Generation of regular expressions for automata by the integral of regular expressions

L. W. Smith and S. S. Yau

Northwestern University, Evanston, Illinois, 60201, USA*

In this paper, the integral of regular expressions is defined and its properties are presented— The concept of the integral is then applied to establishing an algorithm for generating the regular expression of an arbitrary finite automaton specified by its state diagram or flow table. The regular expression for a given automaton obtained by this algorithm is shown to be unique up to the commutative property of the sum. This algorithm is effective and suitable for machine implementation. Programmed results of this algorithm are also given and interpreted.

(Received August 1971)

1. Introduction

The relationship between regular expressions and finite automata is well known (Kleene, 1954; Copi, Elgot and Wright, 1958) and a number of authors have studied the conversion between regular expressions and finite automata. McNaughton and Yamada (1960) presented a pair of algorithms which systematically converted a regular expression to a finite automaton and vice versa. Although these algorithms are systematic, they have been shown to be often lengthy and too tedious. Using the techniques suggested by Arden (1961), Brzozowski and McCluskey (1963) gave a signal flow graph method for finding regular expressions from a finite automaton. Even though quite effective, the technique largely depends on the intuition and insight of the user. Later, Brzozowski (1964) defined the derivative of a regular expression, and showed this to be the natural and most systematic way of going from a regular expression to the state diagram of an automaton.

In this paper, we will describe the notation of the integral of a regular expression analogous to that of a derivative. We will show how, from a given deterministic state diagram, we can use the notion of an integral of a regular expression to determine its corresponding regular expression. The developed algorithm will be shown to be systematic and effective. Finally, we will present and interpret some of the results which were obtained when the algorithm was programmed on the IBM 360/67 using SNOBOL4.

We will concern ourselves with the usual model of a finite deterministic automaton

$M = \{A, Z, Q, f, g\}$,

where

$A = \{a_0, a_1, \ldots, a_{v-1}\}$	is a finite set of inputs to M.
$Z = \{z_0, z_1, \ldots, z_{w-1}\}$	is a finite set of outputs from M.
$Q = \{q_0, q_1, \ldots, q_{u-1}\}$	is a finite set of states of M.
$f: Q \times A \to Q$	is the next state function of M.
$g: Q \to Z$	is the output function of M.

The results will be presented in terms of the Moore model. The output of the automaton is the output of the final state. In the case, where the output alphabet is $\{0, 1\}$, the output of '1' will be denoted by a broken circle around the output state. The results also apply equally as well to the Mealy model with a slight modification of the algorithm.

2. Regular expressions

Definition 1

A sequence over the finite alphabet A is defined to be a series of concatenated symbols from the alphabet A, λ, and ϕ.

The regular operations *sum*, *product*, and *star* on sets of sequences P, P_1, $P_2, \ldots$ over the alphabet A are defined as follows:

Definition 2.1

The regular operation *sum* on the sets of sequences P_1 and P_2, denoted by $(P_1 + P_2)$, is defined to be the set of sequences

$$P_1 + P_2 = \{p | p \in P_1 \text{ or } P_2\} . \qquad (1)$$

Definition 2.2

The regular operation *product* on the sets of sequences P_1 and P_2, denoted by (P_1P_2), is defined to be the set of sequences

$$P_1P_2 = \{p | p = p_1p_2 ; p_1 \in P_1, p_2 \in P_2\} \qquad (2)$$

Definition 2.3

The regular operation *star* on the set of sequences P, denoted by P^*, is defined to be the set of sequences

$$P^* = \sum_{n=0}^{\infty} P^n , \qquad (3)$$

where $P^0 = \lambda$, $P^1 = P$, $P^2 = PP, \ldots$

These regular operations have the following properties:

Property 1: *Sum* is associative and communicative.

Property 2: *Product* is associative but not communicative.

Property 3: *Product* is distributive over *sum*, but *sum* is not distributive over product.

Property 4: λ serves as the multiplicative unity.

Property 5: ϕ serves as the additive and multiplicative zero.

Now a regular expression can be defined as follows:

Definition 3

A *regular expression* over an alphabet $A = \{a_0, a_1, \ldots, a_{v-1}\}$, can be defined as follows:

1. The letters $a_0, a_1, \ldots, a_{v-1}, \lambda$ and ϕ are regular expressions.
2. If P_1 and P_2 are regular expressions, then so are $P_1 + P_2$, P_1P_2 and $P_1{}^*$.
3. Nothing else is a regular expression unless it follows from repeated applications of (1) and (2).

3. The integral of a regular expression

Before we define the integral of a regular expression, we must define another operation on the set of sequences R called the derivative of R (Brzozowski, 1964).

Definition 4

Given a set of sequences R and a sequence s, the *derivative* of R with respect to s, denoted by D_sR, is defined to be

$$D_sR = \{p | sp \in R\} . \qquad (4)$$

*Departments of Computer Sciences and Electrical Engineering.

We now define the operation on the set of sequences I, called the integral of I, in terms of the derivative of R.

Definition 5
Given a set of sequences I which is a derivative of the regular set of sequences R and a letter a_i, the *integral* of I with respect to a_i, denoted by $\int I\,da_i$, is defined to be

$$\int I\,da_i = \{I' | I' = a_i I, I' \subset R\} . \tag{5}$$

The integral of I with respect to a finite sequence $s_n = a_{i_1} a_{i_2} \ldots a_{i_n}$ is defined recursively to be

$$\int \cdots \left[\int \left(\int I\,da_{i_n} \right) da_{i_{n-1}} \right] \cdots da_{i_1} = a_{i_1} \ldots a_{i_{n-1}} a_{i_n} I \subset R . \tag{6}$$

The integration of and the integration by the special letters λ and ϕ are defined as follows:

Definition 6
Given a set of sequences I, the *integral* of I with respect to λ is defined to be

$$\int I\,d\lambda = I \tag{7}$$

and the integral of I with respect to ϕ is defined to be

$$\int I\,d\phi = \phi . \tag{8}$$

Definition 7
Given the special letters λ and ϕ, the *integral* of λ with respect to the letter a_i is defined to be

$$\int \lambda\,da_i = a_i \tag{9}$$

and the integral of ϕ with respect to a letter a_i is defined to be

$$\int \phi\,da_i = \phi . \tag{10}$$

Since the integrations of I with respect to λ and ϕ produce I and ϕ respectively, these integrations will be considered as trivial. We will, hence, consider only the integration of a regular expression I with respect to a series of single letters to form another integral I' which is a subset of the original regular expression R. Therefore, we will next define what we mean by the complete integral of a regular expression with respect to the alphabet A. Before we do this, we must define another integral called the λ-integral.

Definition 8
The *λ-integral*, denoted by Z, for the complete integral D_sR is defined to be

$$Z = \begin{cases} \lambda \text{ if the sequence } s \in R \\ \phi \text{ if the sequence } s \notin R \end{cases} \tag{11}$$

Definition 9
Given the regular expression $D_{a_i}R$, one for every $a_i \in A$, the *complete integral* over the alphabet A is defined to be the regular expression

$$R = D_\lambda R = \sum_{i=0}^{v-1} \int D_{a_i}R\,da_i + Z . \tag{12}$$

If we substitute any finite sequence s of letters of the alphabet A for λ in the above equation, the complete integral D_sR over the alphabet A becomes

$$D_sR = \sum_{i=0}^{v-1} \int D_{sa_i}R\,da_i + Z . \tag{13}$$

Next, consider the relationship between the states of the automaton M and the set of input sequences. Each state q_i of M can be identified by the set of all input sequences, in which every input sequence will take M from q_i into an output state. From this set of input sequences, we form a regular expression and obtain the following properties.

Property 6: Given an automaton M, each state q_i can be characterised by a regular expression, $D_{s_i}R$, in which each input sequence takes M from q_i into some output state.

Property 7: If two regular expressions are equivalent, then their corresponding states of the automaton are equivalent.

Let s_i and s_j be two arbitrary input sequences taking the automaton M from q_0 to q_i and q_j respectively, and let R be the regular expression defining M. Then, we have the following theorem.

Theorem 1
The two states q_i and q_j of an automaton M characterised by the complete integrals $D_{s_i}R$ and $D_{s_j}R$ are equivalent if and only if the complete integrals are equivalent.

Proof
The 'if' part is evident from Property 7, and we only need to show the 'only if' part of the theorem. Let q_i and q_j be equivalent. Then the next state of q_i and q_j via a_i are either equivalent or identical. Therefore, $D_{s_i a_i}R$ is equal to $D_{s_j a_i}R$ for all $a_i \in A$. Also, if q_i is equivalent to q_j, both q_i and q_j have or do not have an output. Therefore,

$$\sum_{i=0}^{v-1} \int D_{s_i a_i}R\,da_i + Z = \sum_{i=0}^{v-1} \int D_{s_j a_i}R\,da_i + Z \tag{14}$$

$$D_{s_i}R = D_{s_j}R .$$ Q.E.D.

As a consequence of this theorem, if we have given a state q_i characterised by the regular expression R_i which is unknown and some regular expression $s_1 R_i + s_2$, derived through integration, we have

$$R_i = s_1 R_i + s_2 = s_1{}^* s_2 . \tag{15}$$

In order to avoid confusion, the state q_i and the characterising regular expression R_i will be represented by just q_i from here on, i.e. (15) can be written as

$$q_i = a_1 q_i + s_2 = s_1{}^* s_2 . \tag{16}$$

4. An algorithm for constructing a regular expression from a state diagram

Definition 10:
Two input sequences over the alphabet A are *similar* with respect to an automaton M if and only if starting in the same state q_i of M they terminate in the same state q_j of M.

From this definition we formulate the following theorem.

Theorem 2
Given the set T of all possible finite input sequences over the input alphabet $A \cup \{\lambda\}$ and the state flow table, T can be partitioned into n similar classes, where n is the number of states in the flow table.

Proof
We know that under a deterministic system the input sequence λ carries any state back into itself. Also, we know that every finite input sequence from the initial state must end with one of the n states of M. Q.E.D.

Following directly from Theorem 2, we have the following theorem.

Theorem 3
Associated with each similar class formed from the set T of all possible input sequences over the input alphabet $A \cup \{\lambda\}$ is a state q_i.

From Brzozowski's work (1964), we know that each state of an automaton M can be represented by a regular expression D_sR which is a derivative of the initial regular expression R with respect to some input sequence s. Using this idea we choose the minimal input sequence (i.e. input sequence of least lexographical ordering) from each of the similar classes and form a *minimal derivative* $D_{s_j}R = q_j$ which is equivalent to the regular expression for the state q_j.

Definition 11
A derivative $D_{s_j}R$ for state q_j is *minimal* if and only if there exists no other derivative $D_{s_k}R$ for state q_j such that s_k is less lexographically than s_j.

For simplicity we will write $D_{s_k}R = q_j$, where q_j is understood to represent both the state and the regular expression representing the state. From Definition 9, we recall that each state can be represented by the complete integral

$$D_{s_j}R = \sum_{i=0}^{v-1} \int D_{s_ja_i}R\, da_i + Z\,, \tag{17}$$

where $D_{s_j}R$ represents some state q_j in the automaton M, and $D_{s_ja_i}R$ the state entered from q_j using the input letter a_i. Thus, we have the relation between the derivative of the regular expression used to determine the states of M and the integral of the states of M used to form the regular expression. Taking the set of minimal derivatives, which were discussed in the preceding paragraph, we place them in a lexographical list according to their input sequences. The minimal derivative associated with the null state is skipped since both its integration and differentiation will always equal ϕ. Then, a tree can be constructed as follows: Starting from the first minimal derivative $D_\lambda R$ on the list, draw a branch to each $D_{a_i}R$, where $a_i \in A$. If $D_{a_i}R$ is not minimal, then no branches will be drawn from $D_{a_i}R$. If $D_{a_i}R$ is minimal, draw a branch from $D_{a_i}R$ to each $D_{a_ia_j}R$, where $a_j \in A$. Apply this process repeatedly until the list of minimal derivatives is exhausted. The corresponding state is next associated with each of these derivatives. If the derivative is equal to the null state, the corresponding state is replaced by ϕ.

At this point, we have taken each state q_i of the state diagram and associated it with a derivative $D_{s_j}R$. We have shown that this derivative is also the complete integral (17). From this information we have built a tree in which each node corresponds to a complete integral, $D_{s_j}R$, and each branch corresponds to one of the integrands $D_{s_ja_i}R$ used to form it. We will now investigate four rules for manipulating these integrals based on the properties of a regular expression under integration.

Rule 1: *Combinatory Rule.* If we have the following integrals over the input alphabet $\{a_i, a_j\}$

$$\int D_{sa_i}R\, da_i = a_iP_1$$
and
$$\int D_{sa_j}R\, da_j = a_jP_2; \tag{18}$$

then we have the integral

$$D_sR = a_iP_1 + a_jP_2\,. \tag{19}$$

Rule 2: *Output Rule.* If the state represented by this integral has an output, add λ to the integral.

Rule 3: *Substitution Rule.* If state q_k is associated with the regular expression P_1 and the state q_j with the regular expression P_2q_k, then q_j corresponds to P_2P_1.

Rule 4: *Star Rule.* If a state q_k is associated with

$$P_1q_k + P_2 + P_3\,,$$

then the integral of q_k corresponds to $P_1^*(P_2 + P_3)$.

Rules 1 and 2 are based on Definition 2.1 and Definition 8 respectively. These two rules give us the general form for the complete integral. Rule 3 is the general substitution rule and needs no further explanation. Rule 4 is a restatement of Arden's theorem (Arden, 1961).

Using these concepts we state the following algorithm for determining a regular expression for a given state diagram.

1. Form the flow table and lexographical list of minimal derivatives beginning with $D_\lambda R$. Go to Step 2.
2. Construct a tree from the minimal derivatives. Go to Step 3.
3. Take the branches from the node corresponding to the last minimal derivative in the list of Step 1 and integrate the derivatives of the terminal nodes of the branches. Go to Step 4.
4. Combine the separate integrals in Step 3 according to the order of the integral manipulation Rules 1 to 4 to form the complete integral of the last minimal derivative. Then, delete this last minimal derivative. Go to Step 5.
5. Repeat Steps 3 and 4 for each minimal derivative in the reverse order of the list in Step 1 until all the integration is complete.

The procedure will now be illustrated by the following example.

Step 1: Given the state diagram in **Fig. 1** with q_0 and q_5 being the starting state and the null state respectively, we generate the flow table shown in **Fig. 2** and the lexographical list of minimal derivatives.

$$D_\lambda R$$
$$D_1R$$
$$D_{10}R$$
$$D_{11}R$$
$$D_{110}R$$

Step 2: Using the list of minimal derivatives from Step 1, the tree shown in **Fig. 3** is constructed and the proper state or ϕ is associated with each entry.

Step 3: Integrating $D_{1100}R$ and $D_{1101}R$, we obtain

$$\int D_{1100}R\, d0 = \int \phi\, d0 = 0\phi = \phi$$
$$\int D_{1101}R\, d1 = \int q_3\, d1 = 1q_3$$

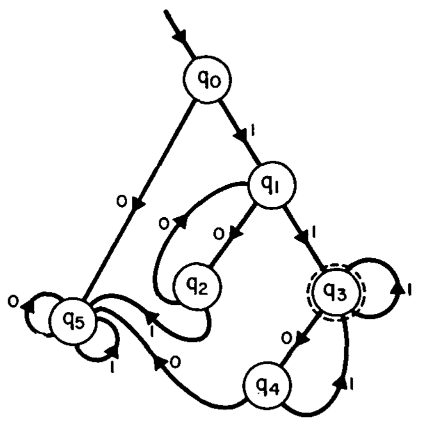


Fig. 1. The state diagram for the illustrative example

	0	1	z
q_0	q_5	q_1	
q_1	q_2	q_3	
q_2	q_1	q_5	
q_3	q_4	q_3	x
q_4	q_5	q_3	
q_5	q_5	q_5	

Fig. 2. The flow table for Fig. 1

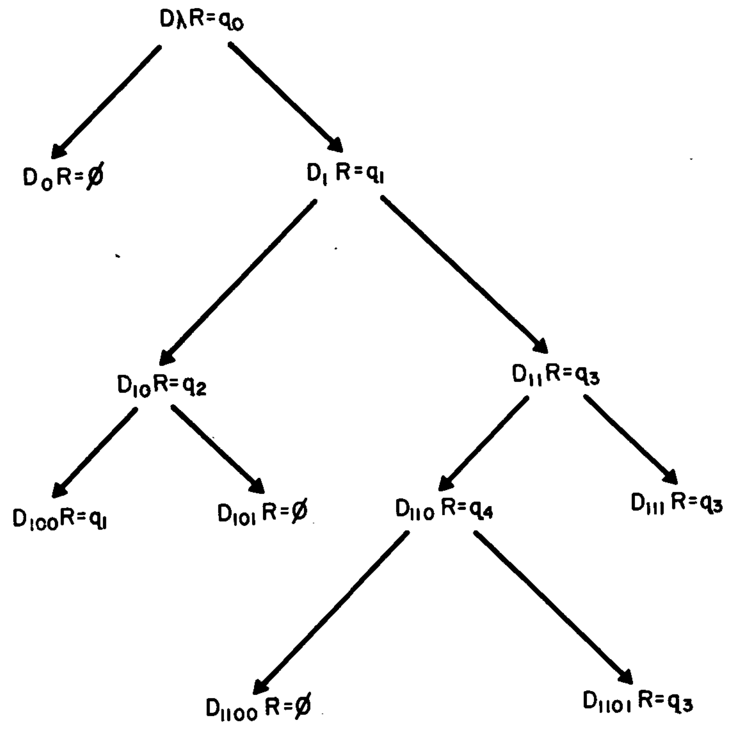


Fig. 3. The tree formed from the set of minimal derivatives obtained in Step 1 for the example

Step 4: Combining these integrals according to Rule 1, we have

$$q_4 = 1q_3$$

Step 5: Repeating Steps 3 and 4 for other minimal derivatives in the reverse order of the list of minimal derivatives in Step 1, we have

$\int D_{110}R\,d0 = \int 1q_3\,d0 = 01q_3$

$\int D_{111}R\,d1 = \int q_3\,d1 = 1q_3$

$q_3 = 01q_3 + 1q_3 + \lambda$ By Rule 1,2

$\quad = (01 + 1)^*$ By Rule 4

$\int D_{100}R\,d0 = \int g_1\,d0 = 0g_1$

$\int D_{101}R\,d1 = \int \phi\,d1 = 1\phi = \phi$

$g_2 = 0g_1$ By Rule 1

$\int D_{10}R\,d0 = \int 0g_1\,d0 = 00g_1$

$\int D_{11}R\,d1 = \int (01 + 1)^*\,d1 = 1(01 + 1)^*$

$q_1 = 00g_1 + 1(01 + 1)^*$ By Rule 1

$\quad = (00)^*1(01 + 1)^*$ By Rule 4

$\int D_0R\,d0 = \int \phi\,d0 = 0\phi = \phi$

$\int D_1R\,d1 = \int (00)^*1(01 + 1)^*\,d1 = 1(00)^*1(01 + 1)^*$

$q_0 = 1(00)^*1(01 + 1)^*$ By Rule 1

From the discussion of the algorithm and the illustrative example, we obtain the following basic properties for the algorithm.

Property 8: The algorithm is independent of the numbering of the states.

Property 9: The resultant regular expression for the state diagram is independent of the order in which the integration is performed.

Property 10: The resultant regular expression is dependent only on the tree formed by the minimal derivatives. This tree is explicitly defined by the algorithm. These properties yield the following theorem.

Theorem 4
The regular expression for a given state diagram obtained by the above algorithm is unique up to the commutative property of the sum.

5. Machine implementation

In this paper we have considered the notion of the integral of a regular expression and have shown how it can be applied to a state diagram to generate its corresponding regular expression. The algorithm has been implemented on IBM 360/67 computer.

A variety of state diagrams ranging in size from 3 to 45 states were used to verify and collect statistical data on the algorithm. **Fig. 4** shows the variation in run time as the number of states in the diagram increased. A set of state diagrams were formed using the same number of states and varying their interconnecting paths. The time required to determine the regular expressions from this set of state diagrams was recorded. Changing the number of states, different sets of state diagrams with varying interconnecting paths were used. Later the times for each of separate sets of state diagrams were arranged in ascending order and graphed. After proper scaling the results were found to be similar. **Fig. 5** shows this basic graph, where T is the least amount of time required to compute any regular expression of n states and the scale along the abscissa gives the proportion of regular expressions computed from the set of state diagrams of n states at some time xT, where $x > 1$. This increase in the amount of time required to compute the various regular expressions of the same number of states was considered to be an indicator of the complexity of that derived regular expression due to the algorithm. When the state diagram generated a regular expression of the form $R = E$ or EU^*, where E did not contain the universal sequence U^* which for the set $\{0, 1\}$ is equal to $(0 + 1)^*$, the run time was almost minimal. In the case where the regular expression could have been written in the form $R = U^*E$, the derived regular expression incorporated the sequence E into the sequence U^*, increasing its complexity and the run time. Combining the information given in Figs. 4 and 5, we are able to determine the

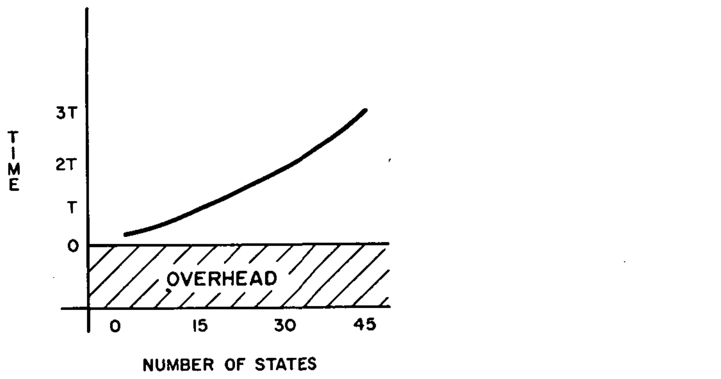


Fig. 4. Variation in computer run time for state diagrams with increasing numbers of states

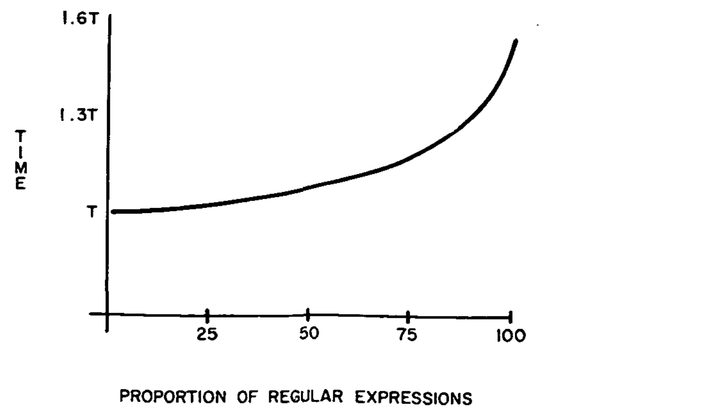


Fig. 5. Variation in computer run time for state diagrams with a constant number of states but varying interconnecting paths

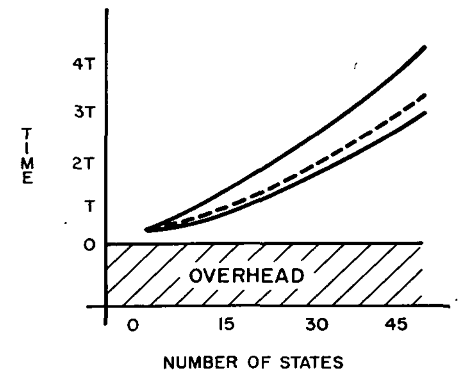
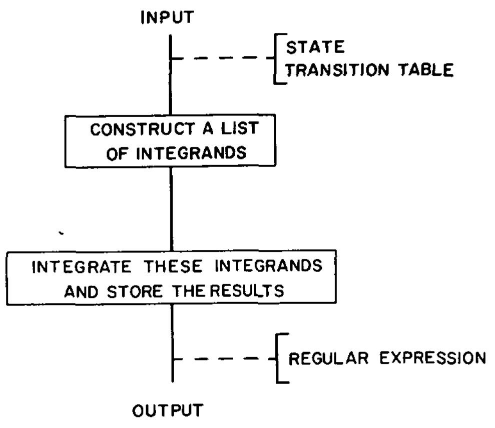
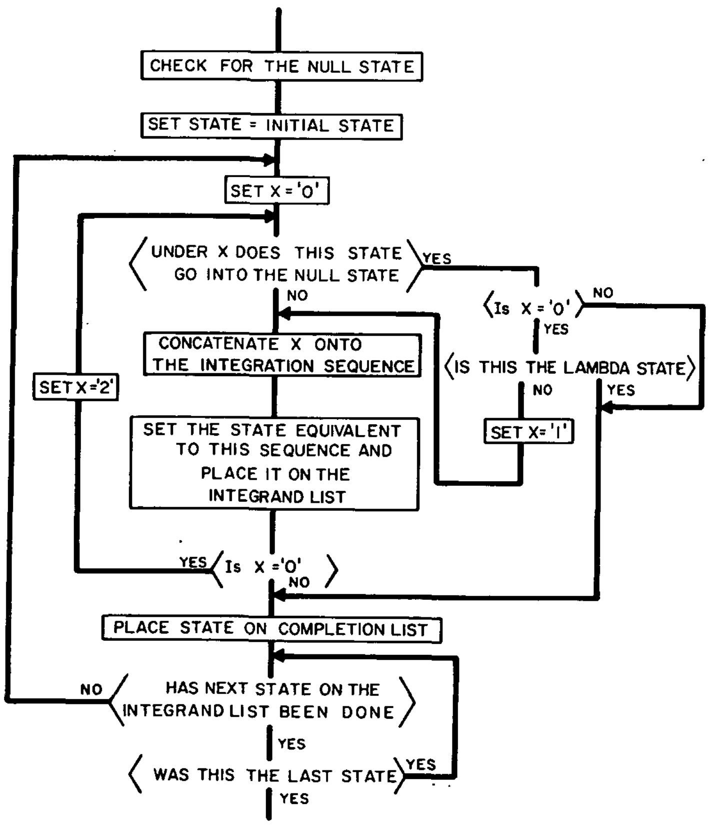


Fig. 6. Derived upper and lower bounds for Fig. 4


INPUT
STATE TRANSITION TABLE
CONSTRUCT A LIST OF INTEGRANDS
INTEGRATE THESE INTEGRANDS AND STORE THE RESULTS
REGULAR EXPRESSION
OUTPUT
Fig. 7. The general flow chart for the SNOBOL4 program

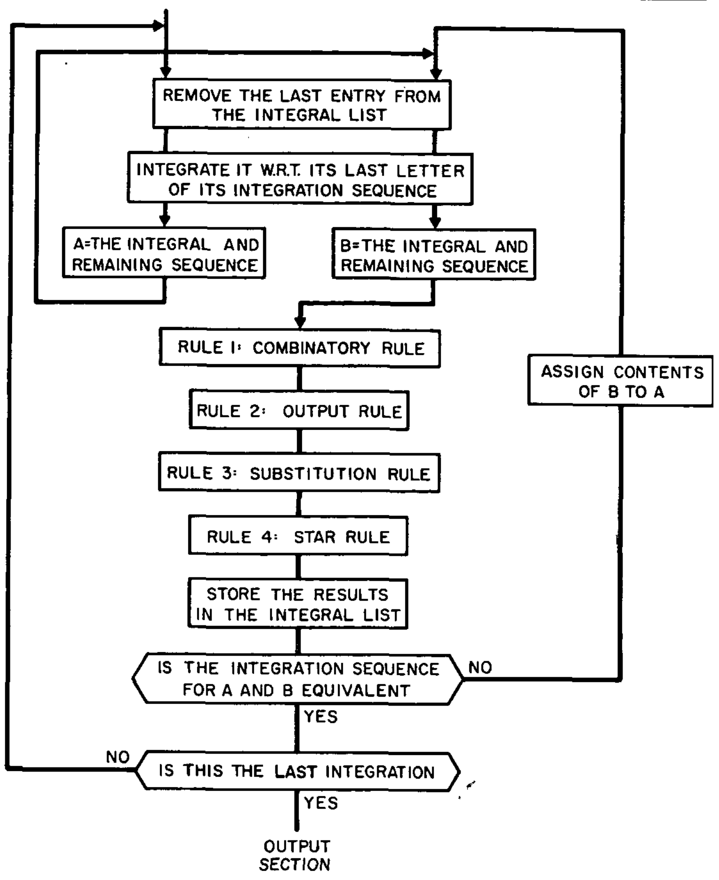

CHECK FOR THE NULL STATE
SET STATE = INITIAL STATE
SET X = '0'
UNDER X DOES THIS STATE GO INTO THE NULL STATE
YES
NO
Is X = '0'
NO
YES
CONCATENATE X ONTO THE INTEGRATION SEQUENCE
IS THIS THE LAMBDA STATE
NO
YES
SET X='2'
SET X='1'
SET THE STATE EQUIVALENT TO THIS SEQUENCE AND PLACE IT ON THE INTEGRAND LIST
YES
Is X = '0'
NO
PLACE STATE ON COMPLETION LIST
NO
HAS NEXT STATE ON THE INTEGRAND LIST BEEN DONE
YES
WAS THIS THE LAST STATE
YES
YES
Fig. 8. Procedure for forming the List of Integrands

upper and lower boundaries for the run time which are given in **Fig. 6.**

Appendix

The SNOBOL4 program, whose flowchart is shown in **Fig. 7,** consists of basically four parts:

1. An input section.


REMOVE THE LAST ENTRY FROM THE INTEGRAL LIST
INTEGRATE IT W.R.T. ITS LAST LETTER OF ITS INTEGRATION SEQUENCE
A=THE INTEGRAL AND REMAINING SEQUENCE
B=THE INTEGRAL AND REMAINING SEQUENCE
RULE 1: COMBINATORY RULE
RULE 2: OUTPUT RULE
RULE 3: SUBSTITUTION RULE
RULE 4: STAR RULE
STORE THE RESULTS IN THE INTEGRAL LIST
ASSIGN CONTENTS OF B TO A
IS THE INTEGRATION SEQUENCE FOR A AND B EQUIVALENT
NO
YES
NO
IS THIS THE LAST INTEGRATION
YES
OUTPUT SECTION
Fig. 9. Procedure for integrating the List of Integrands

2. An integrand list section.
3. An integration section.
4. An output section.

The input section consists of reading in the state diagram in the form of flow table, and the output section consists of printing the derived regular expression. Our emphasis, therefore, is placed on the other two sections, the integrand list shown in **Fig. 8,** and the integration section shown in **Fig. 9.**

Before looking at the details of the section on how the integrand list is formed, it is necessary first to define a series of terms which will be used throughout our discussion:

$X = \{0, 1\}$ alphabetic letters.
STATE = $\{q_0, q_1, \ldots, q_{u-1}\}$.
INTEGRATION SEQUENCE = the shortest series of alphabetic letters needed to go from the initial state to the state designated in the word STATE.
COMPLETION LIST = a series of storage locations containing the states which have been placed on the INTEGRAND LIST.
INTEGRAND LIST = a series of storage locations containing an integration sequence and its associated state.

Also, two other words, NULL and LAMBDA are used to denote the null state and lambda state, if they are used. We return now to the discussion of how the integrand list is formed.

After the flow table, our representation of the state diagram, is read into the computer by the SNOBOL4 input procedure, the program transfers to the second section of the program to form the integrand list. The following procedure is used:

Step 1: Upon entry into this section, a check is made to see if the null state is present within the state transition table. This is done by taking each state, seeing if it has an output and if each input alphabetic letter {0, 1} will carry the state back into itself.

(1a) If so, NULL is set to the state and transfer is made to Step 2.

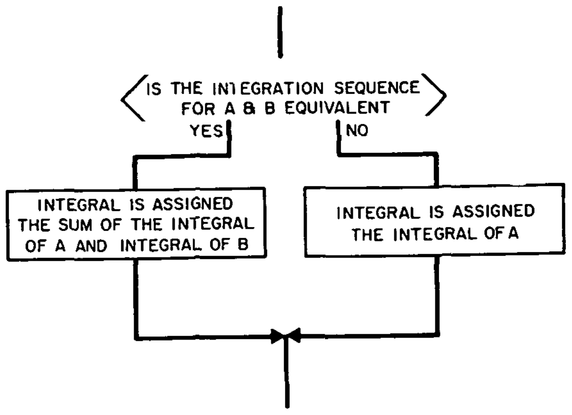

IS THE INTEGRATION SEQUENCE FOR A & B EQUIVALENT
YES
NO
INTEGRAL IS ASSIGNED THE SUM OF THE INTEGRAL OF A AND INTEGRAL OF B
INTEGRAL IS ASSIGNED THE INTEGRAL OF A

Fig. 10. Rule 1. Combinatory rule

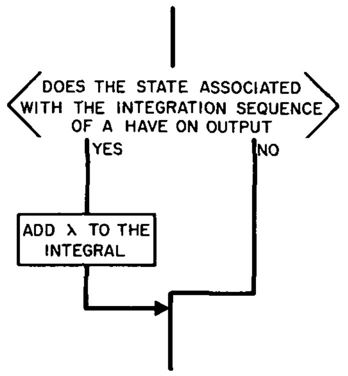

DOES THE STATE ASSOCIATED WITH THE INTEGRATION SEQUENCE OF A HAVE ON OUTPUT
YES
NO
ADD λ TO THE INTEGRAL

Fig. 11. Rule 2. Output rule

(1b) If not, continue.

Step 2: STATE is set to the initial state.

Step 3: X is set to '0'.

Step 4: Under the alphabetic letter denoted by X, a check is made to see if STATE goes into the null state.

- (4a) If so, a check is made to see if X is set to '0'. This is done to see if STATE is possible in the lambda state.
  - (4a, a) If so, a check is made to see if STATE goes to the null state under '1'.
    - (4a,a,a) If so, LAMBDA is set to the contents of STATE and transfer is made to Step 8.
    - (4a,a,b) If not, X is set to '1' and transfer is made to Step 5.
  - (4a,b) If not, transfer is made to Step 8.
- (4b) If not, continue.

Step 5: Next, X is concatenated onto the INTEGRATION SEQUENCE of STATE.

Step 6: This new integration sequence along with its associated state (not the state in STATE) are added to the INTEGRAND LIST.

Step 7: A check is made to see if X is '0'.

- (7a) If so, X is set to '1' and transfer is made to Step 4.
- (7b) If not, continue.

Step 8: Place STATE on the COMPLETION LIST.

Step 9: A check is made to see if the state associated with the next integration sequence has already been done. This is done by taking the state and comparing it with each of the states in the COMPLETION LIST.

- (9a) If so, transfer is made to Step 10.
- (9b) If not, the associated state is assigned to STATE and transfer is made to Step 3.

Step 10: A check is made to see if this is the last entry.

- (10a) If so, transfer is made to the Integration Section.
- (10b) If not, transfer is made to Step 9.

The next section, the integration section, integrates this list of integrands, and manipulates the results according to the four rules of manipulation to form a regular expression. Again before describing the steps followed within this section, we will look at some additional notation which is used:

A = a storage location containing the integral and remaining integration sequence after an integrand has been integrated once, i.e., if the integrand is equal to $D_{s_{m-1}a_i}R = P$ the integral will be equal to $\int D_{s_{m-1}a_i}da_i = D_{s_{m-1}}R = a_iP$ and the remaining integration sequence is s_{m-1}.

B = a second location like A.

INTEGRAL = another location which is used to contain the complete integral of the state which is being worked on.

INTEGRAL LIST = a series of storage locations containing a state and the integral corresponding to that state.

Referring to Fig. 9, we will describe the general procedure for integrating the integrands of the state diagram and the rules of manipulating the derived integrals.

Step 1: Remove the last entry from the INTEGRAND LIST which contains a state and an integration sequence.

Step 2: This integrand is then integrated with respect to the last letter of the integration sequence.

Step 3: The resulting integral and the remaining integration sequences are assigned to A.

Step 4: Steps 1 and 2 are repeated and the results assigned to B.

Step 5: *Combinatory Rule*, **Fig. 10.** A check is made to see if the remaining integration sequence for A and B are equivalent.

- (5a) If so, INTEGRAL is assigned the sum of the integral of A and the integral of B.
- (5b) If not, INTEGRAL is assigned the integral of A only.

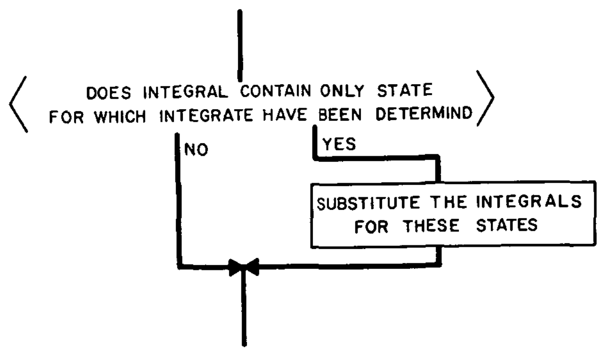

DOES INTEGRAL CONTAIN ONLY STATE FOR WHICH INTEGRATE HAVE BEEN DETERMIND
NO
YES
SUBSTITUTE THE INTEGRALS FOR THESE STATES

Fig. 12. Rule 3. Substitution rule

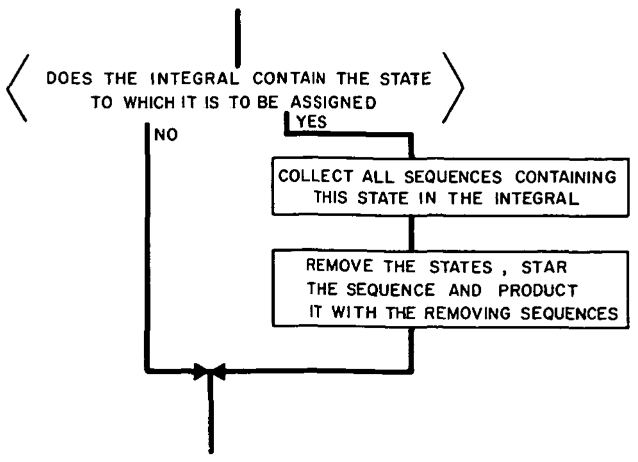

DOES THE INTEGRAL CONTAIN THE STATE TO WHICH IT IS TO BE ASSIGNED
NO
YES
COLLECT ALL SEQUENCES CONTAINING THIS STATE IN THE INTEGRAL
REMOVE THE STATES , STAR THE SEQUENCE AND PRODUCT IT WITH THE REMOVING SEQUENCES

Fig. 13. Rule 4. Star rule

Step 6: *Output Rule*, **Fig. 11.** A check is made to see if the state associated with its remaining integration sequence has an output. This is done by taking the remaining integration sequence, and finding the associated state in the INTEGRAL LIST and checking this state for an output in the state transition state which was read in as input.

(6a) If so, λ is added to the contents of INTEGRAL.

(6b) If not, continue.

Step 7: *Substitution Rule*, **Fig. 12.** A check is made to see if INTEGRAL contains any state for which integrals have already been determined. This is done by taking each state, one at a time, on the INTEGRAL LIST beginning with the first integral which was determined and checking to see if it is in INTEGRAL. This is continued until all the states on the INTEGRAL LIST have been tried.

(7a) If so,

(7a,1) All the sequences containing the state being checked are collected.

(7a,2) The corresponding integral is substituted in for this state.

(7b) If not, continue.

Step 8: *Star Rule*, **Fig. 13.** A check is made to see if INTEGRAL contains the state to which it is to be assigned. This is done by taking the state associated with the remaining integration sequence and checking to see if it occurs in the INTEGRAL.

(8a) If so,

(8a,1) All the sequences containing this state in INTEGRAL are collected and placed as the first sequence in the integral.

(8a,2) The state is removed, and the sequence starred and producted with the remaining sequences.

(8b) If not, continue.

Step 9: Taking the remaining integration sequence in A, find the state associated with it in the INTEGRAND LIST. This state and the contents of INTEGRAL are then placed in the INTEGRAL LIST.

Step 10: Is the remaining integration sequence in A and B equivalent.

(10a) If so, a check is made to see if this is the last integration. This is done by checking to see if both remaining sequences are equal to the null set.

(10a,a) If so, transfer to the Output Section.

(10a,b) If not, transfer to Step 1.

(10b) If not, move the contents of B to A and go to Step 4.

As stated before the output section consists of printing the derived regular expression. A check is also made to see if any more input is waiting. If so, the process begins again. If not, the program terminates.

References

ARDEN, D. N. (1961). Delayed logic and finite state machines, *Proc. 2nd Ann. Symp. on Switching Circuit Theory and Logical Design*, Detroit, Michigan, p. 133.

BRZOZOWSKI, J. A., and MCCLUSKEY, E. J. (1963). Signal flow graph techniques for sequential circuit state diagrams, *IEEE Trans. Elect. Comp.*, Vol. EC-12, p. 67.

BRZOZOWSKI, J. A. (1964). Derivatives of regular expressions, *J. Assoc. Comp. Mach.*, Vol. 11, p. 481.

COPI, I. M., ELGOT, C. L., and WRIGHT, J. B. (1958). Realization of events by logical nets, *J. Assoc. Comp. Mach.*, Vol. 5, p. 181.

KLEENE, S. C. (1954). Representation of events in nerve nets and finite automata, *Automata Studies*, C. E. Shannon and E. J. McCarthy, eds. (Princeton, N.J.: Princeton University Press), study 34, p. 3.

MCNAUGHTON, R., and YAMADA, H. (1960). Regular expressions and state graphs for automata, *IRE Trans. on Elect. Comp.*, Vol. EC-9, p. 39.

Book review

An Analysis of Complexity, by H. van Emden, 1971; 86 pages. (*Amsterdam: Mathematical Centre, Tract* 35, $3.00)

In many branches of science—numerical taxonomy, pattern recognition, artificial intelligence are some examples—classification is a necessary precursor of theoretical study. Often the 'why' comes before the 'how' of classification. A fundamental difficulty is the choice of criteria that distinguish good classifications from bad ones. What might be good for one purpose may indeed not be good for another.

Dr van Emden proceeds to define complexity. A classification is a set of entities into mutually disjoint classes. A subset of the entities, one from each class, is a set of paradigms. If each of the remaining entities is assigned to the same class as its paradigm according to some measure, the classification is perfect if the same result is obtained for every possible set of paradigms. Thus classification depends on similarities or interactions between pairs of entities and between sets of entities. Measures of dissimilarity are defined to be matrics. Complexity is defined as the way in which 'a whole is different from the composition of its parts'. A mathematical definition of interaction in terms of the theory of information. An amount of variety, H, exists in a set so defined that H has the same properties as those which Shannon required the uncertainty of a random variable to have in information theory. Then the amount of complexity C(S) which a system S has is the difference between the sum of the varieties of the individual components of the system itself. This can be related to the interactions between sub-systems of S.

Pairwise interactions between entities to be classified when data is *qualitative* may be used to define a distance function without requiring the qualitative data themselves to constitute a matric space thus allowing a model of classification to be formulated in terms of information. When objects can be described by points in n-dimensional inner-product spaces, the covariance matrix of the set of points can be studied. The author gives a maximum entropy characterisation of the multivariate normal distribution with the aid of which he proposes a measure of the complexity of a covariance matrix. He finds that the condition number of the covariance matrix relates to the complexity.

In a final section he discusses interaction and computational complexity using Jacobi's iteration method for solving linear equations as an example. Here, I think, the author has most success. His work affords an insight into numerical procedures which promises to be valuable. Certainly one can get fresh understanding of processes such as Kron's method of tearing for dealing with large systems, and the various decomposition algorithm of linear programming by applying the author's ideas.

All in all, this little book is a well-written immensely readable introduction to a new and challenging topic.

A. YOUNG (Coleraine)