Derivatives of Regular Expressions

JANUSZ A. BRZOZOWSKI

Princeton University, Princeton, New Jersey†

Abstract. Kleene's regular expressions, which can be used for describing sequential circuits, were defined using three operators (union, concatenation and iterate) on sets of sequences. Word descriptions of problems can be more easily put in the regular expression language if the language is enriched by the inclusion of other logical operations. However, in the problem of converting the regular expression description to a state diagram, the existing methods either cannot handle expressions with additional operators, or are made quite complicated by the presence of such operators. In this paper the notion of a derivative of a regular expression is introduced and the properties of derivatives are discussed. This leads, in a very natural way, to the construction of a state diagram from a regular expression containing any number of logical operators.

1. *Introduction*

In the design of sequential circuits, the first step consists of obtaining an unambiguous description of the circuit behavior. For a certain class of problems, the language of regular expressions [1–10] greatly simplifies this first step of synthesis. In general, the richer the language, the easier it will be to write the problem specification. In this paper we use regular expressions which can have any number of logical connectives, and describe methods for obtaining state diagrams from such regular expressions.

We are concerned with the usual model of a finite automaton M [4, 7, 10, 11, 12]. The n binary inputs $x_1, x_2, \cdots, x_n$ of M are represented by a single 2^n-valued input x, taking the values from $A_k = \{0, 1, \cdots, k-1\}$, where $k = 2^n$. The internal states of M are $q_1, q_2, \cdots, q_m$ and one of these, q_λ, is the starting state of M. The transitions between states are specified by a flow table or by a state diagram. In the Moore model [11] the outputs are associated with internal states and are denoted by Z_j; in the Mealy model [12] the outputs z_j are associated with transitions. The results are presented in terms of a Moore machine; however, the results also apply to Mealy machines with slight modifications which are pointed out where necessary.

2. *Regular Expressions*

We define the following operations on sets of sequences: If P and Q are two sets of sequences of symbols from A_k we have:

This research was supported in part by Bell Telephone Laboratories, Murray Hill, N. J. The majority of the results presented can also be found in Tech. Rep. 15, Princeton University, Dept. of Elec. Eng., Digital Systems Lab., Princeton, N. J., March, 1962.

† Present address: Department of Electrical Engineering, University of Ottawa, Ottawa 2, Canada.

Journal of the Association for Computing Machinery, Vol. 11, No. 4 (October, 1964), pp. 481–494

Product or Concatenation. $(P \cdot Q) = \{s \mid s = pq;\ p \in P, q \in Q\}$. (The dot is omitted for convenience. Also, since the operation is associative we omit parentheses.)

Iterate or Star Operation. $P* = \bigcup_{n=0}^{\infty} P^n$, where $P^2 = PP$, etc., and $P^0 = \lambda$, the set consisting of the *sequence of zero length*, which has the property $R\lambda = \lambda R = R$.

Boolean function. We shall denote any Boolean function of P and Q by $f(P, Q)$. The *empty set* is denoted by ϕ and the *universal set* by I. Thus we have the *complement* P' (with respect to I) of P, the *intersection* $P \,\&\, Q$, the *sum* or *union* $P + Q$, the *modulo-two sum* (exclusive OR) $P \oplus Q$, etc. Of course, all the laws of Boolean algebra apply. The operators *product*, *star* and f are called *regular* operators. For economy of notation, we use the same symbol for a *sequence* s as for the *set of sequences* consisting of only that sequence s.

Definition 2.1. A *regular expression* is defined recursively as follows:

1. The symbols $0, 1, \cdots, k-1, \lambda$ and ϕ are regular expressions.
2. If P and Q are regular expressions, then so are (PQ), $P*$ and $f(P, Q)$, where f is any Boolean function of P and Q.
3. Nothing else is a regular expression unless its being so follows from a finite number of applications of Rules 1 and 2.

The definition is a modification of the definition given by McNaughton and Yamada [3], but is more suitable for our purposes.

The introduction of arbitrary Boolean functions enriches the language of regular expressions. For example, suppose we desire to represent the set of all sequences having three consecutives 1's but not those ending in 01 or consisting of 1's only. The desired expression is easily seen to be

$$R = (I111I)\&(I01+11*)'.$$

3. *Derivatives of Regular Expressions*

We define another operation on a set R of sequences, yielding a new set of sequences called a *derivative* of R.

Definition 3.1. Given a set R of sequences and a finite sequence s, the *derivative of R with respect to s* is denoted by D_sR and is $D_sR = \{t \mid st \in R\}$.

The notion of derivative of a set (under different names) was introduced previously [10, 14, 15], but was not applied to regular expressions. We now present an algorithm for finding derivatives of regular expressions.

We shall need to know when a regular expression contains λ. For this purpose we make the following definition.

Definition 3.2. Given any set R of sequences we define $\delta(R)$ to be

$$\delta(R) = \begin{cases} \lambda \text{ if } \lambda \in R, \\ \phi \text{ if } \lambda \notin R. \end{cases}$$

It is clear that $\delta(a) = \phi$ for any $a \in A_k$, $\delta(\lambda) = \lambda$, and $\delta(\phi) = \phi$. Furthermore $\delta(P*) = \lambda$ (by definition of $P*$), and $\delta(PQ) = \delta(P) \,\&\, \delta(Q)$.

If $R = f(P, Q)$ it is also easy to determine $\delta(R)$. For example,

$$\delta(P + Q) = \delta(P) + \delta(Q), \tag{3.1}$$

$$\delta(P \,\&\, Q) = \delta(P) \,\&\, \delta(Q), \tag{3.2}$$

$$\delta(P') = \begin{cases} \lambda \text{ if } \delta(P) = \phi, \\ \phi \text{ if } \delta(P) = \lambda. \end{cases} \tag{3.3}$$

The δ-function of any other Boolean expression can be obtained using rules (3.1)–(3.3), since the connectives $+$ and $'$ form a complete set of connectives.

THEOREM 3.1. *If R is a regular expression, the derivative of R with respect to a sequence a of unit length ($a \in A_k$) is found recursively as follows:*

$$D_a a = \lambda, \tag{3.4}$$

$$D_a b = \phi, \quad \text{for } b = \lambda \text{ or } b = \phi \text{ or } b \in A_k \text{ and } b \neq a, \tag{3.5}$$

$$D_a(P*) = (D_aP)P*, \tag{3.6}$$

$$D_a(PQ) = (D_aP)Q + \delta(P)D_aQ, \tag{3.7}$$

$$D_a(f(P, Q)) = f(D_aP, D_aQ). \tag{3.8}$$

The proof is given in Appendix I.

THEOREM 3.2. *The derivative of a regular expression R with respect to a finite sequence of input symbols $s = a_1a_2 \cdots a_r$ is found recursively as follows:*

$$Da_1a_2R = Da_2(Da_1R),$$

$$Da_1a_2a_3R = Da_3(Da_1a_2R),$$

$$D_sR = Da_1a_2 \cdots a_rR = Da_r(Da_1a_2 \cdots a_{r-1}R),$$

For completeness, if $s = \lambda$, then $D_\lambda R = R$.

The proof follows from Definition 3.1.

4. *Properties of Derivatives*

It will be shown that the use of derivatives of a regular expression R leads to the construction of a state diagram of a sequential circuit characterized by R, in a very natural way. First, however, let us investigate some of the properties of derivatives.

THEOREM 4.1. *The derivative D_sR of any regular expression R with respect to any sequence s is a regular expression.*

PROOF. It is clear from Theorem 3.2 that $D_\lambda R$ is regular and that D_sR is regular (for s of length greater than 1) if D_aR is regular (where a is a sequence of length 1). The construction of Theorem 3.1 shows that D_aR is regular, since only a finite number of regular operations is required to find the derivative.

THEOREM 4.2. *A sequence s is contained in a regular expression R if and only if λ is contained in D_sR.*

PROOF. If $\lambda \in D_sR$, then $s\lambda = s \in R$ from Definition 3.1. Conversely, if $s \in R$, then $s\lambda \in R$ and $\lambda \in D_sR$, again from Definition 3.1.

Theorem 4.2 reduces the problem of testing whether a sequence s is contained in a regular expression R to the problem of testing whether λ is contained in D_sR. The latter problem is solved through the use of $\delta(D_sR)$.

Two regular expressions which are equal (but not necessarily identical in form) will be said to be of the same *type*.

THEOREM 4.3. (*a*) *Every regular expression R has a finite number d_R of types of derivatives.* (*b*) *At least one derivative of each type must be found among the derivatives with respect to sequences of length not exceeding $d_R - 1$.*

PROOF. The proof of Part (a) of the theorem is given in Appendix II. Part (b) of Theorem 4.3 indicates a method of finding all the d_R different types of derivatives, which will be called the *characteristic* derivatives of R. The sequences of symbols from A_k can be arranged in order of increasing length; for example, for $A_2 = \{0, 1\}$, we have λ, 0, 1, 00, 01, 10, 11, 000, $\cdots$.

The derivatives are now found in the above order, i.e. $D_\lambda R$, D_0R, D_1R, $\cdots$. If for sequences s of length $L(s) = r$ no new types of derivatives are found, the process terminates. For, if no new derivatives are found for $L(s) = r$, then every D_sR is equal to (of the same type as) another derivative D_tR, where $L(t) < r$. Consider $D_{sa}R = D_a(D_sR) = D_a(D_tR) = D_{ta}R$, where $a \in A_k$ and $L(ta) < (r+1)$. Thus every derivative with respect to a sequence sa of length $r+1$ will be equal to some derivative with respect to a sequence ta of length $L(ta) < (r+1)$. Hence, if no new types of derivatives are found for $L(s) = r$, then no new types will be found for $L(s) = r+1$, etc. Therefore at least one new type of derivative must be found for each $L(s)$ or, otherwise, the process terminates.

In the above discussion it is assumed that it is possible to decide when two derivatives are of the same type. This is not always an easy problem but it can be resolved, as is shown in Section 5.

THEOREM 4.4. *Every regular expression R can be written in the form*

$$R = \delta(R) + \sum_{a \in A_k} aD_aR, \tag{4.1}$$

where the terms in the sum are disjoint.

PROOF. First, R may or may not contain λ; this is taken care of by $\delta(R)$. If R contains a sequence s, then that sequence must begin with a letter $a \in A_k$ of the input alphabet. In view of the definition of derivative, the set aD_aR is exactly the set of sequences of R beginning with a. The terms of the sum are obviously disjoint, for the sequences in one term begin with a letter of A_k different from those in another term.

It follows from Theorem 4.4 that every regular expression can be represented by an infinite sum $R = \sum_{s \in I} sD_sR$. The number of types of derivatives is of course finite and so the series is redundant (e.g. the set of sequences beginning with ab is contained in the set of sequences beginning with a). The expansion (4.1) is much more useful, as will be shown below.

THEOREM 4.5. *The relationship between the d_R characteristic derivatives of R can be represented by a unique set of d_R equations of the form*

$$D_sR = \delta(D_sR) + \sum_{a \in A_k} aD_{u_a}R,$$

where D_sR is a characteristic derivative and $D_{u_a}R$ is a characteristic derivative equal to $D_{sa}R$. Such equations will be called the characteristic equations *of R.*

The theorem follows directly from Theorems 4.3 and 4.4.

THEOREM 4.6. *An equation of the form $X = AX + B$, where $\delta(A) = \phi$, has the solution $X = A * B$, which is unique (up to equality of regular expressions).*

The theorem is a modification of a theorem of Arden [8], who has shown that $X = XA + B, \delta(A) = \phi$, has the solution $X = BA*$. The proof of Theorem 4.6 parallels that of Arden's theorem and will not be given here.

THEOREM 4.7. *The set of characteristic equations of R can be solved for R uniquely (up to equality).*

PROOF. The proof follows from Theorem 4.6. The last characteristic derivative can be found in terms of the previous derivatives. Note that the coefficient A in the equation for any derivative is either ϕ or it is one or more symbols of the input alphabet. Thus $\delta(A) = \phi$ in all cases and Theorem 4.6 applies. Next, the solution for the last derivative can be substituted in the first $(d_R - 1)$ equations, reducing the number of equations by 1. The process is repeated until the set of equations is solved for $D_\lambda R = R$.

Thus the regular expression can be always reconstructed from the characteristic equations (although it may be in a different form, depending on the order of elimination of derivatives from the equation).

5. *State Diagram Construction*

Definition 5.1. A sequence s is *accepted by an automaton M* with starting state q_λ iff when s is applied to M in q_λ the output is 1 at the end of s. Otherwise, s is rejected by M. A sequence s is *accepted by a state q_j* of M iff when M is started in q_j the output is 1 at the end of s.

Two states q_j and q_k of M are *indistinguishable* [11] iff every sequence s applied to M started in q_j produces the same output sequence as that produced by applying s to M started in q_k.

THEOREM 5.1 [7]. *Two states q_j and q_k of M are indistinguishable iff R_j and R_k denoting the sets of sequences accepted by q_j and q_k are equal.*

PROOF. (The theorem has been proved by Lee [7] in a more general form.) If q_j and q_k are indistinguishable then M started in q_j and M started in q_k produce identical output sequences for any input sequence. In particular, the sets of sequences ending in $Z = 1$ must be identical, i.e. $R_j = R_k$. Now suppose that $R_j = R_k$ but q_j and q_k are distinguishable. Then there must exist an s producing different output sequences. Consider the first position in which the output sequences differ; clearly, the initial portion of s up to and including that position is accepted in one case and rejected in the other, contradicting $R_j = R_k$.

It is clear that, if a sequence s takes M (which accepts R) from q_λ to q_j, then the regular expression R_j is simply the derivative of R with respect to s. If we take the *first* s that takes M from q_λ to q_j, then with each state we can associate a unique derivative with respect to that first s. Theorem 5.1 can now be restated as follows.

THEOREM 5.1(a). *Two states q_j and q_k of an automaton M characterized by the regular expression R are indistinguishable iff their derivatives are equal,* i.e. $D_{s_j}R = D_{s_k}R$, *where s_j and s_k are any two sequences taking M from state q_λ to q_j and q_k respectively.*

Thus by the arguments presented in this and the preceding sections we have established a very close relationship between derivatives of a regular expression and the states of the corresponding finite automaton. These results are summarized as follows:

1. To obtain a regular expression from a state diagram or a flow table, write the set of characteristic equations and solve for R. There is one equation for each state. If the transition under input a takes the graph from q_j to q_k then the equation for R_j contains the term aR_k; and λ is a term of R_j if and only if the output for q_j is $Z = 1$.

There are other methods [3, 5, 8] of obtaining the regular expressions, but this one is most closely related to the derivative approach.

2. To obtain the minimal state diagram from a regular expression, find the characteristic derivatives and associate one internal state with each characteristic derivative. The output associated with a state is $Z = 1$ iff the corresponding characteristic derivative contains λ.

This procedure is illustrated in the following example.

Let $R = (0 + 1)*1$. Then

$D_\lambda = R$; introduce q_λ with $Z = 0$, for $\delta(R) = \phi$,

$D_0 = R$; return to q_λ under input 0,

$D_1 = R + \lambda$; introduce q_1, with output $Z = 1$ (for $\delta(D_1) = \lambda$), and a transition from q_λ to q_1 under input $x = 1$. D_{00}, D_{01} need not be considered, for D_0 does not correspond to a new characteristic derivative, i.e. 0 returns the state graph to q_λ. Continuing, we find

$D_{10} = R$; go from q_1 to q_λ under $x = 0$,

$D_{11} = R + \lambda$; return from q_1 to q_1 under $x = 1$.

This completes the process. The resulting state diagram is shown in Figure 1(a).

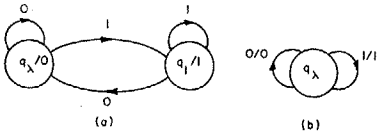

FIG. 1. State graphs for $R = (0 + 1)*1$ (a) Moore model (b) Mealy model

With a very minor modification in the method, a Mealy state diagram can be constructed from any regular expression. The process in this case is identical except that when two derivatives are compared the presence of λ in a derivative

is ignored. In other words, if $D_uR = A$, $A \not\supset \lambda$ and $D_vR = A + \lambda$, then D_uR and D_vR correspond to the same state. This is a consequence of the definition of a derivative: If $D_sR \supset \lambda$, this tells us that s is contained in R, i.e. accepted by the automaton corresponding to R. In the Mealy model this means that the transition caused by the last symbol of the sequence s has been accompanied by the output $z = 1$. However (if the state reached by the application of s is q_s), then the presence of λ in D_sR tells us nothing about the future sequences accepted by the state q_s and hence should be ignored. Thus the Mealy diagram can have a fewer number of states, in general.

The construction of a Mealy diagram can be illustrated by the same example. For the expression $R = (0 + 1)*1$ we have

$D_\lambda = R$; introduce q_λ,
$D_0 = R$; return to q_λ with $z = 0$,
$D_1 = R + \lambda$; return to q_λ with $z = 1$.

The construction terminates here, since no new states are found for sequences of length 1. The diagram is shown in Figure 1(b) and consists of one state, whereas the corresponding Moore model has two states.

The characteristic equations can be obtained from a Mealy model in a way very similar to that for the Moore model. If the transition caused by input a from state q_j to state q_k has an output $Z = 1$ then the equation for R_j contains

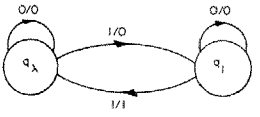

FIG. 2. State graph to be analyzed

the term $a(R_k + \lambda) = aR_k + a$; if $Z = 0$ the term is just aR_k. For example for the state diagram of Figure 2 we obtain the equations

$$R_\lambda = 0R_\lambda + 1R_1 \quad \text{and} \quad R_1 = 0R_1 + 1(R_\lambda + \lambda).$$

Solving these equations we obtain

$$R_1 = 0*1R_\lambda + 0*1; \qquad R_\lambda = (0 + 10*1)R_\lambda + 10*1.$$

Hence $R_\lambda = (0 + 10*1)*10*1$.

In the above discussion we have assumed that it is always possible to recognize the equality of two regular expressions. If this is the case, then the state diagram constructed by associating one internal state with each type of derivative is always minimal. However, it is often quite difficult to determine whether two regular expressions are equal. We now show that this difficulty can be overcome, and a state graph can always be constructed, but not necessarily with the minimum number of states. It should be pointed out that the other existing methods [3, 6] have the same difficulties and, moreover, are limited to regular expressions with $(+)$, $(\cdot)$ and $(*)$ only.

Definition 5.2. Two regular expressions are *similar* if one can be transformed

to the other by using only the identities:

$$R + R = R,$$
$$P + Q = Q + P,$$
$$(P + Q) + R = P + (Q + R).$$

Two regular expressions are *dissimilar* iff they are not similar.

It is clear that similarity implies equality, but equality in general does not imply similarity. Similarity can be easily recognized and is much weaker than equality.

THEOREM 5.2. *Every regular expression has only a finite number of dissimilar derivatives.*

PROOF. The proof is given in Appendix II. As a consequence of this result, *a state diagram can be constructed even if only similarity among the derivatives is recognized.* However, such a method has a serious disadvantage since, in general, the diagram so constructed will be far from minimal. This arises because of the frequent appearance of λ and ϕ in the derivatives. For example, consider $R = (0 + 1)*(01)$ and the derivative D_1R.

$$\begin{aligned} D_1R &= (D_1((0 + 1)*))\ (01) + D_1(01) \\ &= ((D_1(0 + 1))(0 + 1)*)\ (01) + (D_10)1. \\ &= ((D_10 + D_11)(0 + 1)*)\ (01) + (D_10)1 \\ &= ((\phi + \lambda)(0 + 1)*)\ (01) + \phi 1. \end{aligned}$$

In this case, using only similarity, we are forced to conclude that R and D_1R are dissimilar. However, the expression for D_1R can be easily simplified by the identities

$$R + \phi = \phi + R = R, \quad R\phi = \phi R = \phi, \quad R\lambda = \lambda R = R. \qquad (5.1, 5.2, 5.3)$$

The expression for D_1R then becomes

$$D_1R = (\lambda(0 + 1)*)\ (01) + \phi = (0 + 1)*(01) = R.$$

The identities (5.1)–(5.3) are thus very useful and will be incorporated in the method.

We conclude this section with a more complicated example. Let it be desired to have an output if the input sequence contains two consecutive 0's but does not end in 01. The required regular expression is $R = (I00I)\ \&\ (I01)' = P\ \&\ Q'$, where $I = (0 + 1)*$, $P = I00I$, $Q = I01$. The construction of derivatives proceeds as follows:

$D_\lambda = R = P\ \&\ Q'$; introduce q_λ,
$D_0 = (P + 0I)\ \&\ (Q + 1)'$; introduce q_0,
$D_1 = P\ \&\ Q'$; return to q_λ,
$D_{00} = (P + 0I + I)\ \&\ (Q + 1)'$, introduce q_{00}.

Here we note that $I + X = I$ and $I\ \&\ X = X$. Hence we may write D_{00} in a

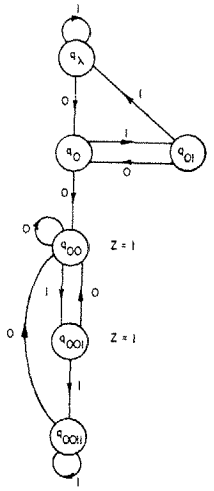

FIG. 3. State diagram for the example

simpler form:

$$
\begin{aligned}
D_{00} &= (Q + 1)', \\
D_{01} &= P \,\&\, (Q + \lambda)'; && \text{introduce } q_{01}, \\
D_{000} &= (Q + 1)'; && \text{return to } q_{00}, \\
D_{001} &= (Q + \lambda)'; && \text{introduce } q_{001}, \\
D_{010} &= (P + 0I) \,\&\, (Q + 1)'; && \text{return to } q_0, \\
D_{011} &= P \,\&\, Q'; && \text{return to } q_\lambda, \\
D_{0010} &= (Q + 1)'; && \text{return to } q_{00}, \\
D_{0011} &= Q'; && \text{introduce } q_{0011}, \\
D_{00110} &= (Q + 1)'; && \text{return to } q_{00}, \\
D_{00111} &= Q'; && \text{return to } q_{0011}.
\end{aligned}
$$

This concludes the construction. The characteristic derivatives are D_λ, D_0, D_{00}, D_{01}, D_{001} and D_{0011}. Therefore the state diagram has 6 states shown in Figure 3. One has to determine $\delta(D_s)$ to find the output associated with q_s. In this case only D_{00} and D_{0011} contain λ; hence the output is $Z = 1$ only for q_{00} and q_{0011}.

Upon examining the state diagram it is seen that states q_λ and q_{01} are indistinguishable and that the reduced state diagram contains only 5 states. We have failed to discover this because we have failed to recognize the equivalence $D_{01} = P \,\&\, (Q + \lambda)' = P \,\&\, Q' \,\&\, \lambda' = P \,\&\, Q' = D_\lambda$.

6. *Regular Expressions for Multiple-Output Circuits*

The behavior of a multiple-output sequential circuit can be represented by specifying one regular expression for each output. Thus, for each output Z_i we have a corresponding regular expression R_i : any sequence of R_i results in $Z_i = 1$; the sequences not contained in R_i result in $Z_i = 0$. The expression R_i carries

no information about any of the other outputs; consequently, each R_i can be specified independently.

In this manner, we can describe the action of a sequential circuit with r outputs by an ordered r-tuple of regular expressions. For convenience, we refer to such r-tuples as *regular vectors*, $\mathbf{R} = (R_1, R_2, \cdots, R_r)$, where the R_i are the components of the vector $\mathbf{R}$. The corresponding outputs at any time can be denoted by $\mathbf{Z} = (Z_1, Z_2, \cdots, Z_r)$, where the Z_i are binary variables.

Definition 6.1. The *derivative of a vector* $\mathbf{R}$ *of regular expressions, with respect to a sequence* s, is a vector of regular expressions denoted by $\mathbf{D}_s\mathbf{R}$ and defined by $\mathbf{D}_s\mathbf{R} = (D_sR_1, D_sR_2, \cdots, D_sR_r)$.

Definition 6.2. Two regular vectors $\mathbf{P}$ and $\mathbf{Q}$, with r components each, are equal, $\mathbf{P} = \mathbf{Q}$, iff their components are equal, i.e. $P_i = Q_i$ for all i.

Since every regular expression has only a finite number of types of derivatives, it follows that every regular vector has a finite number of types of derivatives. Consequently, given a regular vector $\mathbf{R}$, we can construct a state diagram (Moore model), by associating one derivative per state, beginning with $\mathbf{D}_\lambda\mathbf{R} = \mathbf{R}$. The derivatives are constructed for sequences in increasing order, as in the single output case. An output Z_i is 1 iff $\lambda \in D_sR_i$.

Note that, for a multiple-output circuit, the behavior could be described by a set of r state diagrams, one for each regular expression. We are interested in constructing a *single* state diagram which will produce the correct r-tuple of outputs. The result of the state diagram construction described above can be summarized as follows.

THEOREM 6.1. *Given a regular vector* $\mathbf{R}$, *if a state diagram is constructed by associating one type of derivative of* $\mathbf{R}$ *per state, that state diagram represents the desired behavior and is minimal.*

PROOF. Given a regular vector $\mathbf{R} = (R_1, R_2, \cdots, R_r)$, a state diagram is desired with the property that $Z_i = 1$, as a result of applying a sequence s iff $s \in R_i$. By construction, the proposed state diagram will go from the starting state q_λ to state q_s, associated with $\mathbf{D}_s\mathbf{R}$ or with the equivalent previous derivative. Also by construction, the output Z_i (associated with state q_s) will be 1 iff $\lambda \in D_sR_i$, i.e. $s \in R_i$. Thus it is clear that every sequence will produce the desired output vector. To prove the minimality of the state diagram, we note that there must be a distinct state for each distinct derivative. If two derivatives

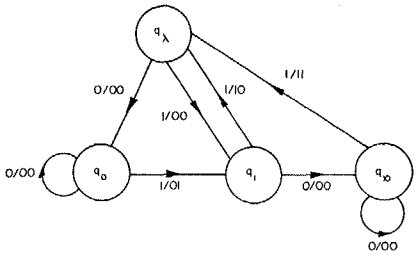

FIG. 4. State diagram for $\mathbf{R} = (R_1, R_2)$

$\mathbf{D}_s\mathbf{R}$ and $\mathbf{D}_t\mathbf{R}$ are not equal, they must differ in at least one component, say $D_sR_j \neq D_tR_j$. Hence $\mathbf{D}_s\mathbf{R}$ and $\mathbf{D}_t\mathbf{R}$ cannot correspond to the same state, for the output Z_j would be incorrect for some sequence which is in D_sR_j but not in D_tR_j or vice versa. Therefore, since it is necessary and sufficient to have one distinct state per distinct derivative, the state diagram is minimal.

The construction of a Mealy state diagram is identical except that two derivatives (vectors) differing by a vector with components ϕ or λ only, can be associated with a single state.

Example. $\mathbf{R} = (R_1, R_2) = ((0 + 10*1)*10*1, (0 + 1)*01)$.

We now construct the Mealy state diagram for this two-output circuit.

$D_\lambda = (R_1, R_2)$; introduce q_λ,
$D_0 = (R_1, R_2 + 1)$; $q_0, z = (0, 0)$,
$D_1 = (0*1R_1 + 0*1, R_2)$; $q_1, z = (0, 0)$,
$D_{00} = (R_1, R_2 + 1)$; to $q_0, z = (0, 0)$,
$D_{01} = (0*1R_1 + 0*1, R_2 + \lambda)$; to $q_1, z = (0, 1)$,
$D_{10} = (0*1R_1 + 0*1, R_2 + 1)$; $q_{10}, z = (0, 0)$,
$D_{11} = (R_1 + \lambda, R_2)$; to $q_\lambda, z = (1, 0)$,
$D_{100} = (0*1R_1 + 0*1, R_2 + 1)$; to $q_{10}, z = (0, 0)$,
$D_{101} = (R_1 + \lambda, R_2 + \lambda)$; to $q_\lambda, z = (1, 1)$.

The state diagram is shown in Figure 4.

7. *Conclusion*

Regular expressions can be obtained more easily from word description of problems if one is allowed to use any logical connective in the formation of the expression. We have introduced here the notion of a derivative of a regular expression as a powerful aid in analyzing the properties of regular expressions with arbitrary logical connectives. The derivative approach leads naturally to state diagrams of sequential circuits and has been extended to cover the multiple output case.

Acknowledgment. The author wishes to thank Professor E. J. McCluskey, J. F. Poage, E. B. Eichelberger and S. O. Chagnon of Princeton University for their comments and suggestions.

APPENDIX I. PROOF OF THEOREM 3.1.

The proof follows for relations (3.4)–(3.8), for finding the derivative of R with respect to a sequence $a \in A_k$ of unit length.

By definition 3.1, $D_aR = \{t \mid at \in R\}$. Then relations (3.4) and (3.5) are obvious. Thus the theorem holds for regular expressions involving no regular operators.

Let us consider now (3.8). It is sufficient to prove this relation for $f(P, Q) = P + Q$ and for $f(P, Q) = P'$, for this is a complete set of Boolean connectives. Now

$$\begin{aligned} D_a(P + Q) &= \{t \mid at \in (P + Q)\} \\ &= \{u \mid au \in P\} + \{v \mid av \in Q\} \\ &= D_aP + D_aQ. \end{aligned}$$

It is clear that this rule can be extended to any number of regular expressions, i.e. that $D_a(R_1 + R_2 + \cdots) = D_aR_1 + D_aR_2 + \cdots$ even when the number of R_i is countably infinite. Next, note that $aD_aR + aD_a(R') = aI$. Taking the derivative with respect to a of both sides, we have $D_aR + D_a(R') = I$. Also $(D_aR)\ \&\ (D_a(R')) = \phi$ and we have $D_a(R') = (D_aR)'$. Thus rule (3.8) holds for union and complementation, and consequently for any Boolean function.

Next consider $D_a(PQ)$. Let $P = \delta(P) + P_0$, where $\delta(P_0) = \phi$. Then

$$\begin{aligned} D_a(PQ) &= \{s \mid as \in (\delta(P) + P_0)Q\} \\ &= \{u \mid au \in \delta(P)Q\} + \{v \mid av \in P_0Q\} \\ &= \delta(P)D_aQ + \{v_1v_2 \mid av_1 \in P_0, v_2 \in Q\} \\ &= \delta(P)D_aQ + \{v_1 \mid av_1 \in P_0\}Q \\ &= \delta(P)D_aQ + (D_aP_0)Q. \end{aligned}$$

But $D_aP = D_a(P_0 + \lambda) = D_aP_0$; hence $D_a(PQ) = \delta(P)D_aQ + (D_aP)Q$, which is rule (3.7).

Finally, we have

$$\begin{aligned} D_aP* &= D_a(\lambda + P + PP + PPP + \cdots) \\ &= D_a\lambda + D_aP + D_aP^2 + \cdots + D_aP^n + \cdots. \end{aligned}$$

But

$$\begin{aligned} \sum_{n=1}^{\infty} D_aP^n &= \sum_{n=1}^{\infty} ((D_aP)P^{n-1} + \delta(P)D_aP^{n-1}) \\ &= \sum_{n=1}^{\infty} (D_aP)P^{n-1}, \end{aligned}$$

since $\delta(P)D_a{}^{n-1}$ is either ϕ or it is D_aP^{n-1}, which is already included. Thus we have

$$D_a\,P* = \sum_{n=1}^{\infty} (D_a\,P)P^{n-1} = (D_a\,P) \sum_{n=1}^{\infty} P^{n-1} = (D_a\,P)P*,$$

which is rule (3.6). This concludes the proof of Theorem 3.1.

APPENDIX II. PROOFS OF THEOREMS 4.3(a) AND 5.2

Theorem 4.3 is proved by induction on a number N of regular operators.

BASIS, $N = 0$. The theorem is certainly true when R is one of ϕ, λ or $a \in A_k$, for we have

$$D_s\phi = \phi \text{ for all } s \in I,$$

$$D_\lambda\lambda = \lambda, \text{ and } D_s\lambda = \phi \text{ for all } s \in I,\ s \neq \lambda.$$

$$D_\lambda a = a. \quad D_a a = \lambda. \quad D_s a = \phi \text{ for all } s \in I,\ s \neq \lambda,\ a.$$

Thus we have $d_\phi = 1$, $d_\lambda = 2$ and $d_a = 3$.

INDUCTION STEP, $N > 0$. Assume that each expression X with N or fewer operators has a finite number d_X of derivatives. If R is an expression with $N + 1$ operators, there are three cases.

Case 1. $R = f(PQ)$. It is easily verified from the definitions that $D_sR = D_s(P + Q) = D_sP + D_sQ$. Thus $d_R \leqq d_P d_Q$. If $R = P'$ then $D_sR = (D_sP)'$. In this case, $d_R = d_P$. Since any Boolean function can be expressed using a finite number of sums and complements, it follows that the number of derivatives of R (of the form $R = f(P, Q)$) is finite.

Case 2. $R = PQ$. Let $s = a_1a_2 \cdots a_r$. Using the definitions of Section 3, we have $D_{a_1}R = (D_{a_1}P)Q + \delta(P)D_{a_1}Q$. Similarly, for a sequence of length 2, we have $D_{a_1a_2}R = (D_{a_1a_2}P)Q + \delta(D_{a_1}P)D_{a_2}Q + \delta(P)D_{a_1a_2}Q$. In general, the derivative with respect to a sequence of length r will have the form

$$\begin{aligned} D_{a_1\cdots a_r}R = {} & (D_{a_1\cdots a_r}P)Q + \delta(D_{a_1\cdots a_{r-1}}P)D_{a_r}Q \cdots \\ & + \delta(D_{a_1}P)D_{a_2\cdots a_r}Q + \delta(P)D_{a_1\cdots a_r}Q. \end{aligned} \tag{II.1}$$

Thus D_sR is the sum of $(D_sP)Q$ and of at most r derivatives of Q. If there are d_P and d_Q types of derivatives of P and Q respectively, there can be at most $d_R \leqq d_P 2^{d_Q}$ types of derivatives of R. (Note that we are finding upper bounds to show the finiteness of d_R, but such bounds do not necessarily have to be achieved.) Hence the inductive step holds for this case also.

Case 3. $R = P*$. Again let us consider the formation of the derivative of P^*. We have

$$\begin{aligned} D_{a_1}(P*) &= (D_{a_1}P)P*, \\ D_{a_1a_2}(P*) &= (D_{a_1a_2}P)P* + \delta(D_{a_1}P)D_{a_2}(P*) \\ &= (D_{a_1a_2}P)P* + \delta(D_{a_1}P)(D_{a_2}P)P*, \text{ etc.} \end{aligned}$$

It can be seen that, in general, D_sR will be the sum of terms of the form $D_t(P)P*$. If P has d_P types of derivatives, then R has at most $d_R \leqq 2^{d_P} - 1$ types of derivatives. This concludes the inductive step.

To prove Theorem 5.2 we must demonstrate that the process of constructing the derivatives will terminate after a finite number of steps, even if only similarity of regular expression is recognized. This result is actually implicit in the proof of Theorem 4.3, but we shall explain it now in more detail.

The result is obvious for the basis step. Now suppose the given R is of the form $R = f(P, Q)$. Then from Case 1 above we have $D_sR = f(D_sP, D_sQ)$, for this result holds for $+$ and $'$. Now, as s takes on all possible values, compare D_sR with all the previously found derivatives. Since D_sP and D_sQ will appear in

$f(D_sP, D_sQ)$ only a finite number of times and the structure of f is fixed, and furthermore, P and Q have a finite number of derivatives, the process will clearly terminate.

Proceeding with Case 2 above, if $R = PQ$ we can write D_sR in the form of (II.1). Here, however we cannot use the argument of Case 1 since the number of terms in (II.1) increases with the length of s. The associative law for sum has been used in (II.1) to remove parentheses. The commutative law for sum can be used to recognize two sums in which the terms appear in different orders. But it is the identity $R + R = R$ which allows us to terminate the process. Note that the inequality $d_R \leqq d_P 2^{d_Q}$ resulting from (II.1) does not depend on the length of s. Thus each new derivative must be simplified using $R + R = R$ and then compared with the previous derivatives.

Finally, the same argument is applicable to $R = P*$ and hence the theorem holds.

REFERENCES

1. KLEENE, S. C. Representation of events in nerve nets and finite automata. In *Automata Studies*, Ann. Math. Studies No. 34, Princeton U. Press, 1956, 3–41.
2. COPI, I. M.; ELGOT, C. C.; AND WRIGHT, J. B. Realization of events by logical nets. *J. ACM 5* (Apr. 1958), 181–196.
3. MCNAUGHTON, R. AND YAMADA, H. Regular expressions and state graphs for automata. *IRE Trans. EC-9* (Mar. 1960), 39–47.
4. BRZOZOWSKI, J. A. A survey of regular expressions and their applications. *IRE Trans. EC-11* (June 1962), 324–335. (Also Tech. Rep. 4, Princeton U., Digital Systems Lab., Apr. 1961).
5. BRZOZOWSKI, J. A. AND MCCLUSKEY, E. J. JR. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Trans. EC-12* (Apr. 1963), 67–76. (Also Tech. Rep. 5, Princeton U., Digital Systems Lab., Apr. 1961).
6. OTT, G. H. AND FEINSTEIN, N. H. Design of sequential machines from their regular expressions, *J. ACM 8* (Oct. 1961), 585–600.
7. LEE, C. Y. Automata and finite automata. *Bell System Tech. J. 39* (Sept. 1960), 1267–1295.
8. ARDEN, D. N. Delayed logic and finite state machines. In *Theory of Computing Machine Design*, pp. 1–35. U. of Michigan Press, Ann Arbor. 1960.
9. MYHILL, J. Finite automata and representation of events. WADC, Tech. Rep. 57–624, 1957.
10. RABIN, M. O. AND SCOTT, D. Finite automata and their decision problems. *IBM J. Res. Develop. 3* (Apr. 1959), 114–125.
11. MOORE, E. F. Gedanken experiments on sequential machines. In *Automata Studies*, Ann. of Math. Studies No. 34, Princeton U. Press, 1956, 129–153.
12. MEALY, G. H. A method for synthesizing sequential circuits. *Bell System Tech. J. 34* (Sept. 1955), 1045–1079.
13. HUFFMAN, D. A. The synthesis of sequential switching circuits. *J. Franklin Inst. 257* (Mar., Apr. 1954), 161–190, 275–303.
14. RANEY, G. N. Sequential functions. *J. ACM 5* (Apr. 1958), 177.
15. ELGOT, C. C. AND RUTLEDGE, J. D. Operations on finite automata. Proc. AIEE Second Ann. Symp. on Switching Circuit Theory and Logical Design, Detroit, Mich., Oct. 1961.

RECEIVED APRIL, 1963; REVISED NOVEMBER, 1963.