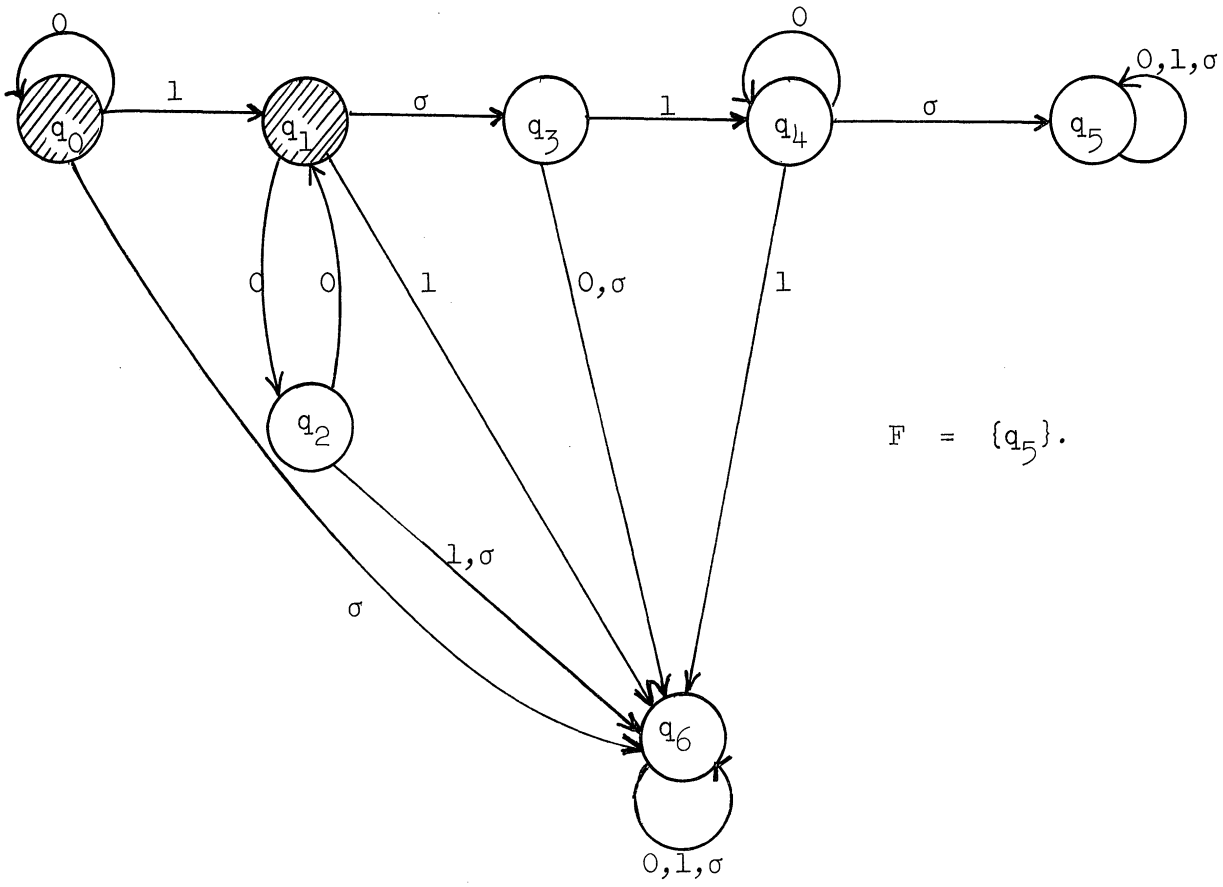
$F = \{q_4\}$

Example 2.

For the following automaton $\mathcal{B}$,

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

and $T_2(\mathcal{B})$ is the set of all pairs $(0^n10^m, 0^\ell 10^n)$, where ℓ, m, and n range over the nonnegative integers.



These examples should indicate the importance of the end marker as a "warning" that the tape has been exhausted.

The first result of Rabin and Scott about two-tape automata establishes that the projection of $T_2(\mathcal{A})$ on either the first or second coordinate is a regular set. Let $E_1(\mathcal{A})$ be the set of all words x such that, for some y, $(x,y) \in T_2(\mathcal{A})$.

Theorem 2

For any two-tape automaton $\mathcal{A}$, $E_1(\mathcal{A})$ is a regular event.

PROOF: For each state q_j, we define the events W_j, X_j, Y_j, and Z_j, each a subset of A_I^*, as follows:

(i) $x \in W_j$ if there exists $y \in A_I^*$ such that for all $t \in A_I^*$ and $u \in A_I^*$ the following implication holds: if $\mathcal{A}$ has the initial instantaneous description $(q_0, xt\sigma, yu\sigma)$, it reaches the instantaneous description $(q_j, t\sigma, u\sigma)$.

(ii) $x \in X_j$ if there exists $y \in A_I^*$ such that, for all u in A_I^*, the following implication holds: if $\mathcal{A}$ has the initial instantaneous description $(q_0, x\sigma, yu\sigma)$, it reaches the instantaneous description $(q_j, e, u\sigma)$.

(iii) $x \in Y_j$ if there exists $y \in A_I^*$ such that, for all t in A_I^*, the following implication holds: if $\mathcal{A}$ has the initial instantaneous description $(q_0, xt\sigma, y\sigma)$, it reaches the instantaneous description $(q_j, t\sigma, e)$.

(iv) $x \in Z_j$ if there exists $y \in A_I^*$ such that the following implication holds: if $\mathcal{A}$ has the initial instantaneous description $(q_0, x\sigma, y\sigma)$, it reaches the instantaneous description (q_j, e, e).

The idea of the proof is to write a system of simultaneous linear event equations from which it will follow, by Theorem 9-2, that each of the events defined above is regular. The desired result will then follow easily. To set up the equations, we need a few more definitions. For $q \in C_1$ and

$$a \in Q \cup \{\sigma\},$$

let $R(q,a)$ be the set of states q_j such that, for some $y \in A_I^*$,

$$\lambda(q,ay) = q_j,$$

and, if z is a prefix of y and is unequal to y,

$$\lambda(q,az) \in C_2.$$

Finally, let the set Q_0 be defined as follows: $q \in Q_0$ if there exists $y \in A_I^*$ such that

$$rp(y) = q$$

and, if z is a prefix of y which is unequal to y,

$$rp(y) \in C_2.$$

With these definitions, we may write the desired equations.

If $q_j \in Q_0$,
$$W_j = e \cup \bigcup_{a \in Q} \bigcup_{\{q_i \mid q_j \in R(q_i,a)\}} W_i \cdot a$$

If $q_j \notin Q_0$,
$$W_j = \bigcup_{a \in Q} \bigcup_{\{q_i \mid q_j \in R(q_i,a)\}} W_i \cdot a$$

$$X_j = \bigcup_{\{q_i \mid q_j \in R(q_j,\sigma)\}} W_i$$

$$Y_j = \bigcup_{\{q_i \mid q_i \in C_2 \text{ and } \lambda(q_i,\sigma) = q_j\}} W_i \cup \bigcup_{a \in Q} \bigcup_{\{q_i \mid q_i \in C_1 \text{ and } \lambda(q_i,a) = q_j\}} Y_i \cdot a$$

$$Z_j = \bigcup_{\substack{\{q_i \mid q_i \in C_1 \text{ and} \\ \lambda(q_i,\sigma) = q_j\}}} Y_i \cup \bigcup_{\substack{\{q_i \mid q_i \in C_2 \text{ and} \\ \lambda(q_i,\sigma) = q_j\}}} X_i$$

It now follows from Theorem 9-2, together with the closure properties of the class of regular events, that each of the events W_j, X_j, Y_j, Z_j, for all j, is regular. To complete the proof, we note that

$$E_1(\mathcal{A}) = \bigcup_{j \in C_1} X_j \cup \bigcup_{j \in C_2} Y_j \cup \bigcup_{j \in Q} Z_j.$$

With Theorem 2 in hand, we can study the closure properties of the class of relations $T_2(\mathcal{A})$ associated with two-tape automata. It is apparent that the class is closed under complementation. For, if

$$\mathcal{A} = (A_I, Q, q_0, F, C_1, C_2, \lambda)$$

and

$$\breve{\mathcal{A}} = (A_I, Q, q_0, Q\text{-}F, C_1, C_2, \lambda),$$

then $T_2(\breve{\mathcal{A}})$ is clearly the complement of $T_2(\mathcal{A})$ in the set $A_I^* \times A_I^*$.

The class of relations definable by two-tape automata is not closed under set intersection, however. Referring to Examples 1 and 2, we have:

$$T_2(\mathcal{A}) = \{(x,x) \mid x \in A_I^*\},$$

$$T_2(\mathcal{B}) = \{(0^n 1\, 0^m,\ 0^\ell 1\, 0^n)\},$$

where ℓ, m, and n range over the nonnegative integers. Then

$$T_2(\mathcal{A}) \cap T_2(\mathcal{B}) = \{(0^n 1\, 0^n,\ 0^n 1\, 0^n)\},$$

where n ranges over the nonnegative integers. But the projection of

$$T_2(\mathcal{A}) \cap T_2(\mathcal{B})$$

on the first coordinate is

$$\{0^n 1 0^n,\ n=1,2,\ldots\},$$

which is easily seen not to be regular. Since the class of relations definable by two-tape automata is closed under complementation, but not intersection, the class cannot be closed under union. For, by the identity

$$\overline{\bar{A} \cup \bar{B}} = A \cap B,$$

closure under union and complementation implies closure under intersection. Summarizing, we have the following result.

Theorem 3

The class of relations definable by two-tape automata is closed under complementation, but not under set union or set intersection.

Since the class of events representable by nondeterministic finite automata is the same as the class of events representable by deterministic finite automata, one might expect that the classes of relations representable by deterministic and nondeterministic two-tape automata would be the same. We shall show that this is not the case. A nondeterministic two-tape automaton is a 7-tuple

$$\mathcal{N} = (A_I, Q, Q_0, F, C_1, C_2, L),$$

where A_I, Q, F, C_1, and C_2 have the same meanings as for two-tape automata; also, $Q_0 \subset Q$ denotes the set of initial states, and L, the transition function, has domain

$$Q \times (A_I \cup \{\sigma\}),$$

and range 2^Q, the set of all subsets of Q. A complete state of $\mathcal{N}$ is a triple

$$(x,y,z) \in Q \times (e \cup A_I^* \sigma) \times (e \cup A_I^* \sigma).$$

If (s,y,z) and (s',y',z') are instantaneous descriptions of $\mathcal{N}$, then (s',y',z') is a successor of (s,y,z) if:

(a) $s \in C_1$, $y = ay'$, where $a \in A_I \cup \{\sigma\}$,
$z' = z$, and $s' \in L(s,a)$,

or

(b) $s \in C_2$, $z = az'$, where $a \in A_I \cup \{\sigma\}$,
$y' = y$, and $s' \in L(s,a)$.

The ordered pair

$$(x_1,x_2) \in A_I^* \times A_I^*$$

is accepted by $\mathcal{N}$ if there is a sequence

$$(s_0,y_0,z_0), (s_1,y_1,z_1), \ldots, (s_r,y_r,z_r)$$

of complete states of $\mathcal{N}$ such that:

(i) $(s_0,y_0,z_0) = (q_0,x_1\sigma,x_2\sigma)$;

(ii) for $i=0,1,\ldots,r-1$, $(s_{i+1},y_{i+1},z_{i+1})$ is a successor of (s_i,y_i,z_i);

(iii) either $s_r \in C_1$ and $y_r = e$, or $s_r \in C_2$ and $z_r = e$;

and

(iv) $s_r \in F$.

Let the set of pairs (x_1,x_2) accepted by $\mathcal{N}$ be denoted $T_2(\mathcal{N})$, and referred to as the relation represented by $\mathcal{N}$.

Theorem 4

Let $\mathcal{N} = (A_I, Q, Q_0, F, C_1, C_2, L)$ and

$\mathcal{N}' = (A_I, Q', Q_0', F', C_1', C_2', L')$

be nondeterministic two-tape automata. Then there exists a nondeterministic two-tape automaton $\mathcal{N}''$ such that

$$T_2(\mathcal{N}) \cup T_2(\mathcal{N}') = T_2(\mathcal{N}'').$$

PROOF: We may assume without loss of generality that $Q \cap Q' = \varphi$. Set

$$\mathcal{N}'' = (A_I, Q \cup Q', Q_0 \cup Q_0', F \cup F', C_1 \cup C_1', C_2 \cup C_2', L''),$$

where

$$L''(q,a) = L(q,a)$$

if $q \in Q$, and

$$L''(q,a) = L'(q,a)$$

if $q \in Q'$. Then

$$T_2(\mathcal{N}) \cup T_2(\mathcal{N}') = T_2(\mathcal{N}'').$$

Let D_2 be the class of relations representable by two-tape automata, and let N_2 be the class of relations representable by nondeterministic two-tape automata.

Corollary 1

D_2 is a proper subset of N_2.

PROOF: Clearly,

$$D_2 \subseteq N_2,$$

since every two-tape automaton is also a nondeterministic two-tape automaton. But

$$D_2 \neq N_2,$$

since N_2 is closed under set union, and D_2 is not.

By a proof similar to the proof of Theorem 2, it can be shown that, for any nondeterministic two-tape automaton $\mathcal{N}$,

$$E_1(\mathcal{N}) = \{x \mid \exists y, (x,y) \in T_2(\mathcal{N})\}$$

is regular. It then follows, by the same argument used in the deterministic case, that N_2 is not closed under intersection. From this it can be inferred that N_2 is not closed under complementation; for, if it were closed under complementation as well as union, N_2 would have to be closed under intersection.

We shall next establish that a certain problem of analysis concerning two-tape automata is recursively unsolvable. In all cases, the strategy we use in proving that a problem R is recursively unsolvable is to show that, if R were solvable, so also would be some other problem P which is known to be recursively unsolvable. The recursively unsolvable P that we shall most frequently resort to in such proofs is the Post correspondence problem.

Theorem 5

There is no algorithm which, given any two-tape automata $\mathcal{A}_1$ and $\mathcal{A}_2$, determines whether $T_2(\mathcal{A}_1) \cap T_2(\mathcal{A}_2)$ is empty.

PROOF: Let

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

and let $a_1,\ldots,a_n$ and $b_1,\ldots,b_n$ be two lists of words in A_I^*. For $i=1,2,\ldots,n$, let $\tilde{i}$ be the sequence consisting of i 1's followed by a zero. Clearly, there exist two-tape automata $\mathcal{A}$ and $\mathcal{B}$ such that

$$T_2(\mathcal{A}) = \{(\tilde{i}_1\tilde{i}_2\ldots\tilde{i}_k,\ a_{i_1}a_{i_2}\ldots a_{i_k})\},$$

and

$$T_2(\mathcal{B}) = \{(\tilde{i}_1\tilde{i}_2\ldots\tilde{i}_k,\ b_{i_1}b_{i_2}\ldots b_{i_k})\},$$

where, in each case, the first member of an ordered pair may be any non-empty sequence obtained by concatenating words from the set $\{\tilde{1},\ \tilde{2},\ \ldots,\ \tilde{n}\}$. To determine whether

$$T_2(\mathcal{A}) \cap T_2(\mathcal{B})$$

is empty is equivalent to solving that instance of the Post correspondence problem given by the lists $a_1,\ldots,a_n$ and $b_1,\ldots,b_n$. Therefore, there is no general decision procedure for the question

$$\text{"is } T_2(\mathcal{A}) \cap T_2(\mathcal{B}) = \varphi\text{?"},$$

since the existence of such a procedure would imply that the Post correspondence problem was recursively solvable.

As a second exercise in the application of the Post correspondence problem, let us consider a question concerning generalized sequential machines. We recall that a generalized sequential machine is specified by a 6-tuple

$$\mathcal{M} = (A_I,\ Q,\ A_o,\ q_0,\ \lambda,\ \delta),$$

all of whose elements, except for δ, have the same interpretation as in Moore machines; the output function δ has domain $Q \times A_I$ and range A_O^*. The function rp(x) is defined exactly as it was for Moore machines, and seq(x), denoting the output sequence generated when an input sequence $x \in A_I^+$ is applied, is defined as follows:

$$\text{if} \quad x = a_1 a_2 \cdots a_n$$

then

$$\begin{aligned} \text{seq}(x) = {} & \delta(q_0,a_1),\ \delta(\text{rp}(a_1),a_2), \\ & \ldots,\ \delta(\text{rp}(a_1, \ldots, a_{n-1}),\ a_n). \end{aligned}$$

Theorem 6

It is undecidable whether, for two generalized sequential machines $\mathcal{H}$ and $\bar{\mathcal{H}}$ with the same input and output alphabets, there exists $x \in A_I^+$ such that

$$\text{seq}_{\mathcal{H}}(x) = \text{seq}_{\bar{\mathcal{H}}}(x).$$

PROOF: Let $\mathcal{H}$ and $\bar{\mathcal{H}}$ be one-state generalized sequential machines, each with the input alphabet $\{1,2,\ldots,n\}$; let q_0 and $\bar{q}_0$ be the (initial) states of $\mathcal{H}$ and $\bar{\mathcal{H}}$, let

$$\delta(q_0,i) = a_i,$$

and

$$\bar{\delta}(\bar{q}_0,i) = b_i,$$

$i=1,2,\ldots,n$. Then

$$\text{seq}_{\mathcal{H}}(i_1,i_2,\ldots,i_k) = a_{i_1} a_{i_2} \cdots a_{i_k},$$

and

$$\text{seq}_{\bar{\mathcal{H}}}(i_1,i_2,\ldots,i_k) = b_{i_1} b_{i_2} \cdots b_{i_k}.$$

Therefore, the problem of deciding whether, for arbitrary one-state generalized sequential machines $\mathcal{H}$ and $\bar{\mathcal{H}}$, there exists x such that

$$\text{seq}_{\mathcal{H}}(x) = \text{seq}_{\bar{\mathcal{H}}}(x),$$

is equivalent to the Post correspondence problem, and is therefore recursively unsolvable.

Two-counter automata

In this section we discuss a class of read-only automata equipped with two counters of unbounded size. The main result, due to Minsky [30], shows that any Turing machine can be simulated by such an automaton. Using the known result that it is undecidable whether a Turing machine with blank initial tape will halt, a corresponding problem concerning two-counter automata is shown to be undecidable. Finally, a class of two-way two-tape automata is defined by "merging" the properties of the two-way and two-tape automata previously considered, and, two problems of analysis concerning these automata are shown to be recursively unsolvable.

A two-counter automaton is equipped with a finite-state control unit, together with two "almost-blank" semi-infinite tapes, denoted θ_1 and θ_2. On each tape, square 0 contains an end marker σ, and each of the other squares contains the "blank" symbol a_0. The machine never writes on its tape, so that this state of affairs is never changed. Since two tapes are used, the set Q of internal states is partitioned into two sets, C_1 and C_2. The quadruples describing the operation of the automaton are of the following types: (q_i,a_j,R,q_ℓ) and (q_i,a_j,L,q_ℓ), where

$$a_j \in \{a_0, \sigma\}.$$

The interpretation of (q_i,a_j,R,q_ℓ), in the case where

$$q_i \in C_\alpha \quad (\alpha \in \{1,2\}),$$

is as follows: if the internal state is q_i and the symbol in the square of θ_α currently being scanned is a_j, then move one square to the right on θ_α, and enter state q_ℓ. To insure that the automaton never goes beyond its end marker, no quadruple of the form (q_i,σ,L,q_ℓ) occurs. Thus, the tapes may be viewed as unbounded counters, each representing a positive integer by the index of the square being scanned.

The complete state of a two-counter automaton is given by a triple (q,m,n), where q is an internal state, and m and n are the indices of the squares being scanned on tapes 1 and 2.

Minsky has described a sense in which any Turing machine T can be simulated by a two-counter automaton W. In discussing this simulation, we make the restriction (for convenience only) that the alphabet of T is $\{0,1\}$. With any complete state (q,x,y) of T, we may associate two non-negative integers, as follows:

$$\ell = \lg(x);$$

if

$$xy = a_0\, a_1 \cdots a_{n-1},$$

then

$$k = \sum_{i=0}^{n-1} a_i \cdot 2^i.$$

For example, the integers associated with the complete state $(q,0110,1010)$ are $\ell=4$, $k=86$. We can now describe the sense in which W simulates T. Let Q be the set of states of T, and Q', the set of states of W. Associated

with the simulation is a one-to-one function h from Q into Q', and a one-to-one function φ from the set of complete states of T into the set of complete states of W, as follows:

$$\varphi[(q,x,y)] = (h(q),2^k \cdot 3^{2^\ell},0).$$

The properties of the simulation are as follows:

(i) if the initial complete state of T is (q_0,x_0,y_0), then the initial complete state of W is $\varphi[(q_0,x_0,y_0)]$;

(ii) if the complete state (q,x,y) of T is followed by (q',x',y') then, if W is in the complete state $\varphi[(q,x,y)]$, it eventually reaches the complete state $\varphi[(q',x',y')]$; however, no uniform bound can be given on the number of steps W requires to pass from $\varphi[(q,x,y)]$ to $\varphi[(q',x',y')]$;

(iii) W halts if and only if T does.

When W is in the complete state $\varphi[(q,x,y)]$ it must first perform a sequence of operations to determine which quadruple of T is to be simulated, and it must then carry out the simulation. Both of these processes are composed of "subroutines" which will be designated $C(p,q)$, where p and q are relatively prime integers. The subroutine $C(p,q)$ may be described as follows: whenever W reaches the instantaneous configuration $(q_\alpha,m,0)$, where

$$m = p^\ell r$$

and $p \nmid r$, then the subroutine $C(p,q)$ produces the final instantaneous configuration $(q_\omega,q^\ell r,0)$. The subroutine operates as follows:

(1) Read θ_1 backwards, beginning at square m; for every p squares covered, move θ_2 to the right once;

(2) If, when θ_1 is exhausted, it is found that m was not a multiple of p, restore θ_1 to its initial condition (using θ_2 to keep count of the number of times p must be added back) and stop;

(3) If it is found that m was a multiple of p, read θ_2 backwards, and move θ_1 forward q squares for every square read on θ_2; return to (1).

All the operations of T can be simulated using a finite number of subroutines. For example, moving to the right on the Turing machine tape leaves k unchanged and increases ℓ. Therefore, if such an operation takes T from (q,x,y) to (q',x',y'), where

$$\varphi[(q,x,y)] = (h(q),q^{2^k}\cdot 3^{2^\ell},0),$$

then

$$\varphi[(q',x',y')] = (h(q'),2^k\cdot 3^{2^{\ell+1}},0).$$

This can be simulated in W by applying C(3,5), followed by C(5,9). Similarly, motion left (except in the special case where x=e) can be simulated by applying C(9,5) first, and then C(5,3). Writing on the tape of T can change the instantaneous configuration in two ways: replacing 0 by 1 in square ℓ, or replacing 1 by 0 in square ℓ. If the former change replaces the configuration (q,x,y) by (q',x',y'), where

$$\varphi[(q,x,y)] = (h(q),2^k\cdot 3^{2^\ell},0),$$

then

$$\varphi[(q',x',y'] = (h(q'),2^{k+2\ell}\cdot 3^{2^\ell},0) = (h(q'),2^k\cdot 6^{2^\ell},0).$$

Therefore, the required change can be effected in W by applying C(3,5), followed by C(5,6). The replacement of 1 by 0 is handled similarly.

We have now shown that each primitive operation on T can be simulated by a sequence of primitive operations of W. To complete our summary of the simulation process, we must show how W determines which operation of T to simulate. Suppose W has the instantaneous description

$$\varphi[q,x,y)] = (h(q),2^k.3^{2^\ell},0).$$

The operation of T is determined by q, together with the symbol on the square being scanned. If the word xy is equal to $c_0,c_1,\ldots,c_{n-1}$, where each symbol c_i is zero or one, then the symbol being scanned is c_ℓ. We shall give a subroutine for W which enables it to determine the value of c_ℓ, and thereby to determine the operation of T which is to be simulated next.

By definition,

$$k = \sum_{i=0}^{n-1} c_i 2^i = 2^\ell\Big(\sum_{i=\ell}^{n-1} c_i 2^{i-\ell}\Big) + \sum_{i=0}^{\ell-1} c_i 2^i.$$

Therefore, the quotient obtained upon dividing k by 2^ℓ is

$$\sum_{i=\ell}^{n-1} c_i 2^{i-\ell}.$$

It is apparent that this quotient is odd if $c_\ell = 1$, and even if $c_\ell = 0$. To determine whether this quotient is even or odd, we subtract 2^ℓ from k as many times as possible, keeping track of the parity of the number of substitutions performed.

The subroutine to perform this process is as follows:

(1) The first step is to make a "spare copy" of k available, to be used eventually in restoring θ_1 to its initial position. Initially, θ_1 is at position $2^k \cdot 3^{2^\ell}$. Apply C(2,35), placing θ_1 at position $5^k \cdot 7^k \cdot 3^{2^\ell}$; then apply C(7,2), obtaining $2^k \cdot 5^k \cdot 3^{2^\ell}$. Go to step 2.

(2) Whenever (2) is reached, the position on θ_1 is of the form $2^k \cdot 5^r \cdot 3^{2^\ell}$. Apply C(15,7); two results are possible:

(a) $2^\ell \leq r$ $\quad 2^k \cdot 5^{r-2^\ell} \cdot 7^{2^\ell}$

(b) $2^\ell > r$ $\quad 2^k \cdot 3^{2^\ell - r} \cdot 7^r$

The occurrence of result (b) can be tested by a simple subroutine which checks whether the position on θ_1 is a multiple of 3, and then restoring θ_1 to its initial position. In case (a), go to (3); in case (b), go to (4).

(3) Whenever (3) is reached, the position on θ_1 is of the form $2^k \cdot 5^s \cdot 7^{2^\ell}$. Apply C(35,3); two results are possible:

(a) $2^\ell \leq s$ $\quad 2^k \cdot 5^{s-2^\ell} \cdot 3^{2^\ell}$

(b) $2^\ell > s$ $\quad 2^k \cdot 7^{2^\ell - s} \cdot 3^s$.

In case (a), go to (2); in case (b), go to (5).

(4) $C_\ell = 0$. To restore θ_1 to its original position, apply C(7,3).

(5) $C_\ell = 1$. To restore θ_1 to its original position, apply C(7,3).

This completes our outline of the simulation of T by W; clearly, the number of primitive steps required by W to simulate one step of T is astronomically large. If W is started with the initial instantaneous description

$$(h(q_0),3,0) \quad = \quad \varphi(q_0,e,a_0)],$$

then W halts if and only if T halts when started with an initially blank tape. Since it is not effectively decidable whether a Turing machine with tape initially blank will halt, it is also not effectively decidable whether a two-counter automaton with initial instantaneous description $(q,3,0)$ will halt.

We shall show that this undecidability result for two-counter automata yields some undecidability results concerning two-way two-tape automata of the type studied by Rabin and Scott [17]. A two-way two-tape automaton is obtained, loosely speaking, by combining the properties of the two-way automata and the two-tape automata that we have studied previously. Such an automaton is specified by an 8-tuple

$$\mathcal{A} \quad = \quad (A_I,\ Q,\ q_0,\ F,\ C_1,\ C_2,\ \lambda,\ D).$$

Each tape has a left end marker σ_1 and a right end marker σ_2; λ is a function

$$\lambda\colon\ Q \times (A_I \cup \{\sigma_1,\sigma_2\}) \to Q,$$

$\{C_1,C_2\}$ is a partition of Q, and D is a function

$$D\colon Q \times (A_I \cup \{\sigma_1,\sigma_2\}) \to \{-1,1\}.$$

If $q \in C_\alpha$, $\alpha \in \{1,2\}$, then, upon entering state q, $\mathcal{A}$ reads tape α. If the symbol being read is a, then $\mathcal{A}$ moves tape α one square in the

direction determined by $D(q,a)$, and enters the state $\lambda(q,a)$. A pair of words

$$(x_1,x_2) \in A_I^* \times A_I^*$$

is accepted by $\mathcal{A}$ if, upon starting in state q_0 with $\sigma_1 x_1 \sigma_2$ on tape 1 and $\sigma_1 x_2 \sigma_2$ on tape 2, scanning the left end marker of each tape, $\mathcal{A}$ eventually stops by entering a state

$$q \in F \cap C_\alpha$$

while scanning the square to the right of σ_2 on tape α, where α may be either zero or one.

Now let W be a two-counter automaton which simulates a Turing machine T. We shall define a two-way two-tape automaton $\tilde{W}$ derived from W; the alphabet A_I for $\tilde{W}$ will contain a single letter 0. The operation of $\tilde{W}$, when presented with

$$(x_1,x_2) = (0^{n_1},0^{n_2})$$

is as follows.

To begin its operation, $\tilde{W}$ moves three squares to the right on tape 1 (corresponding to the fact that W begins its operation scanning square 3 of tape 1). $\tilde{W}$ then simulates W exactly (with σ_1 corresponding to the end marker for W) until one of the following situations occurs:

(a) $\tilde{W}$ reads the symbol σ_2 on tape α. In that case it moves to the right on tape α, and enters a state q not in F; the pair $(0^{n_1},0^{n_2})$ is rejected;

(b) $\tilde{W}$ reaches a configuration corresponding to a "halting configuration" for W; $\tilde{W}$ then moves off one of its tapes to the right,

while remaining in a state $q \in F$; in this case, the pair $(0^{\ell_1},0^{\ell_2})$ is accepted.

Under what conditions is the set of tapes accepted by $\tilde{W}$ empty? If W does not halt, then no tapes are accepted by $\tilde{W}$; $\tilde{W}$ either shuttles its tapes back and forth perpetually or moves off a tape to the right in some state $q \notin F$. On the other hand, suppose W does halt. Then, in the course of its operation, W passes through a finite sequence of complete states:

$$(q_1,m_1,n_1),\ldots,(q_r,m_r,n_r).$$

Let

$$M = \max_i m_i,$$

and

$$N = \max_i n_i.$$

Then, given the tapes $(0^{\ell_1},0^{\ell_2})$, $\tilde{W}$ can faithfully simulate the entire computation of W if and only if $\ell_1 \geq M$ and $\ell_2 \geq N$; therefore $\tilde{W}$ accepts $(0^{\ell_1},0^{\ell_2})$ if and only if W halts,

$$\ell_1 \geq M,$$

and

$$\ell_2 \geq N.$$

The set of tapes accepted by $\tilde{W}$ is nonempty (and, in fact, infinite) if and only if W halts. Since there is no general algorithm to decide whether W halts, we maydraw the following conclusion.

Theorem 7

It is undecidable whether the relation represented by a two-way two-tape automaton with one symbol in its alphabet is empty; it is also undecidable whether the relation represented by such an automaton is infinite.

We have now completed a discussion of the events and relations representable by various automata that are not permitted to write on their tapes. It was found that two-way one-tape automata can do no more than one-way one-tape automata (finite automata). The capabilities of one-way two-tape automata are less clearly understood. We did find, however, that the class of relations representable by nondeterministic automata of this type is larger than the class representable by deterministic automata, and the closure properties of both classes were explored. Using the Post correspondence problem, it was shown that there is no algorithm to determine whether

$$T_2(\mathcal{A}) \cap T_2(\mathcal{B}) = \varphi.$$

It was also shown that two-counter automata can simulate Turing machines, although they seem to do so quite inefficiently. Finally, using our result about two-counter automata, we were able to obtain some undecidability results concerning two-way two-tape automata.

14. GRAMMARS AND LANGUAGES

It is of central interest in automata theory to characterize the classes of sets of words, or n-tuples of words, accepted by various classes of automata. When a class of events is defined naturally in some manner apparently unrelated to automata, it is of interest to determine whether this class is the class of events represented by some interesting class of automata. When such a happy coincidence is discovered, the discovery is a contribution to automata theory, and also confirms the original importance of the class of events.

In this section we study the events defined by certain types of generative grammars. The most general such grammars, sometimes called unrestricted rewriting systems, first arose in computability theory, where they provide one method (Turing machines are another) for specifying the class of recursively enumerable sets. A particular class of such systems, the context-free grammars, will be our main object of study. Context-free grammars suggest themselves in several connections. They provide a formalization of constituent-structure analysis of sentences in natural languages. They also may be used to specify the well-formed formulas in many mathematical and logical systems, and in many computer programming languages. Finally, the study of noncommutative formal power series leads naturally to the introduction of context-free grammars.

Let us introduce generative grammars by means of an example. Suppose we wish to give a finite specification of the set of all well-formed (and fully parenthesized) arithmetic expressions composed of left and right parentheses, the operations +, $\times$, $\div$, -, and the variables x, y, and z.

Such a specification is given by the following set of rewriting rules:

$$S \to (S + S)$$

$$S \to (S \times S)$$

$$S \to (S \div S)$$

$$S \to (- S)$$

$$S \to x$$

$$S \to y$$

$$S \to z.$$

Suppose we start with the string consisting of S alone, and repeatedly replace the left-hand side of some rewriting rule by the corresponding right-hand side, until finally a string is reached in which there are no occurrences of S. The set of all strings that may be derived in this way is precisely the desired set of arithmetic formulas. The following is an example of the derivation of a formula.

$$S$$

$$(S + S)$$

$$(S + (S \div S))$$

$$((-S) + (S \div S))$$

$$((-S) + (x \div S))$$

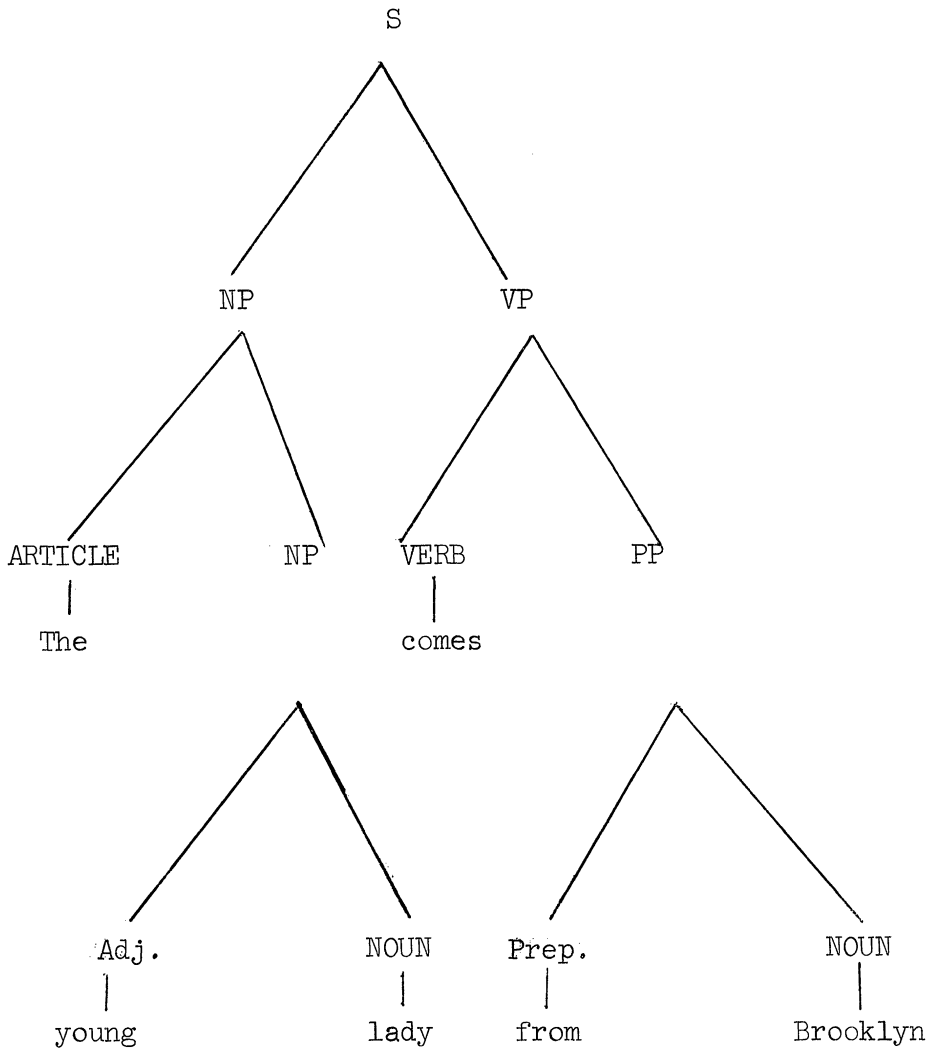
$$((- (S + S)) + (x \div S))$$

$$((- (y + S)) + (x \div S))$$

$$((- (y + x)) + (x \div S))$$

$$((- (y + x)) + (x \div z)).$$

It may be well to indicate how the same kind of rewriting rules that we just illustrated suggest themselves in the analysis of natural languages. Consider the sentence "The young lady comes from Brooklyn." This sentence may be viewed as being composed of the noun phrase "The young lady," and the verb phrase, "comes from Brooklyn"; these may be called the immediate constituents of the sentence. These constituents may then be broken down further, with the complete breakdown shown in the following tree diagram, which gives a structural description of the sentence, with appropriate abbreviations defined below.



S = sentence

NP = noun phrase

VP = verb phrase

PP = prepositional phrase

Adj. = adjective

Prep. = preposition

Alternatively, the sentence may be generated by application of the following rewriting rules:

S → NP VP

VP → Verb PP

NP → Article NP

NP → Adj. Noun

PP → Prep. Noun

Article → The

Adj. → young

Noun → lady

Verb → comes

Prep. → from

Noun → Brooklyn.

An important, and largely unsolved, class of linguistic problems is to find sets of rewriting rules that generate all and only the sentences of particular natural languages, or to show that, within certain constraints, such sets do not exist.

Let us now enter into the formal development. A grammar is a quadruple

$$\mathcal{G} = (V, T, S, P)$$

where V is a finite alphabet called the vocabulary, T is a proper subset of V called the terminal vocabulary, V-T is the nonterminal vocabulary, S is a designated element of V-T called the initial symbol, and P is a set of ordered pairs (x_i, y_i), where $x_i \in V^*$, $y_i \in V^*$, and $x_i \notin T^*$. These ordered pairs are called productions. Sometimes it is convenient to write a production in the form

$$'x_i \rightarrow y_i'.$$

The relation $\rightarrow$ (to be read 'directly yields') over $V^* \times V^*$ is defined as follows: $x \rightarrow y$ if there exists words $u \in V^*$ and $v \in V^*$, and a production (x_i, y_i), such that

$$x = u\, x_i\, v,$$

and

$$y = u\, y_i\, v.$$

The relation $\Longrightarrow$ (to be read 'yields') over $V^* \times V^*$ is the transitive closure of $\rightarrow$. In other words, $x \Longrightarrow y$ if there is a sequence $z_1, z_2, \ldots, z_n$ such that $x = z_1$, $y = z_n$ and

$$z_i \rightarrow z_{i+1}, \qquad i=1,2,\ldots,n-1.$$

Such a sequence is called a derivation or a z_1-derivation of length n. $L(\mathcal{G})$, the language generated by $\mathcal{G}$, is the set of all words x such that $x \in T^*$ and $S \Longrightarrow x$. Thus, grammars generate languages, whereas automata may be viewed either as recognizers of languages or, if they have output tapes, as generators of languages.

Grammars are very similar to semi-Thue systems (Davis [3]), which can represent arbitrary recursively enumerable sets. The following theorem

should therefore not be surprising to those familiar with semi-Thue systems.

Theorem 1

For any recursively enumerable set R, there is a grammar $\mathcal{G}$ such that $R = L(\mathcal{G})$.

PROOF: Since R is recursively enumerable, there is a Turing machine T that halts if and only if its initial complete state is (q_0, e, x), where $x \in R$. Let A be the alphabet of T, and let Q be its set of internal states. The construction of $\mathcal{G}$ is based on the structure of T. The vocabulary V for $\mathcal{G}$ is

$$V = Q \cup A \cup \{S, \sigma, r, t\}.$$

The terminal vocabulary of $\mathcal{G}$ is A, and its initial symbol is S. The productions of $\mathcal{G}$ may be divided according to their function, into three subsets, P_1, P_2, and P_3. The set of productions P_1 enables the derivation from S of every word of the form

$$\sigma\ u\ q\ v\ \sigma$$

such that (q,u,v) is a complete state in which T halts. The productions in P_2 simulate the operation of T running backwards in time. They enable the derivation from every word

$$\sigma\ u\ q\ v\ \sigma$$

corresponding to a final complete state (q,u,v), of all words

$$\sigma\ q_0\ x\ \sigma$$

such that, starting in the complete state (q_0, e, x), T halts in the complete state (q,u,v). Finally, the productions in P_3 generate, from

each word

$$\sigma\, q_0\, x\, \sigma,$$

the word x. The specification of the productions can now be given.

P1

For all $a \in A$,
$$\begin{cases} S \to \sigma r \sigma \\ r \to ra \\ r \to ar. \end{cases}$$

For all pairs q_i, a_j that are not in the first two positions of any quadruple of T,

$$r \to q_i a_j .$$

P2

For each quadruple of the form $q_i a_j a_k q_\ell$,

$$q_\ell\, a_k \to q_i a_j .$$

For each quadruple of the form $q_i a_j R q_\ell$:

(a) for all $a \in A$,

$$a_j q_\ell a \to q_i a_j a;$$

(b) $a_j q_\ell a_o \sigma \to q_i a_j \sigma$.

For each quadruple of the form $q_i a_j L q_\ell$:

(a) for all $a \in A$,

$$q_\ell a a_j \to a q_i a_j;$$

(b) $\sigma q_\ell a_o a_j \to \sigma q_i a_j$.

P3

$\sigma q_0 \rightarrow t$

For all $a \in A$,

$$ta \rightarrow at$$

$t\sigma \rightarrow e$.

It is easily verified that $R = L(\mathcal{H})$.

Theorem 2

For any grammar $\mathcal{H}$, $L(\mathcal{H})$ is recursively enumerable.

PROOF: We give an informal description of a Turing machine T that halts if and only if its initial complete state is (q,e,x), where $x \in L(\mathcal{H})$. T generates the S-derivations of $\mathcal{H}$ in some effectively calculable order; for example, it might generate the derivations in order of increasing length, using a lexicographic rule to order the derivations of the same length. T halts if and only if it obtains a derivation whose final word is S.

Theorem 3

Let $\mathcal{H}$ be a grammar. Then there exists a grammar $\mathcal{H}'$ such that $L(\mathcal{H}') = L(\mathcal{H})$, and such that every production of $\mathcal{H}'$ is of the form (uAv, uyv), where A is a nonterminal symbol, $lg(uAv) \leq 2$, and $lg(uyv) \leq 2$.

PROOF: Every nonterminal (terminal) symbol for $\mathcal{G}$ will also be a nonterminal (terminal) symbol for $\mathcal{G}'$. Additional nonterminal symbols for $\mathcal{G}'$, as well as the productions for $\mathcal{G}'$, are determined from the productions for $\mathcal{G}$. For each production

$$(a_1\ a_2 \cdots a_m,\ b_1\ b_2 \cdots b_n) \text{ of } \mathcal{G},$$

$\mathcal{G}'$ has the new nonterminal symbols $\alpha_1,\alpha_2,\ldots,\alpha_m$, and $\beta_1,\beta_2,\ldots,\beta_r$, where

$$r = \max(m,n).$$

These sets of new symbols are disjoint for the different productions of $\mathcal{G}$. Corresponding to the production

$$(a_1\ a_2 \cdots a_m,\ b_1\ b_2 \cdots b_n) \text{ of } \mathcal{G},$$

$\mathcal{G}'$ has the following productions:

I
$$\begin{cases} a_1 \to \alpha_1 \\ \alpha_1 a_2 \to \alpha_1\alpha_2 \\ \quad\vdots \\ \alpha_{m-2}a_{m-1} \to \alpha_{m-2}\alpha_{m-1} \\ \alpha_{m-1}a_m \to \beta_{m-1}\beta_m \end{cases}$$

II
$$\begin{cases} \alpha_{m-2}\beta_{m-1} \to \beta_{m-2}\beta_{m-1} \\ \quad\vdots \\ \alpha_1\beta_2 \to \beta_1\beta_2 \end{cases}$$

III

if $m < n$	if $m = n$	if $m > n$
$\beta_1 \rightarrow b_1$	$\beta_1 \rightarrow b_1$	$\beta_1 \rightarrow b_1$
$b_1\beta_2 \rightarrow b_1b_2$	$b_1\beta_2 \rightarrow b_1b_2$	$b_1\beta_2 \rightarrow b_1b_2$
$\vdots$	$\vdots$	$\vdots$
$b_{m-1}\beta_m \rightarrow b_m\beta_{m+1}$	$b_{m-1}\beta_m \rightarrow b_{m-1}b_m$	$b_{n-1}\beta_n \rightarrow b_{n-1}b_n$
$\beta_{m+1} \rightarrow b_{m+1}\beta_{m+2}$		$b_n\beta_{n+1} \rightarrow b_n$
$\vdots$		$b_n\beta_{n+2} \rightarrow b_n$
$\beta_{n-1} \rightarrow b_{n-1}\beta_n$		$\vdots$
$\beta_n \rightarrow b_n$		$b_n\beta_m \rightarrow b_n$

Of course, if m=1, the sets I and II given above are replaced by $a_1 \rightarrow \beta_1$. It is easy to verify that

$$L(\mathcal{G}') = L(\mathcal{G}).$$

This completes the proof.

We have shown that the class of languages associated with grammars is precisely the class of recursively enumerable sets. Any class of machines which includes a machine to recognize each language must be as "powerful" as the class of Turing machines; in fact, Turing machines may be considered to perform this task only if rejection of a word is equated with failure to halt. It appears, then, that if we wish to

derive from the study of grammars information about new classes of automata not equivalent to the class of Turing machines, we must introduce restricted types of grammars. These considerations motivate the following definitions. A Type 1 grammar is a grammar in which, for every production (x_i, y_i),

$$lg(x_i) \leq lg(y_i).$$

A context-sensitive grammar is a Type 1 grammar in which every production is of the form (uAv, uyv), where A is a nonterminal symbol.

The following corollary is immediate from the proof of Theorem 3.

Corollary 1

Let $\mathcal{G}$ be a Type 1 grammar. There exists a context-sensitive grammar $\mathcal{G}'$ that generates the same language as $\mathcal{G}$ in which, for each production (x_i, y_i),

$$lg(x_i) \leq lg(y_i) \leq 2.$$

Theorem 4

If $\mathcal{G} = (V, T, P, S)$ is a Type 1 grammar, then $L(\mathcal{G})$ is recursive.

PROOF: Let x be an element of T*. If $x \in L(\mathcal{G})$, then there is an S-derivation of x in which no word is repeated. Moreover, since $\mathcal{G}$ is of Type 1, every word in such a derivation is of length less than or equal to lg(x). Since the number of words $z \in V^*$ such that

$$lg(z) \leq lg(x)$$

is finite, the number of sequences of such words without repetition is finite. An algorithm to determine whether $x \in L(\mathcal{G})$ is to enumerate all

such sequences, and check whether any of them are S-derivations of x. Since there obviously exists a Turing machine that can perform this process, $L(\mathcal{H})$ is recursive.

Corollary 2

There exist languages generated by grammars but not generated by Type 1 grammars.

PROOF: It is known (Davis [3]) that there are recursively enumerable sets that are not recursive.

Theorem 5

There exist recursive sets that are not Type 1 languages.

PROOF: The proof is based on a simple diagonal argument. Let the terminal alphabet T be fixed, and equal to $\{a_0,a_1,\ldots,a_{p-1}\}$. It is easy to construct a 1-1 function f from T* onto the nonnegative integers. One way to do so is as follows: set

$$\theta(a_i) = i, \qquad i=0,1,2,\ldots,p-1.$$

Then

$$f(e) = 0,$$

and

$$f(b_0,b_1,\ldots,b_{n-1}) = \frac{p^n - p}{p-1} + 1 + \sum_{i=0}^{n-1} \theta(b_i)p^i.$$

Clearly, the function $f^{-1}(n)$ can be computed by a Turing machine. On the other hand, if a language over the terminal alphabet $\{a_0,a_1,\ldots,a_{p-1}\}$ is generated by a Type 1 grammar, then it is generated by a Type 1 grammar

in which the nonterminal symbols are chosen from the infinite sequence $S_1, S_2, \ldots, S_n, \ldots$. But, by a device similar to Gödel numbering, we can construct a one-to-many mapping g from the set of all such grammars into the nonnegative integers, as follows:

$$\text{let } \alpha(a_i) = 3 + 2i, \qquad i=0,1,\ldots,p-1$$

$$\text{let } \alpha(\rightarrow) = 3 + 2p$$

$$\text{let } \alpha(S_r) = 3 + 2p + 2r.$$

Any production

$$\rho = x_i \rightarrow y_i$$

can be viewed as a word $b_1 b_2 \ldots b_s$ over the infinite alphabet

$$T \cup \{\rightarrow\} \cup \{S_1, S_2, \ldots\},$$

and its Gödel number is

$$gn(\rho) = \prod_{i=1}^{s} Pr(i)^{\alpha(b_i)}.$$

Then, if $\mathcal{G}$ has productions $\rho_1, \rho_2, \ldots, \rho_\ell$, one of the n! possible values of $g(\mathcal{G})$ is

$$\prod_{i=1}^{\ell} Pr(i)^{gn(\rho_i)}.$$

Clearly, for any integer N, there is at most one grammar $\mathcal{G}$ such that

$$g(\mathcal{G}) = N,$$

and it can be determined by a Turing machine whether such a grammar exists, and if so, what it is. Now, consider the following set:

$$H \subseteq T^*;$$

$x \in H$ if either

(a) there is no grammar $\mathcal{H}$ such that $f(x) = g(\mathcal{H})$

or

(b) there is a unique grammar $\mathcal{H}$ such that $f(x) = g(\mathcal{H})$, and $x \notin L(\mathcal{H})$.

Clearly, H is a recursive set. To decide whether $x \in H$, compute $f(x)$. If the inverse image $g^{-1}(f(x))$ is empty, $x \in H$. If

$$g^{-1}(f(x)) = \{\mathcal{H}\},$$

test whether $x \in L(\mathcal{H})$. If not, $x \in H$. On the other hand, H is not Type 1. For, suppose H were equal to $L(\mathcal{H})$, where $\mathcal{H}$ is Type 1. Consider

$$x = f^{-1}(g(\mathcal{H}));$$

$x \in H$ if and only if $x \notin L(\mathcal{H})$. This completes the proof.

Even though the Type 1 languages form only a proper subclass of the class of recursive sets, there is a sense in which they are closely related to the class of recursively enumerable sets. Consider a grammar

$$\mathcal{H} = (V, T, S, P);$$

$\mathcal{H}$ is not necessarily Type 1. We shall construct a Type 1 grammar $\mathcal{H}'$ such that $L(\mathcal{H}')$ is empty if and only if $L(\mathcal{H})$ is empty. $\mathcal{H}'$ will have the terminal vocabulary $T \cup \{\tau\}$, and nonterminal vocabulary

$$(V - T) \cup \{\sigma\},$$

where σ and τ are new symbols not in V. The productions of $\mathcal{H}'$ are as follows: if $(a_1 \ldots a_m, b_1 \ldots b_n)$ is a production of $\mathcal{H}$, with $m \leq n$, then $(a_1 \ldots a_m, b_1 \ldots b_n)$ is also a production of $\mathcal{H}'$; if $(a_1 \ldots a_m, b_1 \ldots b_n)$ is a production of $\mathcal{H}$, with $m > n$, then $(a_1 \ldots a_m, b_1 \ldots b_n \sigma^{m-n})$ is a production of $\mathcal{H}'$; for $a \in V$, $(\sigma a, a\sigma)$ is a production of $\mathcal{H}'$; finally, (σ, τ) is a production of $\mathcal{H}'$.

For any word

$$y \in (T \cup \{\tau\})^*,$$

let h(y) be the word obtained by deleting all occurrences of τ. Then, if $y \in L(\mathcal{H}')$,

$$h(y) \in L(\mathcal{H});$$

also, if $\bar{y} \in L(\mathcal{H})$, then

$$\bar{y} = h(y),$$

for some word y in $L(\mathcal{H}')$. Therefore, $L(\mathcal{H}')$ is empty if and only if $L(\mathcal{H})$ is empty. Using this fact, we can show that it is undecidable whether $L(\mathcal{H}')$ is empty. We begin by recalling that there is no algorithm to determine, of a given Turing machine T, whether there exists an initial tape that causes T to halt. But, for every Turing machine T we can construct a grammar $\mathcal{H}$ such that $x \in L(\mathcal{H})$ if and only if T halts when it starts in the complete state (q_0, e, x). Therefore there is no algorithm to determine whether $L(\mathcal{H})$ (or $L(\mathcal{H}')$) is empty. Thus, we have established the following result.

Theorem 6

It is undecidable whether the language generated by a Type 1 grammar is empty.

Corollary 3

It is undecidable whether two Type 1 grammars generate the same language.

PROOF: Choose one of the grammars so that its set of productions is empty; the result then follows from Theorem 6.

Later we shall use the Post correspondence problem in the derivation of several undecidability results concerning context-free grammars, which will be studied in the next section.

Having reached some understanding of Type 1 languages, we now undertake to characterize the class of automata that recognize such languages. In proving Theorem 1, we showed how to construct, for a given Turing machine T, a grammar $\mathcal{G}$ that generates the set of words accepted by T; that is, T halts if and only if its initial tape configuration is an element of $L(\mathcal{G})$. Inspecting this construction to determine why $\mathcal{G}$ fails to be Type 1, we find that the productions of $\mathcal{G}$ that do not meet the requirements of a Type 1 grammar include the following: for each quadruple of the form $q_i a_j R q_\ell$, the production

$$a_j q_\ell a_0 \sigma \rightarrow q_i a_j \sigma;$$

for each quadruple of the form $q_i a_j L q_\ell$, the production

$$\sigma q_\ell a_0 a_j \rightarrow \sigma q_i a_j .$$

Both these productions occur because of the possibility that T may read blank squares of tape that did not initially contain data; one might expect that, if an automaton could be defined that had all the capabilities of T except the ability to "grow new tape", then the events recognized by such automata might turn out to be the Type 1 languages. Landweber [28] and Kuroda [27] have shown that this is indeed the case; the desired automata are the nondeterministic linear bounded automata.

A (nondeterministic) linear bounded automaton (l.b.a.) $\mathcal{A}$ is specified by its alphabet A, its set Q of internal states, its set of initial states Q_0, its set of final states F, and its commands, in the form of

Turing machine quadruples. The set of symbols occurring on a tape is

$$A \cup \{\sigma_1, \sigma_2\},$$

where σ_1 and σ_2 are, respectively, left and right end markers. Among the quadruples of the form $q_i a_j a_k, q_\ell$, the following conditions must be met: if

$$a_j = \sigma_1,$$

then

$$a_k = \sigma_1;$$

if

$$a_j = \sigma_2,$$

then

$$a_k = \sigma_2;$$

if $a_j \in A$, then $a_k \in A$. In other words, end markers are neither created nor destroyed. Since $\mathcal{A}$ is nondeterministic, there is no restriction on the number of quadruples having a given pair (q_i, a_j) in the first two positions.

The l.b.a. $\mathcal{A}$ begins its operation in some state $q \in Q_0$, with a word of the form $\sigma_1 x \sigma_2$ on its tape, where $x \in A^*$, and with the square containing σ_1 being scanned. At each step, $\mathcal{A}$ executes one of the quadruples associated with its internal state q_i, and a_j, the symbol on the square being scanned. Thus $\mathcal{A}$ operates in the same manner as a (nondeterministic) Turing machine, except that it halts upon entering that portion of the tape beyond its end markers; in other words, $\mathcal{A}$ halts upon executing a command of the form $q_i \sigma_2 R q_\ell$ or $q_i \sigma_1 L q_\ell$. Thus, the

operation of $\mathcal{A}$ is similar to that of a two-way one-tape read-only automaton with end markers, except that $\mathcal{A}$ is capable of writing on its tape. The word x is accepted by $\mathcal{A}$ if, starting with the initial tape configuration $\sigma_1 x \sigma_2$, $\mathcal{A}$ has some sequence of operation such that it eventually runs off its tape to the right in some state $q_i \in F$; in such a sequence, the final quadruple is $q_i \sigma_2 R q_\ell$. Sequences in which $\mathcal{A}$ fails to terminate, runs off its tape to the left, stops without running off the tape, or runs off the tape to the right in a state belonging to Q-F, do not contribute to the acceptance of any words.

The following theorem is a variant of a theorem due to Landweber [28] whose definition of 'linear bounded automaton' differs slightly from ours.

Theorem 7

Let B be the set of nonempty words accepted by a nondeterministic linear bounded automaton $\mathcal{A}$. Then there exists a Type 1 grammar such that $B = L(\mathcal{G})$.

PROOF: To clarify the idea behind the proof, we first give a construction of a grammar $\mathcal{G}$ that does not quite satisfy the Type 1 restriction, and then modify it. The first construction is similar to the construction used in showing that every recursively enumerable set is a language. In making this construction, we think of a complete state of $\mathcal{A}$ as represented by a word $\sigma_1 x q_i y \sigma_2$, where $xy \in A^*$. This word indicates that the tape contains the word $\sigma_1 xy\sigma_2$, that the internal state is q_i, and that the square containing the first symbol of $y\sigma_2$ is being scanned.

The productions of $\mathcal{G}$ may be partitioned into the three subsets P_1, P_2, and P_3. P_1 generates all those complete states that can occur

immediately prior to acceptance of a nonempty word; i.e., all sequences of the form $\sigma_1 x q_i \sigma_2$ such that $x \in A^+$ and $\mathcal{A}$ has a quadruple of the form $(q_i, \sigma_2, R, q_\ell)$, where $q_\ell \in F$. P_2 then "runs $\mathcal{A}$ backwards", and P_3 takes all words of the form $q_i \sigma_1 x \sigma_2$ such that $q_i \in Q_0$, and removes the symbols q_i, σ_1, and σ_2. It is in this final phase that the Type 1 condition is violated. The productions of $\mathcal{H}$ are as follows.

P1

$$S \rightarrow \sigma_1 \; r \; \sigma_2$$

For all $a \in A$,

$$r \rightarrow ar.$$

For all $a \in A$, and all q_i such that $\mathcal{A}$ has a quadruple $q_i \sigma_2 R q_\ell$, where $q_\ell \in F$,

$$r \rightarrow a \; q_i.$$

P2

For each quadruple of the form $q_i a_j a_k q_\ell$,

$$q_\ell a_k \rightarrow q_i a_j.$$

For each quadruple of the form $q_i a_j R q_\ell$, $\quad a_j \neq \sigma_2$,

$$a_j q_\ell \rightarrow q_i a_j.$$

For each quadruple of the form $q_i a_j L q_\ell$, $\quad a_j \neq \sigma_1$,

and for all $a \in A$,

$$q_\ell \; a \; a_j \rightarrow a \; q_i \; a_j.$$

P3

For all $q_i \in Q_0$, and for all $a \in A$,

$$q_i \sigma_1 a \rightarrow a.$$

For all $a \in A$,

$$\sigma_2 \rightarrow a.$$

We omit the proof that this grammar generates the set B. In supplying the proof, one may not presuppose that the productions in P_1 are applied before those in P_2, and the productions in P_2, before those in P_3.

Because of the productions in P_3, $\mathcal{G}$ fails to be Type 1. An equivalent Type 1 grammar can be obtained, however. Note that, in each production of $\mathcal{G}$, the number of occurrences of elements of A on the left-hand side is less than or equal to the number on the right-hand side. Therefore, if $u \rightarrow v$ in $\mathcal{G}$, then u has no more occurrences of elements of A than v does. Suppose, then, that u and v are "factored" into subsequences, each containing only one occurrence of an element of A, and the same is done for v; then u will contain no more subsequences than v does. Moreover, only a finite number of distinct subsequences are required in all of the factorizations. This suggests the construction of a new grammar $\mathcal{G}'$ with a nonterminal element serving as a "coded representation" of each required subsequence. For clarity, we use the following mnemonic device: if x is one of the required subsequences, then (x) denotes the corresponding nonterminal symbols of $\mathcal{G}'$. The following is a list of the new nonterminal symbols:

For all $a \in A$, for all $q \in Q$:

$$(q\ \sigma_1\ a\ \sigma_2) \qquad (q,\ \sigma_1\ a) \qquad (a\ \sigma_R)$$
$$(\sigma_1\ q\ a\ \sigma_2) \qquad (\sigma_1\ q\ a) \qquad (\sigma_1\ a)$$
$$(\sigma_1\ a\ q\ \sigma_2) \qquad (q\ a\ \sigma_2) \qquad (a\ q)$$
$$(\sigma_1\ a\ q) \qquad (q\ a)$$
$$(a\ q\ \sigma_2)$$

We shall not give the full details of the modification of the grammar given above, using these new methods. As an illustration, however, we note that the set of productions P3 is replaced by the following set of productions P'3:

P'3

For all $a \in A$, for all $q_i \in Q_0$

$$(q_i\ \sigma_1\ a\ \sigma_2) \rightarrow a;$$
$$(q_i\ \sigma_1\ a) \rightarrow a;$$
$$(a\ \sigma_2) \rightarrow a.$$

As a second illustration, we remark that each quadruple of the form $(q_i a_j R q_\ell)$, $a_j \in A$, gives rise to the following productions, in which 'a' is understood to range over the elements of A.

$$(\sigma_1 a_j q_\ell \sigma_2) \rightarrow (\sigma_1 q_i a_j \sigma_2)$$
$$(\sigma_1 a_j)(q_\ell a \sigma_2) \rightarrow (\sigma_1 q_i a_j)(a \sigma_2)$$
$$(\sigma_1 a_j q_\ell) \rightarrow (\sigma_1 q_i a_j)$$
$$(\sigma_1 a_j)(q_\ell a) \rightarrow (\sigma_1 q_i a_j) a$$
$$(q_\ell a_k \sigma_2) \rightarrow (q_i a_j \sigma_2)$$
$$(\sigma_1 a q_\ell)(a_k \sigma_2) \rightarrow (\sigma_1 a q_i)(a_j \sigma_j)$$
$$(q_\ell a_k) \rightarrow (q_i a_j)$$

$$(\sigma_1 a q_\ell) a_k \rightarrow (\sigma_1 a q_i) a_j$$
$$(q_\ell a_k \sigma_2) \rightarrow (q_i a_j \sigma_2)$$
$$(a q_\ell)(a_k \sigma_2) \rightarrow (a\ q_i)(a_j \sigma_2)$$
$$(a\ q_\ell) a_k \rightarrow (a\ q_i) a_j$$

The truth of the converse of Theorem 7 is quite apparent. It was first pointed out by Kuroda [27].

Theorem 8

Let $\mathcal{H}$ = (V, T, S, P) be a Type 1 grammar. Then there exists a linear bounded automaton $\mathcal{A}$ for which the set of words accepted is L($\mathcal{H}$).

In specifying the l.b.a. $\mathcal{A}$, we do not require that the alphabet of $\mathcal{A}$ be restricted to the terminal vocabulary of $\mathcal{H}$; A finite number of additional symbols may be present in the alphabet of $\mathcal{A}$; these symbols may be written in the course of the calculation, as $\mathcal{A}$ uses its tape for "scratchwork".

PROOF (of Theorem 8): We shall describe informally how $\mathcal{A}$ determines whether a word x is an element of L($\mathcal{H}$). The alphabet of $\mathcal{A}$ is

$$V \cup \{\varphi\},$$

where $\varphi \notin V$. The symbol φ serves as a "blank". Initially, $\mathcal{A}$ reads its tape from left to right, checking that all symbols between the end markers are terminal, and that the end markers are not adjacent. $\mathcal{A}$ then moves back to the left end marker σ_1, and enters the "search" mode. Again $\mathcal{A}$ scans the tape from left to right. So long as no symbol read is the first symbol of the right-hand side of a production, the left-to-

right scan is continued. Suppose, however, that a symbol is encountered that is the first symbol of the right-hand sides of the productions

$$(x_{i_1},y_{i_1}),\ (x_{i_2},y_{i_2}),\ \ldots,\ (x_{i_r},y_{i_r}).$$

Then $\mathcal{A}$ has r+1 possible courses of action (remember that $\mathcal{A}$ is non-deterministic!). $\mathcal{A}$ may either continue searching, or it may enter a state q_{i_j} which corresponds to the "guess" that the symbol being scanned is the first symbol of y_{i_j}. Suppose

$$(x_{i_j},y_{i_j}) = (a_1,\ldots,a_m,\ b_1,\ldots,b_n).$$

When the state q_{i_j} is entered, the left-to-right scan is continued, to determine whether the symbols $b_1,b_2,\ldots,b_n$ occur successively. (The answer is automatically "yes" if

$$\lg(y_{i_j}) = 1.)$$

In this process, occurrences of the symbol φ are ignored. If the answer is "no", then $\mathcal{A}$ moves off the right end of the tape in a non-final state. If the answer is "yes", then a_j is written in place of b_j, $j=1,2,\ldots,m$, and φ is written in place of b_j, $j=m+1,\ldots,n$. Since $\mathcal{G}$ is Type 1, this is always possible:

$$\lg(x_{i_j}) \leq \lg(y_{i_j}).$$

$\mathcal{A}$ then moves back to the left-hand end marker, and reads the tape from left to right to determine whether the word between end markers is of the form $\varphi^p S \varphi^q$; if so, $\mathcal{A}$ moves off the right end of the tape in some state $q \in F$. If not, $\mathcal{A}$ returns to the left end of the tape, and reenters the search mode. It is easy to check that $\mathcal{A}$ accepts x if and only if there is an S-derivation of x in $\mathcal{G}$.

Theorems 7 and 8 establish that a language is Type 1 if and only if it is the set of all nonempty words accepted by some nondeterministic linear bounded automaton. From this, one can easily see that the following events are Type 1 languages:

(1) the set of all words with more zeros than ones;

(2) the set of all nonempty words whose length is a square;

(3) the set of all words whose length is a prime;

(4) the set of all words of the form $a^k b^k c^k$, $k=1,2,\ldots,n$.

Deterministic linear bounded automata suffice for the constructions required in these examples. It is not known whether there exists events representable by nondeterministic linear bounded automata, but not by deterministic linear bounded automata.

Myhill [31] has obtained a result which, when taken together with Theorem 7, shows that the Type 1 languages form a very substantial subclass of the class of recursive sets. His result is that the class of events represented by deterministic linear bounded automata includes the class of rudimentary events. The rudimentary events (cf. R. M. Smullyan, "Theory of Formal Systems") are those that can be expressed in a vocabulary consisting of:

(i) the alphabet A;

(ii) the three place predicate uv = w, with the interpretation "the word w is the concatenation of u and v";

(iii) the Boolean connectives 'and', 'or', and 'not';

(iv) the bounded existential quantifier $\exists\, v \leq w$, with the interpretation "there exists a word v such that $lg(v) \leq lg(w)$".

For example, the set of all words of the form xx is expressed by the formula

$$\exists\, x \leq y\ (xx = y).$$

It can be shown then, in any grammar, the set of all derivations is a rudimentary event, and therefore a Type 1 language. Similarly, the set of proofs in any of a large class of formal systems is rudimentary, and therefore Type 1.

Theorems 7 and 8 may also be used in studying the closure properties of the class of Type 1 languages.

Theorem 9

The intersection of two Type 1 languages is a Type 1 language.

PROOF: Let L_1 and L_2 be Type 1 languages over the alphabet T. By Theorem 8, there exist nondeterministic l.b.a.'s $\mathcal{A}_1$ and $\mathcal{A}_2$ that represent the events L_1 and L_2, respectively. Let the vocabularies of these l.b.a.'s be V_1 and V_2; of course,

$$T \subseteq V_1 \cap V_2.$$

We shall specify a nondeterministic l.b.a. $\mathcal{A}_3$ that represents the event $L_1 \cap L_2$. The alphabet of $\mathcal{A}_3$ is

$$(V_1 \times V_2) \cup T;$$

the operation of $\mathcal{A}_3$ is as follows. First, $\mathcal{A}_3$ reads its tape from left ro right, checking that each symbol encountered is an element of T, and replacing each symbol a by (a,a). Throughout the rest of the operation of $\mathcal{A}_3$, all symbols on the tape are elements of $V_1 \times V_2$. We may

view the tape, therefore, as having two parallel "channels", one for the first components of ordered pairs, and the other for the second components. After its initial reading of the tape, a_3 resets the tape to its initial position, and simulates a_1, using only the first channel (first component of each ordered pair); however, at the point where a_1 would run off the tape to the right in a final state, a_3 returns to the left-hand end marker and simulates a_2, using the second channel. Clearly, a word is accepted by a_3 if and only if it is an element of $L_1 \cap L_2$. Therefore, by Theorem 7, $L_1 \cap L_2$ is a Type 1 language.

Some of the other closure properties of the Type 1 languages are quite easy to prove. We derive them by means of a sequence of lemmas.

Lemma 1

For any Type 1 grammar $G = (V, T, S, P)$, there is a Type 1 grammar G' in which no terminal symbol occurs on the left-hand side of a production, such that $L(G') = L(G)$.

PROOF: Let T' be a set of symbols not in V, in one-to-one correspondence with T. Let α be a homomorphism from V* into $((V-T) \cup T')^*$, defined as follows: if $A \in V-T$,

$$\alpha(A) = A$$

if $a \in T$, $\alpha(a)$ is the corresponding element of T'. Then

$$G' = (V \cup T', T, S, P'),$$

where the set of productions P' is defined as follows:

$$P' = \{(\alpha(x_i),\alpha(y_i)) \mid (x_i,y_i) \in P\} \cup \{(\alpha(a),a) \mid a \in T\}.$$

It is now clear that $L(\mathcal{H}') = L(\mathcal{H})$.

Let L_1 and L_2 be Type 1 languages. With no loss of generality we may assume that

$$L_1 = L(\mathcal{H}_1)$$

and

$$L_2 = L(\mathcal{H}_2),$$

where

$$\mathcal{H}_1 = (V_1, T_1, S_1, P_1),$$
$$\mathcal{H}_2 = (V_2, T_2, S_2, P_2),$$
$$(V_1 - T_1) \cap (V_2 - T_2) = \varphi,$$

and neither $\mathcal{H}_1$ nor $\mathcal{H}_2$ has a terminal symbol on the left-hand side of any production.

Lemma 2

$L_1 \cup L_2$ is a Type 1 language.

PROOF: Let S be a symbol not in $V_1 \cup V_2$. Then $L_1 \cup L_2$ is generated by the grammar

$$(V_1 \cup V_2 \cup \{S\},\ T_1 \cup T_2,\ S,\ P_1 \cup P_2 \cup \{(S,S_1),\ (S,S_2)\}).$$

Lemma 3

$\tilde{L}_1 = \{\tilde{x} \mid x \in L\}$ is a Type 1 language.

PROOF: $\tilde{L}_1$ is generated by the grammar

$$(V_1,\ T_1,\ S_1,\ \{(\tilde{x}_i, \tilde{y}_i) \mid (x_i, y_i) \in P\}).$$

Lemma 4

$L_1 \cdot L_2$ is a Type 1 language.

PROOF: Let S be a symbol not in $V_1 \cup V_2$. Then $L_1 \cdot L_2$ is generated by the grammar

$$(V_1 \cup V_2 \cup \{S\},\ T_1 \cup T_2,\ S,\ P_1 \cup P_2 \cup \{(S,\ S_1\ S_2)\}).$$

Lemma 5

$L_1^\dagger$ is a Type 1 language.

PROOF: With no loss of generality, we may assume that S_1 does not occur on the right-hand side of a production. Let

$$\mathcal{H}_3 = (V_3,\ T_1,\ S_3,\ P_3)$$

be a grammar that generates L_1, obtained from $\mathcal{H}_1$ by renaming each element of $(V_1 - T_1)$. Then

$$|V_1 - T_1| = |V_3 - T_1|,$$

and

$$V_1 \cap V_3 = T_1.$$

Let S and S' be symbols not in $V_1 \cup V_3$. Then the following grammar generates $L_1^\dagger$.

$$(V_1 \cup V_3 \cup \{S,S'\},\ T_1,\ S,\ P_1 \cup P_3 \cup \{(S,SS'),\ (S',S\ S),\ (S,S_1),\ (S',S_3)\}).$$

It is not known whether the class of Type 1 languages is closed under complementation. Kuroda [27] has shown, however, that the class of events represented by deterministic linear bounded automata is a Boolean algebra of sets. From this we can draw the following inference: if every event representable by a nondeterministic l.b.a. is representable by a deterministic l.b.a., then the Type 1 languages are closed under complementation.

Context-free grammars and languages

A grammar

$$\mathcal{G} = (V, T, S, P)$$

is a context-free grammar if each of its productions is of the form (A,y), where A ϵ V-T. Since there is no requirement that y $\neq$ e, there are context-free grammars that are not Type 1. If the additional requirement y $\neq$ e is imposed, a context-free grammar is called a 1-context-free grammar. Every 1-context-free grammar is a Type 1 grammar. The language generated by a context-free grammar is called a context-free language.

The grammars that were used as examples at the beginning of Chapter 14 are both context-free. The properties of context-free grammars and languages have considerable relevance to the study of natural languages, as well as artificial languages used in programming digital computers. Although natural languages appear not to be context-free (In fact, Postal [35] has proved this for the Mohawk language.), reasonable approximations to natural languages can be generated by context-free grammars. Oettinger [31] and Yngve [38] have made use of this fact in their studies of automatic translation. Also, large portions of ALGOL and other compiler languages for digital computers have been specified by context-free grammars.

Our discussion of context-free grammars and languages is based, to a considerable extent, on Bar-Hillel,Perles, and Shamir [20], Landweber [29], and Chomsky [21, 22, 23].

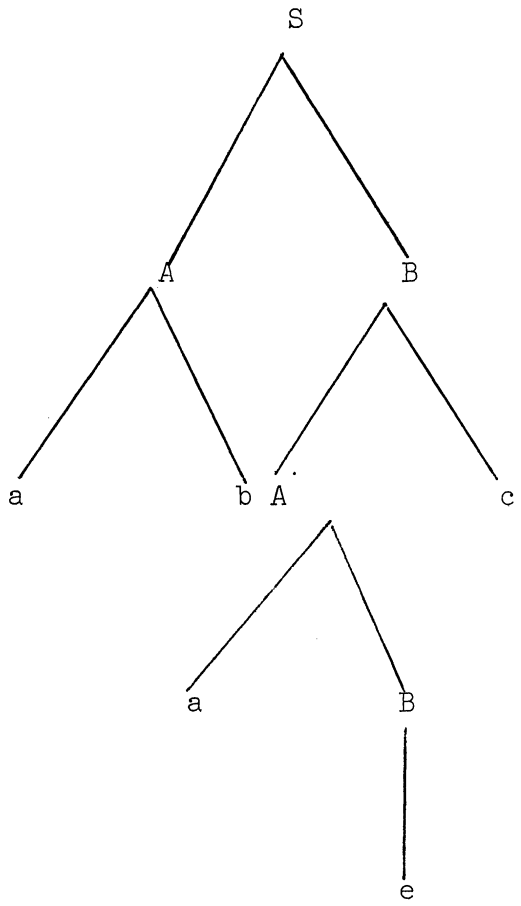
In thinking about properties of context-free languages, it is useful to bear in mind a natural method of representing a derivation in a context-free grammar by a tree. An example should suffice to explain this representation.

Example 3

Consider the following S-derivation of the word abac.

Word	Production used in obtaining word
S	
AB	S → AB
abB	A → ab
abAc	B → Ac
abaBc	A → aB
abac	B → e.

The tree representing this S-derivation is as follows:



The length of a longest path from the root to a leaf in a tree representing a derivation is called the depth of the derivation.

Theorem 10

Every regular event is a context-free language.

PROOF: Let R be the regular event represented by the finite automaton

$$\mathcal{A} = (A_I, Q, q_0, F, \lambda).$$

The context-free grammar,

$$\mathcal{a} = (A_I \cup Q, A_I, q_0, P)$$

may easily be seen to generate R:

For all $(q,a) \in Q \times A_I$,

$$q \rightarrow a\, \lambda(q,a).$$

For all $q \in F$,

$$q \rightarrow e.$$

Bar-Hillel, Perles, and Shamir [20] have shown that, if $\mathcal{a}$ is a one-way two-tape read-only automaton, and if σ is a symbol not in the alphabet of $\mathcal{a}$, then

$$\{x\, \sigma\, \tilde{y} \mid (x,y) \in T_2(\mathcal{a})\}$$

is a context-free language. We shall obtain this result quite easily later, once we have defined the class of automata that accept all, and only, the context-free languages. Using the methods of proof employed in Lemmas 1, 2, 3, and 4, the following closure properties of the context-free languages are easily obtained.

Theorem 11

The class of context-free languages is closed under the unary operations $\dagger$, $*$, and $\sim$, and under the binary operations $\cup$ and $\cdot$.

PROOF: Let L_1 and L_2, respectively, be generated by the context-free grammars

$$\mathcal{G}_1 = (V_1, T_1, S_1, P_1)$$

and

$$\mathcal{G}_2 = (V_2, T_2, S_2, P_2).$$

With no loss of generality, we may assume that the nonterminal vocabularies of $\mathcal{G}_1$ and $\mathcal{G}_2$ are disjoint, and that neither S_1 nor S_2 occurs on the right-hand side of a production. Then $L_1^{\dagger}$ is generated by the grammar

$$\mathcal{G}_1^{\dagger} = (V_1, T_1, S_1, P_1 \cup \{(S_1, S_1S_1)\}).$$

L_1^*, by the grammar

$$\mathcal{G}_1^* = (V_1, T_1, S_1, P_1 \cup \{(S_1, S_1S_1), (S_1, e)\}),$$

and $\tilde{L}_1$, by the grammar

$$\mathcal{G}_1^{\sim} = (V_1, T_1, S_1, \tilde{P}_1)$$

where

$$\tilde{P}_1 = \{(A, \tilde{y}) \mid (A, y) \in P_1\}.$$

Also, $L_1 \cup L_2$ is generated by the grammar

$$(V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{(S, S_1), (S, S_2)\},$$

where $S \notin V_1 \cup V_2$, and $L_1 \cdot L_2$ is generated by the grammar

$$(V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{(S, S_1S_2)\}.$$

We omit the proofs that these grammars have the properties that are claimed.

It turns out that the class of context-free languages is not closed under set intersection or complementation. In order to prove this, we must first develop a technique for proving that certain events are not context-free languages; the development of this technique will require some preliminary results.

Lemma 6

It is effectively decidable whether the null sequence is in the language generated by a context-free grammar $\mathcal{G} = (V, T, S, P)$.

PROOF: Let the following sequence of subsets of V-T be defined

$$V_1 = \{A \mid A \rightarrow e\}$$

for k=1,2,...

$$V_{k+1} = V_k \cup \{A \mid \text{there exists } (A,y) \in P \text{ such that } y \in V_k^*\}.$$

Thus, $A \in V_k$ if and only if there is an A-derivation of e of depth less than or equal to k. Therefore

$$V_1 \subseteq V_2 \cdots \subseteq V_k \subseteq V_{k+1};$$

also, if $V_k = V_{k+1}$, then $V_k = V_r$ for any r greater than or equal to k. Therefore, if

$$|V\text{-}T| = n,$$

then $e \in L(\mathcal{G})$ if and only if $S \in V_n$. The required decision procedure, then, is to construct V_n and determine whether $S \in V_n$.

Theorem 12

Let $\mathcal{H}$ = (V, T, S, P) be a context-free grammar. Then one can construct a 1-context-free grammar $\mathcal{H}'$ such that $L(\mathcal{H}') = L(\mathcal{H}) \cap T^{\dagger}$.

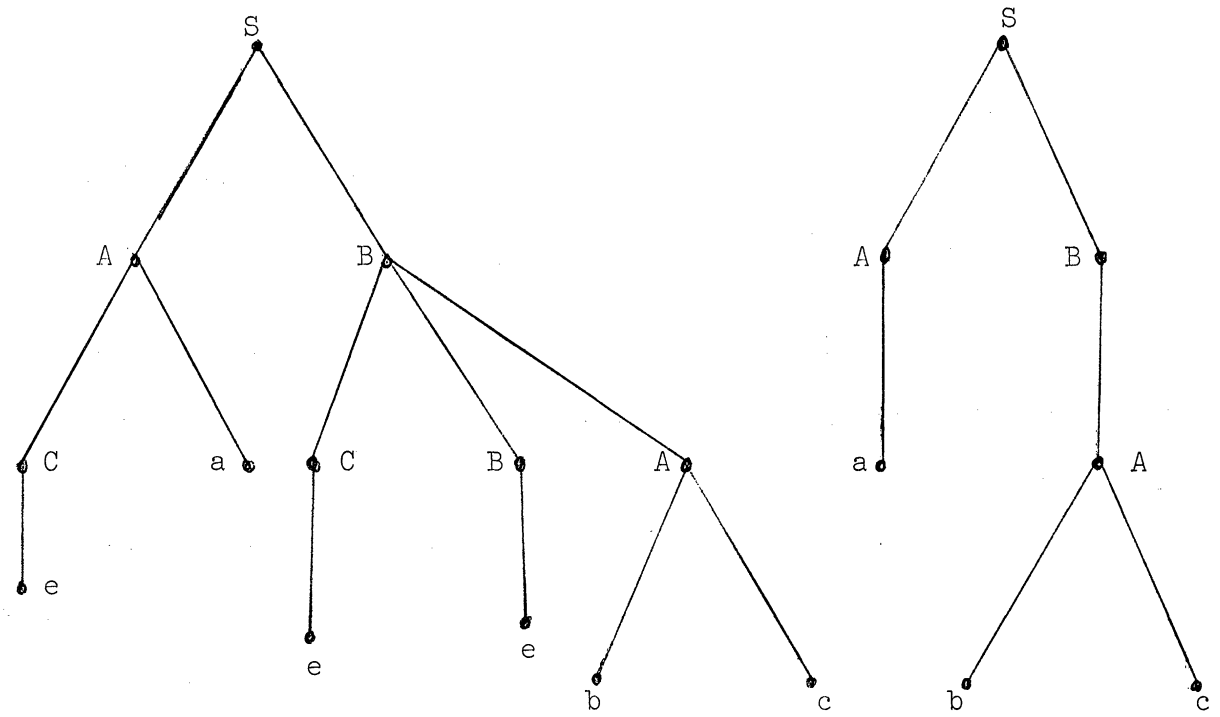
PROOF: There is nothing to prove unless the set V_n defined in the proof of Lemma 6 is nonempty. If V_n is nonempty, the productions of $\mathcal{H}'$ are all the pairs (A,z) such that $z \neq e$, and there is some element $(A,y) \in P$ such that $z = y$ or z can be obtained by deleting from y some occurrences of elements of V_n. Clearly, $\mathcal{H}'$ is a 1-context-free grammar. Also, if

$$S \Longrightarrow x \text{ in } \mathcal{H} \text{ and } x \neq e,$$

then

$$S \Longrightarrow x \text{ in } \mathcal{H}'.$$

To see this, consider the tree T specifying an S-derivation of x in $\mathcal{H}$. Associated with each internal node n of T is a subtree T(n) containing all the nodes connected to the root through n. Delete from T every leaf labelled e, every node n such that all the leaves of T(n) are labelled e, and every branch connected to a deleted node. Then the resulting tree T' represents an S-derivation of x in $\mathcal{H}'$. The following diagram illustrates the construction of T' from T.



T T'

Conversely, to every S-derivation of x in $\mathcal{G}'$, there corresponds an S-derivation of x in $\mathcal{G}$. Every time a production (A,z) is used in the former derivation, where (A,z) is obtained from some production (A,y) of $\mathcal{G}$, then the latter derivation uses (A,y), followed by additional productions to replace by the null sequence each of the nonterminal elements deleted in passing from y to z.

Corollary 4

Every context-free language is a recursive set.

PROOF: Every 1-context-free language is Type 1, and therefore recursive. Taking this together with Lemma 6 and Theorem 12, one obtains the desired conclusion.

Let us define a 2-context-free grammar as a contest-free grammar in which, for every production (A,y), $lg(y) \geq 2$.

Theorem 13

Let $\mathcal{G}$ = (V, T, S, P) be a 1-context-free grammar. Then one can construct a 2-context-free grammar $\mathcal{G}'$ such that $x \in L(\mathcal{G}')$ if and only if $x \in L(\mathcal{G})$ and $lg(x) \geq 2$.

PROOF: For each $A \in V-T$, let the following sequence of sets be defined:

$$W_1(A) = \{A\}$$

for $k=1,2,\ldots$

$$W_{k+1}(A) = W_k(A) \cup \{B \mid (B \in V \text{ and } \exists\, x \mid x \in W_k(A) \text{ and } (x,B) \in P)\}.$$

Thus, for any $B \in V$, $B \in W_{k+1}(A)$ if and only if there is an A-derivation of B of length less than or equal to k. It also follows that

$$W_1(A) \subseteq W_2(A) \quad \ldots \subseteq W_k(A) \subseteq \ldots,$$

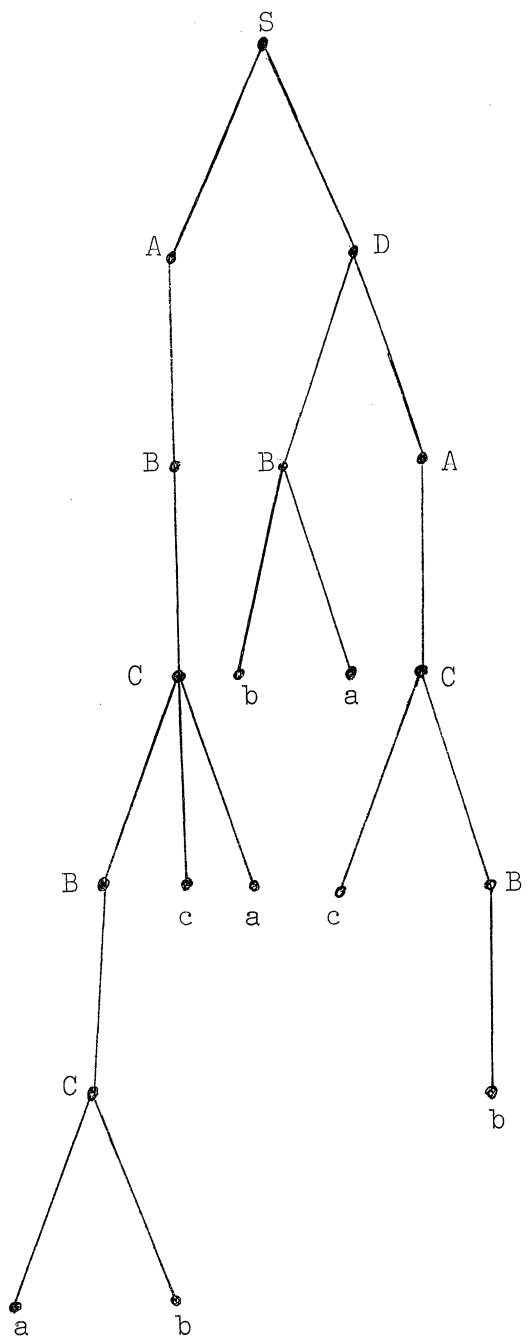
and that, if

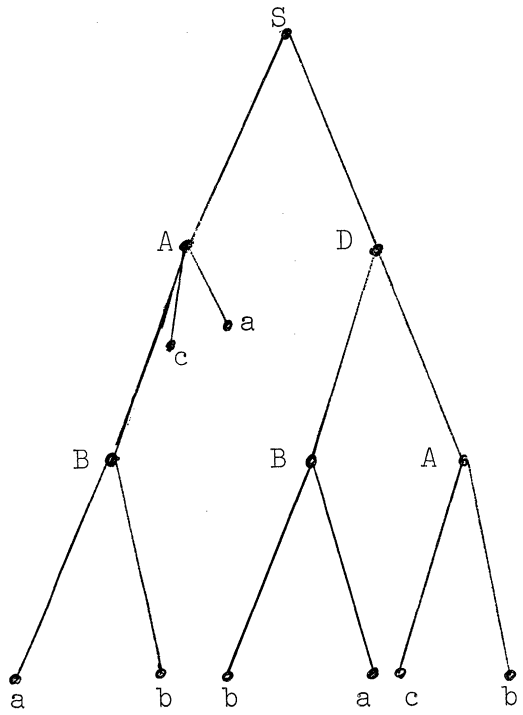
$$|V-T| = n,$$

there is an A-derivation of B if and only if $B \in W_{n+1}(A)$. In particular, $a \in L(\mathcal{G})$ if and only if $a \in W_{n+1}(S)$. The productions of the grammar $\mathcal{G}'$ are defined as follows: $(A', a_1' a_2' \ldots a_n')$, $n \geq 2$, is a production of $\mathcal{G}'$ if and only if $\mathcal{G}$ has a production $(A, a_1 a_2 \ldots a_n)$ such that $A \in W_{n+1}(A')$ and, for $i=1,2,\ldots,n$,

$$a_i' \in W_{n+1}(a_i).$$

We omit the proof that $\mathcal{G}'$ has the desired properties. As a substitute, we give the following tree diagrams, which illustrate the correspondence between derivations in $\mathcal{G}$ and derivations in $\mathcal{G}'$.





Thus, the class of context-free languages and the class of 2-context-free languages differ very little from one another. The relationship can be summed up as follows: an event P is a context-free language if and only if there exists a 2-context-free language $\mathcal{G}$ such that $L(\mathcal{G}) \subseteq P$ and every word in P and not in $L(\mathcal{G})$ is of length less than or equal to 2.

The following theorem, due to Bar-Hillel, Perles, and Shamir [20], provides the tool that we require to prove that certain languages are not context free.

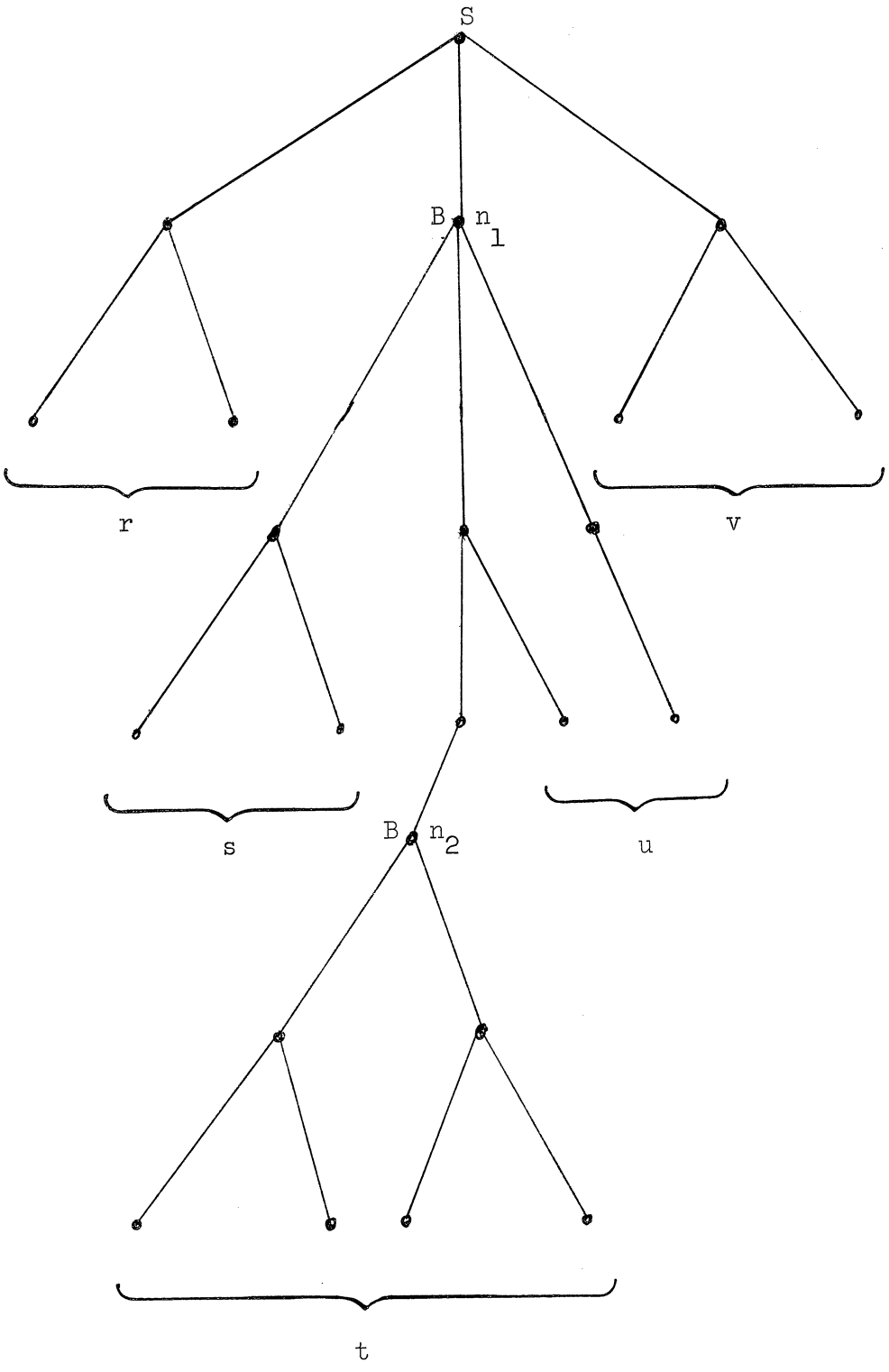
Theorem 14

Let $\mathcal{G}$ = (V, T, S, P) be a 2-context-free grammar. Then there exists an integer p such that, if $x \in L(\mathcal{G})$ and $lg(x) > p$, then x can be written in the form $x = rstuv$, where s and u are not both null and, for all $k \geq 1$, $rs^k tu^k v \in L(\mathcal{G})$.

PROOF: Let

$$n = |V-T| .$$

Then the number of S-derivations of depth less than or equal to n is finite. Let p be the maximum length of a word over the terminal alphabet T generated by an S-derivation of depth less than or equal to n. Let x be an element of $L(\mathcal{G})$ such that $lg(x) > p$. Then any S-derivation of x is of depth greater than n; therefore its tree includes some path from the root to a leaf which includes two occurrences of some nonterminal symbol B. Let the occurrence of B closer to the root be at node n_1, and the second occurrence, at n_2. Then the tree for the S-derivation of x has the form indicated in the following diagram.


S
B n1
r
v
s
u
B n2
t

Thus, the following derivations exist in $\mathcal{H}$:

$$S \Longrightarrow rBv,$$

$$B \Longrightarrow sBu,$$

$$B \Longrightarrow t,$$

where $x = rstuv.$

Since $\mathcal{H}$ is a 2-context-free grammar, it follows that s and u are not both null. But, any word $rs^k tu^k v$, $k \geq 1$, has an S-derivation in $\mathcal{H}$, as follows: apply the derivation

$$S \Longrightarrow rBv,$$

followed by k iterations of

$$B \Longrightarrow sBu,$$

followed by

$$B \Longrightarrow t.$$

This completes the proof of Theorem 14.

Using Theorem 14, we can now continue our studies of the closure properties of the class of context-free languages.

Theorem 15

The class of context-free languages is not closed under intersection.

PROOF: The language

$$L_1 = \{0^n 1 0^n, n \geq 1\}$$

is a context-free language generated by the grammar.

$$S \rightarrow 0 S 0,$$

$$S \rightarrow 010;$$

the language

$$L_2 = 1\ 0\ 0^*$$

is regular, and is therefore context-free. By Theorem 11, the language

$$L_1 \cdot L_2 = \{0^n\ 1\ 0^n\ 1\ 0^k,\ n \geq 1,\ k \geq 1\}$$

is context-free, and so is

$$\widetilde{L_1 \cdot L_2} = \{0^k\ 1\ 0^n\ 1\ 0^n,\ n \geq 1,\ k \geq 1\}.$$

Suppose

$$L_1 \cdot L_2 \cap \widetilde{L_1 \cdot L_2} = \{0^n\ 1\ 0^n\ 1\ 0^n,\ n \geq 1\}$$

is context-free. Since it contains no words of length less than 3, it must be the language generated by some 2-context-free grammar $\mathcal{H}$. Theorem 14 therefore applies, and it asserts that, for large enough n, the word

$$0^n\ 1\ 0^n\ 1\ 0^n = rstuv,$$

where s and u are not both empty and, for all $k \geq 1$,

$$r\ s^k\ t\ u^k\ v \in L(\mathcal{H}).$$

Since every word in $L(\mathcal{H})$ contains exactly 2 1's, neither s nor t contains a 1. Then rs^ktu^kv must be of the form

$$0^{n_1}\ 1\ 0^{n_2}\ 1\ 0^{n_3},$$

where

$$\max(n_1,n_2,n_3) > n_1$$

and

$$\mathrm{Min}(n_1,n_2,n_3) = n.$$

This contradicts the assumption that

$$L_1 \cdot L_2 \cap \widetilde{L_1 \cdot L_2}$$

is context-free; and this contradiction completes the proof.

Corollary 5

The class of context-free languages is not closed under complementation.

PROOF: By the identity

$$L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2},$$

closure under union and complementation would imply closure under intersection.

Corollary 6

There exist Type 1 languages that are not 1-context-free languages.

PROOF: The languages $L_1 \cdot L_2$ and $\widetilde{L_1 \cdot L_2}$ are 1-context-free, and therefore Type 1. Since the intersection of Type 1 languages is Type 1,

$$L_1 \cdot L_2 \cap \widetilde{L_1 \cdot L_2}$$

is Type 1, but not context-free.

The following theorem, first given in [20], is a fundamental one in the study of context-free languages.

Theorem 16

The intersection of a context-free language and a regular event is a context-free language.

PROOF: Let

$$\mathcal{G} = (V, T, S, P)$$

be a context-free grammar and let

$$\mathcal{A} = (T, Q, q_0, \lambda, F)$$

be a finite automaton representing the event $P(\mathcal{A})$. We shall construct a context-free grammar $\mathcal{G}'$ that generates the language

$$L(\mathcal{G}) \cap P(\mathcal{A}).$$

The vocabulary of $\mathcal{G}'$ will be

$$\{S\} \cup T \cup (Q \times (V-T) \times Q),$$

and the productions of $\mathcal{G}'$ will be as follows:

(i) For all $q \in F$,

$$S \to (q_0, S, q).$$

(ii) For every production of $\mathcal{G}$ of the form

$$A \to a_1, a_2, \ldots, a_m,$$

and for every sequence of m+1 (not necessarily distinct) states $q_{i_1}, q_{i_2}, \ldots, q_{i_m}, q_{i_{m+1}}$,

$$(q_{i_1}, A, q_{i_{m+1}}) \to (q_{i_1}, a_1, q_{i_2})(q_{i_2}, a_2, q_{i_3}) \cdots (q_{i_m}, a_m, q_{i_{m+1}}).$$

(iii) For every triple (q_i, a, q_j) such that $a \in T$ and

$$\lambda(q_i, a) = q_j,$$

$$(q_i, a, q_j) \to a.$$

Then, if $q \in F$, and $a_1, \ldots, a_m$ is a word over the alphabet V, the grammar $\mathcal{G}'$ generates all strings of the form

$$(q_0,a_1,q_{j_1})\ (q_{j_1},a_2,q_{j_2})\ \cdots\ (q_{j_{m-1}},a_m,q), \tag{1}$$

where $q_{j_1},\ldots,q_{j_{m-1}}$ are arbitrary elements of Q, provided that there is an S-derivation of $a_1,\ldots,a_m$ in q; otherwise, no string of the form (1) is generated. Suppose, further, that $a_1,\ldots,a_m$ is a string over the alphabet T, and that

$$rp_A(a_1,\ldots,a_m) = q.$$

Then, in particular, $\mathcal{G}'$ generates the sequence

$$(q_0,a_1,rp_A(a_1)),(rp_A(a_1),a_2,rp_A(a_2)),\ldots,(rp_A(a_1,\ldots,a_{m-1}),a_m,q).$$

And, from this sequence, productions listed above under (iii) yield $a_1,\ldots,a_m$. On the other hand, if

$$a_1,\ldots,a_m \in L(\mathcal{G})$$

but $rp_A(a_1,\ldots,a_m)$ is not an element of F, there can be no derivation of $a_1,\ldots,a_m$ in $\mathcal{G}'$. Therefore,

$$L(\mathcal{G}') = L(\mathcal{G}) \cap P(\mathcal{A}).$$

By a proof quite similar to the proof of Theorem 16, we shall now establish what Ginsburg [9] has called the Machine Mapping Theorem: a (nondeterministic) generalized sequential machine maps languages onto languages. A (nondeterministic) generalized sequential machine (g.s.m.) is a 5-tuple

$$\mathcal{M} = (A_I, Q, A_O, Q_O, R),$$

where the symbols A_I, Q, A_O, and Q_O have their usual meanings, and R is a set of quadruples of the form (q_i,a_j,x,q_ℓ), where q_i and q_ℓ are elements of Q, $a_j \in A_I$, and $x \in A_I^*$. The interpretation of the quadruple is as follows: if $\mathcal{M}$ is in state q_i and receives the input a_j, it may produce the output sequence x and enter state q_j. The number of quadruples with

q_i and a_j in the first two positions may be 0, 1, or greater than 1. If $a_1,\ldots,a_m$ is an element of $A_I^\dagger$, then the set

$$seq(a_1,\ldots,a_m) \subseteq A_O^*$$

is defined as follows:

$$x_1x_2\cdots x_m \in seq(a_1,\ldots,a_m)$$

if and only if R includes a set of quadruples of the form

$$\{(q_0,a_1,x_1,q_1)\ (q_1,a_2,x_2,q_2)\ \cdots\ (q_{m-1},a_m,x_m,q_m)\},$$

where $q_0 \in Q_0$. A g.s.m. is sometimes called a transducer.

Theorem 17 (Maching Mapping Theorem)

Let $\mathcal{G}$ = (V, T, S, P) be a context-free grammar, and let $\mathcal{M}$ = (T, Q, A_0, Q_0, R) be a g.s.m. with input alphabet T. Then the set

$$L' = \bigcup_{x \in L(\mathcal{G})} seq(x)$$

is a context-free language.

PROOF: We shall exhibit a context-free grammar $\mathcal{G}'$ that generates the language L'. The nonterminal vocabulary of $\mathcal{G}'$ is

$$\{S\} \cup Q \times V \times Q,$$

its terminal vocabulary is A_0, and its initial symbol is S. The productions of $\mathcal{G}'$ fall into three sets, as follows:

(i) For each pair $(q,q') \in Q_0 \times Q$,

$$S \to (q,S,q').$$

(ii) For each production of the form

$$A \to a_1a_2 \cdots a_m,$$

and for every sequence of m+1 (not necessarily distinct)

states $q_{i_1}, q_{i_2}, \ldots, q_{i_{m+1}}$,

$$(q_{0_1}, A, q_{i_{m+1}}) \rightarrow (q_{i_1}, a_1, q_{i_2})(q_{i_2}, a_2, q_{i_3}) \ldots (q_{i_m}, a_m, q_{i_{m+1}}).$$

(iii) For each quadruple $(q_i, a_j, x, q_\ell) \in R$,

$$(q_i, a_j, q_\ell) \rightarrow x.$$

The verification that this grammar generates L' will be omitted; it follows the lines of the discussion given in the proof of Theorem 16.

Decision problems of context-free grammars

In this section we show how the Post Correspondence Theorem may be applied to prove the undecidability of certain questions concerning the analysis of context-free grammars. Let A be a finite alphabet, and let $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ be ordered lists of words over the alphabet A. Let

$$Z = \{z_1, \ldots, z_n\}$$

be a set of additional symbols not in A. Let L(a) be the language over the terminal vocabulary Z U A generated by the following context-free garmmar $\mathcal{H}(a)$.

$$S \rightarrow z_1 \; S \; a_i \qquad i=1,2,\ldots,n$$

$$S \rightarrow z_i \; a_i \qquad i=1,2,\ldots,n$$

Then L(a) consists of all nonempty sequences of the form

$$z_{i_r} \; z_{i_{r-1}} \; \ldots \; z_{i_1} \; a_{i_1} \; \ldots \; a_{i_r}.$$

Similarly, let L(b) be the language over the terminal vocabulary Z U A generated by the following context-free grammar $\mathcal{H}(b)$.

$$S' \to z_i\ S'\ b_i \qquad i=1,2,\ldots,n$$

$$S' \to z_i\ b_i \qquad i=1,2,\ldots,n.$$

Then $L(a) \cap L(b)$ is nonempty if and only if the instance of the Post Correspondence Problem given by the lists $(a_1,\ldots,a_n)$ and $(b_1,\ldots,b_n)$ has a positive solution (i.e., there exists a nonempty sequence $i_1,i_2,\ldots,i_k$ such that

$$a_{i_1}\ a_{i_2}\ \ldots\ a_{i_k} = b_{i_1}\ b_{i_2}\ \ldots\ b_{i_k}).$$

Also, if

$$L(a) \cap L(b)$$

is nonempty, it is easily seen to be infinite. Thus, if there were a general procedure which, when given any two grammars of the form $\mathcal{G}(a)$ and $\mathcal{G}(b)$, could determine whether

$$L(a) \cap L(b)$$

is empty (or infinite), then the Post Correspondence Problem would be recursively solvable. Thus, we are led to the following conclusion.

Theorem 18

There is no algorithm to decide, given context-free grammars $\mathcal{G}_1$ and $\mathcal{G}_2$, whether $L(\mathcal{G}_1) \cap L(\mathcal{G}_2)$ is empty, nonempty and finite, or infinite.

Let $\mathcal{G} = (V, T, S, P)$

be a context free-grammar. Two S-derivations of a word $x \in L(\mathcal{G})$ are equivalent if the labelled trees associated with the derivations are identical. The grammar $\mathcal{G}$ is unambiguous if, for all $x \in L(\mathcal{G})$, all S-derivations of x are equivalent; otherwise, $\mathcal{G}$ is ambiguous. To

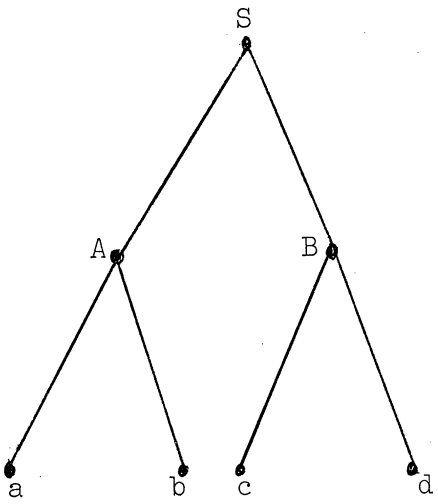
illustrate this concept, suppose $\mathcal{G}$ has the following three S-derivations of the word abcd.

$$S \to AB \to abB \to abcd;$$

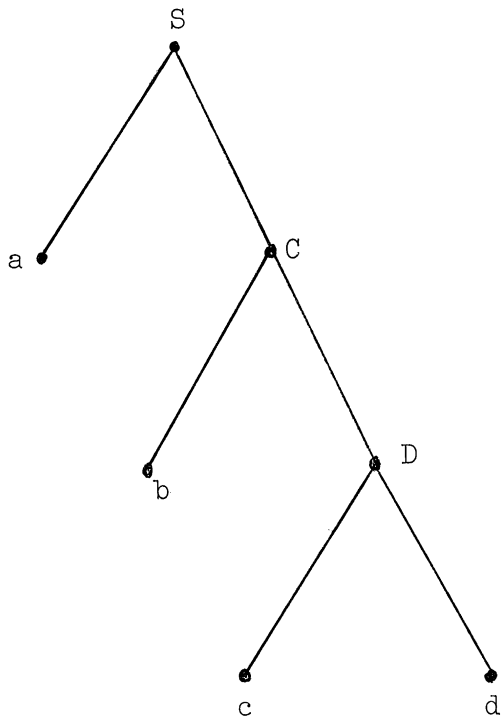
$$S \to AB \to Acd \to abcd;$$

$$S \to aC \to abD \to abcd.$$

The first two derivations are equivalent; their tree is as follows.



The third derivation is not equivalent to the other two; it has the following tree.



Because abcd has two inequivalent derivations the grammar $\mathcal{G}$ is ambiguous.

The concept of ambiguity is important in the application of context-free grammars to natural and artificial languages. From the point of view of such applications, an ambiguous context-free grammar is one that does not attach a unique structural description to each of its sentences. Its explanatory value is therefore diminished. It has been shown that there exist inherently ambiguous context-free languages, all of whose grammars are ambiguous.

Theorem 19

There is no algorithm to decide whether a context-free grammar is unambiguous.

PROOF: It is evident that the grammars $\mathcal{G}(a)$ and $\mathcal{G}(b)$ defined above are unambiguous. Consider the following grammar $\mathcal{G}(a,b)$, which generates the language L(a) ∪ L(b):

$$T \to S$$

$$T \to S'$$

$$S \to z_i\ S\ a_i \qquad i=1,2,\ldots,n$$

$$S \to z_i\ a_i \qquad i=1,2,\ldots,n$$

$$S' \to z_i\ S'\ b_i \qquad i=1,2,\ldots,n$$

$$S' \to z_i\ b_i \qquad i=1,2,\ldots,n$$

In this grammar, a word x has two inequivalent T-derivations if and only if it has one derivation in which the nonterminal symbol S occurs, and another in which S' occurs; i.e., if and only if $x \in L(a)$ and $x \in L(b)$. It follows that this grammar is ambiguous if and only if

$$L(a) \cap L(b) \neq \varphi;$$

i.e., if and only if the Correspondence Problem for the lists $(a_1,\ldots,a_n)$ and $(b_1,\ldots,b_n)$ has a positive solution. Thus, there can be no general algorithm to decide whether a grammar G(a,b) is ambiguous.

Pushdown automata

In this section we introduce the class of pushdown automata, and prove that a language is context-free if and only if it is the set of words accepted by some pushdown automaton. This result provides a deepened understanding of the technique of "predictive analysis" used in computer processing of natural language text. For, since predictive analysis is equivalent to the use of a nondeterministic pushdown automaton, the result implies that the class of languages analyzable by a predictive analysis is precisely the class of context-free languages; i.e., predictive analysis is equivalent to immediate constituent analysis.

A (nondeterministic) pushdown automaton has a one-way input tape, and a one-way infinite storage tape called a pushdown. If we call the right-most square of the pushdown that is not blank the top of the pushdown, then the characteristics of a pushdown may be described as follows:

(1) the only square that may be read or erased is the top.

(2) the only square that may be written upon is the square to the right of the top.

Thus, a pushdown is a kind of last-in first-out memory.

Formally, a (nondeterministic) pushdown automaton is a 6-tuple

$$\mathcal{A} = (A_I, Q, A_P, q_0, F, I),$$

where A_I is the input alphabet, Q is the set of internal states, A_P is the pushdown alphabet, $q_0 \in Q$ is the initial state, $F \subset Q$ is the set of final states, and I is the set of instructions, to be defined below. A complete state of $\mathcal{A}$ is a triple (x,q,z), where $x \in A_I^*$, $q \in Q$, z is of the form σy, where σ is an end marker not in $A_I \cup A_P$ and $y \in A_P^*$; x, of course, denotes the word on the input tape, and σy denotes the word on the pushdown; the rightmost symbol of σy is at the top of the pushdown. The instructions are of the form

$$(a,q_i,b) \rightarrow (q_j,w),$$

where

(i) Either $a=e$ or $a \in A_I$.

(ii) $q_i \in Q$ and $q_j \in Q$.

(iii) Either $b=e$ or $b \in A_P$ or $b = \sigma$.

(iv) Either $w \in A_P^*$ or w is the special symbol ρ;
if $w=\rho$, then $b \in A_P$.

The action of the pushdown automaton $\mathcal{A}$ is dictated by its instructions as follows. If $\mathcal{A}$ is in the complete state (x,q_i,z) where

$$x = x'a$$

$$z = z'b,$$

and has an instruction

$$(a,q_i,b) \rightarrow (q_j,w), \qquad w \neq \rho,$$

then $\mathcal{A}$ may enter the complete state

$$(x', q_j, z'bw).$$

Thus, this instruction corresponds to the operation of writing on the pushdown. If $\mathcal{A}$ is in the complete state

$$(x'a,\ q_i,\ z'b)$$

and has an instruction

$$(a,q_i,b) \rightarrow (q_j,\rho),$$

then $\mathcal{A}$ may enter the complete state

$$(x',\ q_j,\ z');$$

thus, the occurrence of ρ in an instruction indicates that the top of the pushdown is to be erased. A word x is accepted by $\mathcal{A}$ if and only if, starting from the complete state

$$(x,\ q_0,\ \sigma),$$

$\mathcal{A}$ has a sequence of operations that causes it to move beyond the end marker of the pushdown in a state $q_j \in F$, after having read all of the input tape. More precisely, the requirement is that $\mathcal{A}$ should be able to reach a complete state

$$(e,\ q_i,\ \sigma)$$

such that $\mathcal{A}$ has an instruction

$$(e,\ q_i,\ \sigma) \rightarrow (q_j,\ \rho),$$

where $q_j \in F$.

Example 1.

Consider the pushdown automaton $\mathcal{A}$ having the following instruction set:

$(a,q_0,e) \rightarrow (q_1,a)$

$(a,q_1,e) \rightarrow (q_1,a)$ $\qquad F = \{q_3\}$

$(b,q_1,e) \rightarrow (q_2,e)$

$(a,q_2,a) \rightarrow (q_2,\rho)$

$(e,q_2,\sigma) \rightarrow (q_3,\rho)$

The set of words accepted by $\mathcal{A}$ is $\{a^n b a^n, n \geq 1\}$. A sequence of operation leading to the acceptance of the word $a^2 b a^2$ is as follows.

Input tape	State	Pushdown	Instruction
a a b a a	q_0	σ	$(a,q_0,e) \rightarrow (q_1,a)$
a a b a	q_1	σa	$(a,q_1,e) \rightarrow (q_1,a)$
a a b	q_1	σaa	$(b,q_1,e) \rightarrow (q_2,e)$
a a	q_2	σaa	$(a,q_2,a) \rightarrow (q_2,\rho)$
a	q_2	σa	$(a,q_2,a) \rightarrow (q_2,\rho)$
e	q_2	σ	$(e,q_2,\sigma) \rightarrow (q_3,\rho)$.

Theorem 20

Let $\mathcal{G} = (V, T, S, P)$ be a context-free grammar. Then one can construct a nondeterministic pushdown automaton $\mathcal{A} = (T, Q, V\text{-}T, S_{\downarrow}, I)$ for which the set of words accepted is $L(\mathcal{G})$.

PROOF: We shall give the construction of $\mathcal{A}$ and an indication of why it works. Without loss of generality, we may assume that each of the productions of $\mathcal{G}$ is of one of the following forms:

$A \rightarrow a,$

$A \rightarrow e,$

or

$$A \to BC,$$

where $a \in T$,

$$A \in V\text{-}T,$$

$$B \in V\text{-}T,$$

$$C \in V\text{-}T,$$

and no nonterminal symbol X is contained both in productions of the form

$$A \to XB$$

and productions of the form

$$C \to DX.$$

Corresponding to each element

$$A \in V\text{-}T,$$

there are two internal states, $A_{\downarrow}$ and $A_{\uparrow}$. The elements of I are determined from the productions as follows:

Production	Instructions
$A \to a$	$(a, A_{\downarrow}, e) \to (A_{\uparrow}, e)$
$A \to e$	$(e, A_{\downarrow}, e) \to (A_{\uparrow}, e)$
$A \to BC$	$(e, A_{\downarrow}, e) \to (B_{\downarrow}, A)$
	$(e, B_{\uparrow}, e) \to (C_{\downarrow}, e)$
	$(e, C_{\uparrow}, A) \to (A_{\uparrow}, \rho).$

In addition, there is the instruction

$$(e, S_{\uparrow}, \sigma) \to (\Sigma, \rho)$$

where Σ is the only element of F.

Example 2.

The context-free language

$$\{a^n b a^n,\ a \geq 1\}$$

is generated by the following context-free grammar (with initial symbol S):

$$S \to BA$$

$$B \to CS$$

$$A \to a$$

$$C \to a$$

$$S \to b.$$

The corresponding pushdown automaton has the following instructions:

$$(a, A\downarrow, e) \to (A\uparrow, e)$$

$$(a, C\downarrow, e) \to (C\uparrow, e)$$

$$(b, S\downarrow, e) \to (S\uparrow, e)$$

$$(e, S\downarrow, e) \to (B\downarrow, S)$$

$$(e, B\uparrow, e) \to (A\downarrow, e)$$

$$(e, A\uparrow, S) \to (S\uparrow, \rho)$$

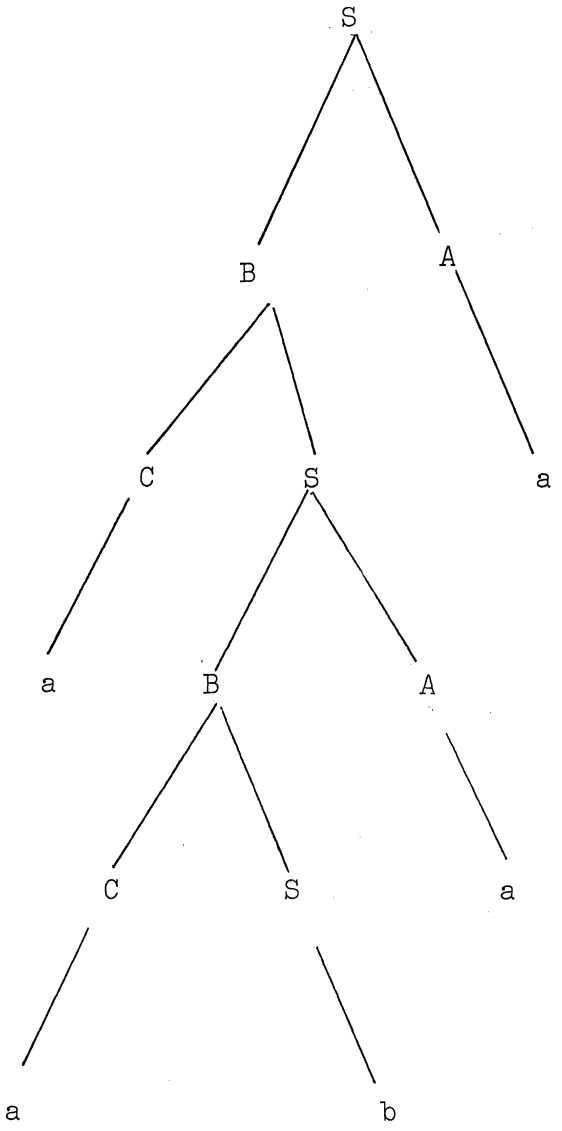
$$(e, B\downarrow, e) \to (C\downarrow, B)$$

$$(e, C\uparrow, e) \to (S\downarrow, e)$$

$$(e, S\uparrow, B) \to (B\uparrow, \rho)$$

$$(e, S\uparrow, \sigma) \to (\Sigma, \rho).$$

The following tree denotes an S-derivation of the word aabaa.



A sequence of operation corresponding to the acceptance of the word aabaa is as follows:

Input tape	State	Pushdown
a a b a a	$S\downarrow$	σ
a a b a a	$B\downarrow$	σS
a a b a a	$C\downarrow$	σSB
a a b a	$C\uparrow$	σSB
a a b a	$S\downarrow$	σSB
a a b a	$B\downarrow$	σBS
a a b a	$C\downarrow$	σSBSB
a a b	$C\uparrow$	σSBSB
a a b	$S\downarrow$	σSBSB
a a	$S\uparrow$	σSBSB
a a	$B\uparrow$	σSBS
a a	$A\downarrow$	σSBS
a	$A\uparrow$	σSBS
a	$S\uparrow$	σSB
a	$B\uparrow$	σS
a	$A\downarrow$	σS
e	$A\uparrow$	σS
e	$S\uparrow$	σ

When the pushdown automaton is in a state $A\downarrow$, it "predicts" what production with A as its left-hand side is being applied. If the predicted production is of the form

$$A \rightarrow a,$$

it checks the input tape to determine whether a is the first symbol; if the predicted production is of the form

$$A \rightarrow BC,$$

the automaton stores A on the pushdown for reference in later "backtracking," and goes to the state $B\downarrow$. The states of the form $A\uparrow$ are used in the later backtracking process. The converse of Theorem 20 is also true. We omit the proof.

Theorem 21

Let $\mathcal{A} = (A_I, Q, A_P, q_0, F, I)$. Then the set of words accepted by $\mathcal{A}$ is a context-free language.

References

[1] G. Birkhoff and S. MacLane, A Survey of Modern Algebra, Revised Edition, Macmillan, New York (1953).

[2] A. H. Clifford and G. B. Preston, The Algebraic Theory of Semigroups, vol. I, American Mathematical Society, Providence (1961).

[3] M. Davis, Computability and Unsolvability, McGraw-Hill, New York (1958).

[4] M. Hall, The Theory of Groups, Macmillan, New York (1959).

[5] D. Arden, "Delayed logic and finite state machines," The University of Michigan Summer Session on the Theory of Computing Machine Design (1960).

[6] J. Brzozowski, "A survey of regular expressions and their applications," IRE Transactions on Electronic Computers, vol. EC-11, June 1962, pp. 324-335.

[7] J. Brzozowski, "Derivatives of regular expressions," Journal of the ACM, vol. 11, October 1964, pp. 481-494.

[8] J. R. Büchi, Mathematical Theory of Automata, notes for Communication Sciences 403, given by J. B. Wright and J. R. Büchi, The University of Michigan, fall 1960.

[9] S. Ginsburg, An Introduction to Mathematical Machine Theory, Addison-Wesley, Palo Alto (1962).

[10] M. Harrison, Notes for Electrical Engineering 667, The University of Michigan, spring 1963.

[11] J. Hartmanis, "On the state assignment problem for sequential machines, I," IRE Transactions on Electronic Computers, vol. EC-10, June 1961, pp. 157-165.

[12] J. Hartmanis and R. E. Stearns, "Pair algebra and its application to automata theory," Information and Control, vol. 7, December 1964, pp. 485-507.

[13] K. Krohn and J. Rhodes, "Algebraic theory of machines, I," to appear in Transactions of the American Mathematical Society.

[14] E. F. Moore, "Gedanken-experiments on sequential machines," Automata Studies [Annals of Mathematics Studies, No. 34], Princeton University Press (1956), pp. 129-153.

[15] E. F. Moore (editor), Sequential Machines: Selected Papers, Addison-Wesley, Palo Alto (1964).

[16] J. Myhill, "Finite automata, semigroups and simulation," The University of Michigan Summer Conference on Automata Theory (1963).

[17] M. Rabin and D. Scott, "Finite automata and their decision problems," IBM Journal of Research and Development, vol. 3, no. 2, April 1959, pp. 114-125. (Also in [15], pp. 63-91).

[18] R. E. Stearns and J. Hartmanis, "On the state assignment problem for sequential machines, II," IRE Transactions on Electronic Computers, vol. EC-10, December 1961, pp. 593-603.

[19] H. P. Zeiger, "Loop-free synthesis of finite-state machines," Massachusetts Institute of Technology Research Laboratory for Electronics report, September 1964.

[20] Y. Bar-Hillel, M. Perles, and E. Shamir, "On formal properties of simple phrase structure grammars," Zeit. für Phonetik Sprachwissenschaft und Kommunikationsforschung, Band 14, Heft 2 (1961), pp. 143-172.

[21] N. Chomsky, "Formal properties of grammars," Handbook of Mathematical Psychology, vol. 2, Wiley, New York (1962), pp. 324-418.

[22] N. Chomsky, "Context-free grammars and pushdown storage," Massachusetts Institute of Technology Research Laboratory for Electronics report (1962).

[23] N. Chomsky and M. Schützenberger, "The algebraic theory of context-free languages," in Studies in Logic, Computer Programming and Formal Systems, North-Holland, Amsterdam (1963), pp. 118-161.

[24] C. C. Elgot and J. Mezei, "Two-sided finite-state transductions," Proceedings of the Fourth Symposium on Switching Circuit Theory and Logical Design, IEEE (1963).

[25] C. C. Elgot and J. L. Rutledge, "RS-machines with almost blank tape," Journal of the ACM, July 1964, pp. 313-337.

[26] J. Evey, "Application of pushdown-store machines," AFIPS Conference Proceedings, vol. 24 (1963), pp. 215-227.

[27] S.-Y. Kuroda, "Classes of languages and linear bounded automata," Information and Control, vol. 7 (1964), pp. 207-223.

[28] P. S. Landweber, "Three theorems on phrase structure grammars of type 1," Information and Control, vol. 6 (1963), pp. 131-137.

[29] P.S. Landweber, "Decision problems of phrase-structure grammars," IEEE Transactions on Electronic Computers, vol. EC-13, no. 4, August 1964, pp. 354-362.

[30] M. L. Minsky, "Recursive unsolvability of Post's problem of Tag," Ann. of Math., vol. 74, (1961), pp. 437-455.

[31] J. Myhill, "Linear bounded automata," WADD Tech. Note 60-165, Wright Air Development Division, Wright-Patterson Air Force Base (1960).

[32] A. G. Oettinger, "Automatic analysis and the pushdown stores," Proceedings of the 12th Symposium in Applied Mathematics (R. Jakobson, editor), (1961).

[33] E. L. Post, Bull. A.M.S., vol. 52 (1945), pp. 264-268.

[34] E. L. Post, J. Symbolic Logic, vol. 12 (1947), pp. 1-11.

[35] P. M. Postal, "Constituent analysis," Int. J. Am. Ling., Supplement (1964).

[36] A. L. Rosenberg, "On N-tape finite-state acceptors," Proceedings of the Fifth Symposium on Switching Theory and Logical Design, IEEE (1964).

[37] J. C. Shepherdson, "The reduction of two-way automata to one-way automata," IBM J. of Research and Development, vol. 3 (1959), pp. 198-200. (Also in [15], pp. 92-97.)

[38] V. Yngve, "The depth hypothesis," Proceedings of the 12th Symposium in Applied Mathematics (R. Jakobson, editor (1961).