A Survey of Regular Expressions and Their Applications*

JANUSZ A. BRZOZOWSKI†, STUDENT MEMBER, IRE

Summary—This paper is an exposition of the theory of regular expressions and its applications to sequential circuits. The results of several authors are presented in a unified manner, pointing out the similarities and differences in the various treatments of the subject. Whenever possible, the terminology and notation of sequential circuit theory are used. The topics presented include: the relation of regular expressions to sequential circuits; algorithms for constructing sequential circuits and state diagrams corresponding to a given regular expression; methods for obtaining a regular expression from a state diagram of a sequential circuit, improper state diagrams, algebraic properties of regular expressions, and applications to codes.

I. Introduction

IN 1943 McCULLOCH and Pitts[1] proposed a logical theory for describing the behavior of nerve nets. These ideas were further generalized by Kleene[2] who, in 1956, showed that regular expressions can be applied to abstract automata. One of his most important results is the theorem that any finite-state, deterministic, synchronous automaton can be characterized by a regular expression and that every regular expression can be realized by such an automaton. However, Kleene's exposition was mostly in terms of nerve nets and was rather complicated. In order to clarify some of his results. Copi, Elgot and Wright[3] published an expository paper in 1957, but restricted their discussion to the so-called instantaneous logic. A further addition to the theory of regular expressions was presented in 1960 by McNaughton and Yamada,[4] who gave algorithms showing the connection between regular expressions and state graphs, and introduced the notion of an extended regular expression with intersection and complementation. Lee[5] has considered finite automata as a subfamily of Turing machines and has also presented methods for relating regular expressions and state graphs. The paper by Arden[6] in the summer of 1960, contained a discussion of both instantaneous logic and delayed logic in a language more closely related to that of sequential circuit theory. A different treatment of the relation between regular expressions and state graphs was also presented. Myhill[7] and Rabin and Scott[8] have also pursued and simplified some of the problems raised by Kleene.[2]

Although several of the above mentioned works are partly or wholly expository in nature, the terminology and the initial assumptions vary considerably from one paper to another. Therefore, there appears to be a need for a unified presentation of the ideas and results obtained in the references given above, and it is hoped that this paper will achieve that purpose.

Regular expressions will be used to characterize the external behavior of a certain class of sequential circuits.[9–11] The discussion will be, in general, independent of the type of circuitry used to realize the desired input-output characteristics and hence the problem can be stated in terms of abstract automata. The term "sequential circuit" will be used to describe a physical circuit which performs the action specified by the behavior of the corresponding abstract "automaton," although sometimes the terms will be used interchangeably.

The advantages[4] of using the regular expression language are the following:

1) The method is quite general because it applies to the entire class of pulse-mode and clocked sequential circuits.
2) In many cases the language is more closely related to the word description of a sequential circuit than other methods of characterization.
3) The language is precisely and formal, and permits no ambiguities.
4) The description can be written in one line in contrast to stage graphs or tables.

* Received November 17, 1961; revised manuscript received, March 16, 1962. The research reported was supported in part by Bell Telephone Labs., Murray, Hill, N. J.; also Tech. Rept. 4, Princeton University, Dept. of Elec. Engrg., Digital Systems Lab., Princeton, N. J.

† Department of Electrical Engineering, Princeton University, Princeton, N. J.

[1] W. S. McCulloch, and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," Bull. Math. Biophys., vol. 5, pp. 115–133; 1943.

[2] S. C. Kleene, "Representation of events in nerve nets and finite automata," in "Automata Studies, Annals of Math. Studies," C. E. Shannon and J. McCarthy, Eds., Princeton University Press, Princeton, N. J., no. 34, pp. 3–41; 1956.

[3] I. M. Copi, C. L. Elgot, and J. B. Wright, "Realization of events by logical nets," *J. ACM*, vol. 5, pp. 181–196; April, 1958.

[4] R. McNaughton and H. Yamada, "Regular expressions and state graphs for automata," IRE TRANS. ON ELECTRONIC COMPUTERS, vol. EC-9, pp. 39–47; March, 1960.

[5] C. Y. Lee, "Automata and finite automata," *Bell Sys. Tech. J.*, vol. 39, pp. 1267–1295; September, 1960.

[6] D. N. Arden, "Delayed logic and finite state machines," in "Theory of Computing Machine Design," University of Michigan Press, Ann Arbor, pp. 1–35; 1960.

[7] J. Myhill, "Finite Automata and Representation of Events," WADC, Tech. Rept. 57-624; 1957.

[8] M. O. Rabin, and D. Scott, "Finite automata and their decision problems," *IBM J. Res. & Dev.*, vol. 3, pp. 114–125; April, 1959.

[9] E. F. Moore, "Gedanken experiments on sequential machines," in "Automata Studies, Annals of Math. Studies," C. E. Shannon and J. McCarthy, Eds., Princeton University Press, Princeton, N. J., no. 34, pp. 129–153; 1956.

[10] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell Sys. Tech. J.*, vol. 34, pp. 1045–1079; September, 1955.

[11] D. A. Huffman, "The synthesis of sequential switching circuits," *J. Franklin Inst.*, vol. 257, pp. 161–190, 275–303; March, April, 1954.

II. Input Events and Regular Expressions

Consider a set of n input terminals, numbered from 0 to $n-1$. With each terminal j, associate a variable x_j which may take on one of two possible values, called 0 and 1. These binary variables are called *input signals* and the ordered n-tuple of 0's and 1's, $(x_{n-1}, x_{n-2}, \cdots, x_0)$, is an *input configuration*. The set of all $k=2^n$ different input configurations is called the *input alphabet* A_k, and the individual configurations are *symbols* of the alphabet. With each symbol we can associate an integer corresponding to the binary number formed by the input configuration, as shown in Table I. The input alphabet thus consists of the symbols $0, 1, 2, \cdots, k-1$, where $k=2^n$. The symbols 0 and 1 are used to denote the binary values of the inputs and of the output, and also the input configurations $00\cdots0$ and $00\cdots01$. The meaning is clear from the context.

TABLE I
Input Alphabet

Symbols	Signals			
	x_{n-1}	x_{n-2} · · · · · · · · ·	x_1	x_0
0	0	0 · · · · · · · · ·	0	0
1	0	0	0	1
⋮	⋮	⋮	⋮	⋮
2^n-1	1	1 · · · · · · · · ·	1	1

It is assumed that the inputs are observed only at discrete moments of time and that, at these moments, any changes in the inputs have all been completed, *i.e.*, we consider the *synchronous* case only. Without loss of generality the time scale can be represented by integers, with intervals of unit length. Therefore at any integral time t every input signal has a definite value, 0 or 1. A sequence of input configurations (in time) is called an *input sequence*. Since the input sequences will be applied later to physical sequential circuits, it is assumed that all sequences have a beginning, *i.e.*, they do not extend indefinitely into the past. The time scale can be represented by the integers $0, 1, \cdots, p$, where 0 is a fixed origin and p denotes the present. Sometimes, it will be convenient to consider the last q moments of time and, in this case, the time scale will be represented by the integers $p-q+1, \cdots, p-1, p$, without showing explicitly the relation of the present to the origin. This is illustrated in Fig. 1. All input sequences will begin at $t=1$.

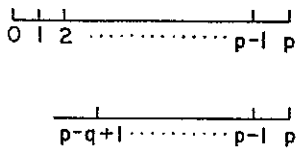

Fig. 1—Time scales.

An *event* is a subset of the set of all input sequences. Thus a partition is made of the set of all possible input sequences into two disjoint subsets: the subset of sequences for which the event occurs and the subset of sequences for which it does not occur. An event is said to *consist* of the sequences for which it occurs, and it *occurs* at the time of the application of the last symbol of any sequence which belongs to that event.

It will be shown later that the sets of input sequences which are of interest are precisely those which can be described by *regular expressions* which are defined[4] recursively as follows:

Given the *alphabet* $A_k=\{0, 1, 2, \cdots, k-1\}$, the *symbols* ϕ and λ, the *regular operators* "+," "·" and "*" and parentheses,

1) A string consisting of a single symbol of the input alphabet, a single ϕ, or a single λ, is a regular expression.
2) If P and Q are regular expressions, then so are $(P+Q)$, $(P\cdot Q)$ and P^*.
3) No other string of symbols is a regular expression unless its being so follows from (1) and (2) in a finite number of steps.

(This definition admits only restricted regular expressions, and will be extended later to include other operators.)

Regular expressions are formulae describing sets of input sequences. The *sum* or *union* $(+)$ of two sets of sequences, P and Q, is the set $(P+Q)$ consisting of the sequences which are members of P or Q or both. The *product* or *concatenation* $(\cdot)$ of two sets, P and Q, is the the set $(P\cdot Q)$, consisting of sequences formed by taking any sequence of P and following it by any sequence of Q. Note that this operation is not commutative, *i.e.*, $(P\cdot Q)\neq(Q\cdot P)$, in general. For convenience the dot will be omitted from now on and the product will be written (PQ). The *star* operation or the *iterate* of a set P is defined as the infinite sum,

$$P^* = \lambda + P + PP + \cdots .$$

(The "+" and "·" are associative; hence parentheses can be omitted.) For convenience PP is abbreviated to P^2, PPP to P^3, etc. Note that P^2 consists of those sequences formed by taking *any* two sequences of P and placing one after the other and is not restricted to sequences formed by repeating the same sequence twice. For example, $(21+3)(21+3)=2121+213+321+33$. For simplicity, no distinction in the notation will be made between a sequence and a set of sequences consisting of only that sequence. A regular expression R will denote a set of sequences and, also, R will denote any one sequence in that set.

The symbol λ is defined as the sequence of *zero length*. At this point, the introduction of this symbol may appear unnatural, but it will be shown later that its use has certain advantages. In connection with input events, we may interpret λ as a special starting signal

occurring only at $t=0$. Since by convention all input sequences begin at $t=1$, the occurrence of λ means that we are now ready to receive and recognize input signals. This will become clearer when automata are discussed.

Sometimes it may be convenient to consider sets of sequences which do not contain λ. We define the *definite iterate* of a set P as an abbreviation,

$$P^+ = P + P^2 + P^3 + \cdots = \sum_{n=1}^{\infty} P^n.$$

It can be seen that the iterate can be expressed in terms of the definite iterate as follows:

$$P^* = \lambda + P^+ \quad \text{and} \quad PP^* = P^*P = P^+.$$

Finally, the symbol ϕ is used to denote the *null* or *empty* set of sequences.

For simplicity of notation the symbol i is defined: $i=[0+1+2+\cdots+(k-1)]$, and is called the "don't care" event, *i.e.*, i denotes any symbol of the input alphabet.

Since events are defined in terms of input sequences, they can be combined by means of the regular operators. Thus we will speak of the sums, concatenations and iterates of events, the null event, and the "*don't care*" event i. By definition, an *event* is *regular*, if and only if, the set of sequences which constitutes the event can be described by a regular expression.

Certain basic properties of the regular operators are shown below and others will be derived when necessary. These relations follow directly from the definitions. If P, Q, and R are regular expressions, then:

$$\begin{aligned}
P + Q &= Q + P, && (+ \text{ commutative})\\
(P + Q) + R &= P + (Q + R), && (+ \text{ associative})\\
(PQ)R &= P(QR), && (\cdot \text{ associative})\\
PQ + PR &= P(Q + R),\\
PQ + RQ &= (P + R)Q, && (\text{distributive laws})\\
R + \phi &= \phi + R = R, && (\phi \text{ is an additive zero})\\
R\phi &= \phi R = \phi, && (\phi \text{ is a multiplicative zero})\\
R\lambda &= \lambda R = R, && (\lambda \text{ is a multiplicative unity})\\
R + R &= R,\\
\lambda^* &= \lambda,\\
\phi^* &= \lambda + \phi + \phi\phi + \cdots = \lambda.
\end{aligned}$$

At this point, it may be questionable whether the relation $R\phi=\phi R=\phi$ is simply a consequence of the definition of the empty set ϕ, *i.e.*, whether a set of sequences followed by the empty set of sequences constitutes the empty set of sequences. It will be seen later that this property of ϕ is a natural one, when state diagrams and automata are considered.

III. Classification of Regular Events

In order to see more clearly what types of events can be characterized by regular expressions we shall begin by considering certain simple events and showing how these are used in building the class of regular events. First of all, it is assumed throughout that all events start at $t=1$.

We begin by discussing *definite events* which are characterized by finite sets of sequences in the sense described below.

The class of definite events can be constructed by first considering initial definite events. An *elementary initial definite* event of length q is an event consisting of a single finite sequence of length q which occurs at times $1, 2, \cdots, p=q$. There are 2^{nq} elementary initial events of length q on n inputs. For example, 011101 is an elementary initial event of length 6. An *initial event* is an event consisting of a finite sum of elementary initial events. The length of an initial event is the length of the longest sequence in the event. For example $(01+01101+1110)$ is an initial event consisting of the elementary initial events 01, 01101 and 1110, and its length is 5.

We define an *elementary non-initial definite* event of length q as an event consisting of a sequence of don't care events i which can be of any length (including zero length), followed by a single finite sequence of length q. The sequence of don't care events begins at $t=1$, if its length is greater than zero, and the finite sequence occurs at times $p-q+1, \cdots, p-1, p$. This finite sequence determines the length of the event and will be called the *definite part* of the event. If P is an elementary initial event and R are corresponding elementary non-initial event, we can write,

$$R = i^*P.$$

A *non-initial definite* event is an event consisting of a finite sum of elementary non-initial events, *i.e.*,

$$\begin{aligned}
Q &= R_1 + R_2 + \cdots + R_j\\
&= i^*P_1 + i^*P_2 + \cdots + i^*P_j\\
&= i^*(P_1 + P_2 + \cdots + P_j).
\end{aligned}$$

The length of a non-initial event is the length of the longest sequence in the definite part of the event. Note that a non-initial event may occur many times at different moments of time, *i.e.*, the sequence of don't cares i^* may contain the desired definite part many times. At any given time, t, we are interested in knowing whether the event has occurred *at* time t and this can be checked by inspecting the last q symbols up to and including time t. From this inspection we cannot decide whether the event has occurred in the past. On the other hand, an elementary initial event can only occur once at time q. A non-elementary initial event may occur more than once, if and only if, at least one of its sequences is identical to the initial portion of another sequence. For example, the event $R=(01+0110+11)$ may occur twice, if the sequence is 01 followed by 10. Then R occurs at $t=2$ and at $t=4$. Note also that every non-initial event $R=i^*P$ contains the initial event, P, because i^* contains λ.

The most general type of definite event is the *composite definite* event which consists of the sum of an initial event and a non-initial event. For example, $M=01+i^*101$ is a composite event of length 3, containing the initial event (01) and the non-initial event i^*101. The class of definite events thus consists of initial, non-initial and composite events. It can be determined whether or not a non-initial event of length q has occurred, by inspection of the last q input symbols; *i.e.*, a non-initial event consists of all sequences ending in a specified set of sequences. However, in order to determine whether an initial or a composite event of length q has occurred at time t, we not only require the knowledge of the last q symbols but also the relation of t to the time origin. In general, the occurrence of a definite event of length q can be ascertained by knowing t and the last q symbols of the input sequence.

The remaining events which are regular but not definite are called indefinite. For example, the events $E=(01)^*$ and $F=i^*11i^*$ are not definite because the entire past must be known in order to determine whether they have occurred or not. We may now redefine the class of regular events to consist of only those events which are formed by applying the regular operations to members of the class of definite events. In the examples above, $E=P^*$, $P=(01)$ and $F=i^*Ri^*$, $i=(0+1)$, $R=(11)$. This interpretation is identical with the previous one in terms of individual symbols, because the class of definite events includes the events corresponding to the symbols of the input alphabet, and any event which can be constructed using sequences of length 1 (*i.e.*, the symbols) can also be constructed from definite events and vice versa.

Finally, events which are not regular are called *irregular*. An example of an irregular event[2] on the alphabet {0, 1} is the sequence of inputs which consists of zeros except when the total number of input symbols is a perfect square, in which case the sequence has a 1. In other words, the desired set of sequences is the infinite sum

$$1 + 0^81 + 0^81 + 0^{15}1 + \cdots 0^{n^2-1}1 + \cdots .$$

The event is not regular because it cannot be represented by a finite number of symbols and the regular operators; *i.e.*, the regular operations must be applied an infinite number of times. In contrast to this consider $01+(01)^2+(01)^3+\cdots$ which is also an infinite sum, but can be written as $(01)(01)^*$.

IV. Automata

We shall consider a restricted class of *automata*[5-11] satisfying the following conditions:

1) The automaton is of *fixed, finite* size.
2) It has a *finite number n* of *binary inputs* and *one binary output.*
3) The inputs are independent variables.
4) The automaton has a *finite number* of *internal states* which depend on the previous internal states and the inputs.
5) The output at any time is *determined* by the *internal state* of the automaton and the *inputs* just before that time. (The automaton is *deterministic.*)
6) The automaton in *synchronous*, *i.e.*, its input and output history can be described completely as occurring during certain discrete moments of time labelled by the integers 0, 1, 2, $\cdots$.

In other words, we shall consider only automata which are *fixed, finite, deterministic, discrete* and *synchronous.* Such automata will be referred to as *finite automata.*

The external behavior of a finite automaton may be completely described by a state diagram.[9,10] Given the state and the inputs at any time t, the next state and the output at time $t+1$ can be determined. A state diagram of an automaton consists of nodes (one for each internal state) and directed lines representing transitions from one state to another. There is exactly one directed line leaving each state for every input symbol. The output can be associated with transitions and each transition will be labeled x/z, where x is an input symbol and z the output (0 or 1). If it is desired to associate the output with an internal state rather than with a transition, that state will be labeled q/Z.

In general, automata are constructed from a finite number of copies of *devices*, by connecting the output terminals of some devices to the input terminals of others. The resulting arrangement of devices is a *net* or a *circuit.* The independent inputs are called *direct inputs.*

The above class of automata includes *clocked* sequential circuits, in which the circuit inputs do not change while the clock pulse is present, and the interval between clock pulses is usually constant. The class also includes sequential circuits operating in *pulse mode* without a clock pulse. In this case the duration of the input pulses is controlled, and a minimum spacing between pulses is preserved. Furthermore, since flow tables and state diagrams can be used to characterize iterative combinational circuits,[12] the regular expression methods are also applicable to this class of circuits.

V. Realization of Definite Events

Consider an automaton constructed from a given set of devices.[6] We can represent the automaton by showing the devices and the interconnections between them. If, in such a diagram, every device is replaced by a node, the connections between devices by a set of directed lines, and the input and output terminals by nodes, we obtain a *directed graph.* If it is not possible to start at any node and return to that node by following the directed lines, the automaton is *circle free.*

[12] E. J. McCluskey, Jr., "A comparison of sequential and iterative circuits," *Trans. AIEE* Commun. and Electronics, no. 46, vol. 78, pp. 1039–1044; January, 1960.

If the output of an automaton at any time t is uniquely determined by the direct inputs at times $t-q+1, \cdots, t-1, t$, the automaton is called *definite*.[5] (A device can be considered as a simple automaton and we may speak also of definite devices.) In other words the action of a definite automaton can be described by the last q symbols of the input alphabet, *i.e.*, by a definite event. An automaton *realizes* an event E, if and only if, an output of 1 is produced by the automaton at the time of the application of the last symbol of any input sequence contained in E. Therefore, an automaton is definite, if and only if, the set of input sequences resulting in an output constitutes a definite event. If the output at time t is independent of the inputs at times $t-d+1, \cdots, t-1, t$, but is uniquely determined by the inputs at times $t-q+1, \cdots, t-d$, it is convenient to say that the automaton realizes the definite event occurring at times $t-q+1, \cdots, t-d$ with a *delay* d.

Figs. 2 and 3 show several devices which are defined in terms of input-output relationships, but have physical counterparts used in construction of sequential circuits. All of these devices are definite devices realizing non-initial definite events. The input configurations are coded into decimal integers. For example, in case of the neuron, the input combinations which produce an output are $(x_1x_2w_1w_2) = (0100)$, (1000) and (1100), corresponding to the integers 4, 8 and 12. The complement of x is denoted by x', the logical "and" or conjunction by $x \,\&\, y$, and the "inclusive or" by $x+y$. There is no need to differentiate between the "inclusive or" as applied to Boolean variables and the regular operation "sum," because both have identical properties.

To illustrate how these devices can be described by regular expressions, consider the "AND" gate of Fig. 2(a). The output is 1 at time t, if and only if, the input configuration at time t is (11), which corresponds to the symbol 3. Since any sequence can precede this sequence of length 1, the regular expression is i^*3, and denotes all sequences ending with the symbol 3. The other devices are similarly described in Figs. 2 and 3.

A set of definite devices is defined to be *complete*, if it is possible to realize every definite event, with delay allowed, using a finite number of copies of these devices. Arden[6] presents the following theorem: "A set of definite devices, such that the output of each device, D_j, depends only on the inputs at a single earlier time $t-d_j$, is complete, if and only if, (a) Any Boolean function of the inputs at time t can be realized with some delay, and (b) it is possible to synthesize either a unit delay or two relatively prime delays."

Examples:

1) The Sheffer stroke with unit delay is not complete[6] because condition (b) is not satisfied; *e.g.*, the function $C(t+d) = A(t) \,\&\, B'(t) + A'(t) \,\&\, B(t)$ cannot be synthesized.
2) The Sheffer stroke with unit delay and an inverter form a complete set, because a unit delay can be synthesized as shown in Fig. 4.
3) The devices shown in Fig. 5(a) also form a complete set, in which two relatively prime delays of 2 and 3 units can be constructed. Fig. 5(c) shows a realization of the event $i^*(0+01)$ with a delay of 5 units, using these devices.
4) Neurons form a complete set.

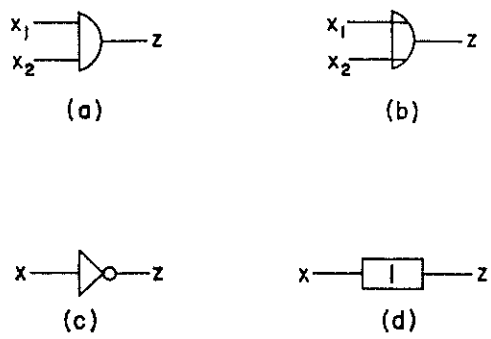

Fig. 2—Examples of devices. (a) AND gate; $z(t) = x_1(t) \,\&\, x_2(t)$, $R = i^*3$. (b) OR gate; $z(t) = x_1(t) + x_2(t)$, $R = i^*(1+2+3)$. (c) Inverter; $z(t) = x'(t)$, $R = i^*0$. (d) Unit delay; $z(t) = x(t-1)$, $R = i^*1i$ with zero delay or $R = i^*1$ with unit delay.

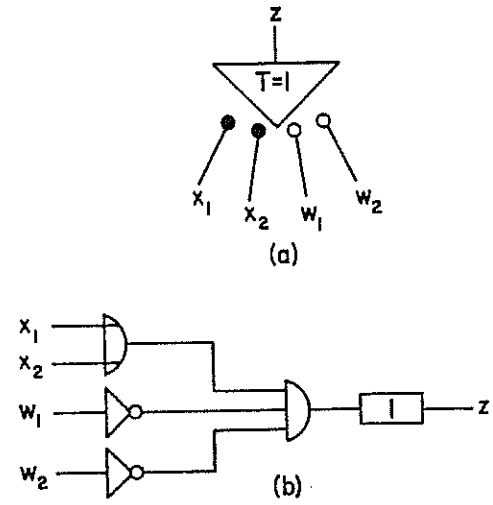

Fig. 3—Example of a neuron. (a) Neuron symbol; x_1, x_2—excitory inputs, w_1, w_2—inhibitory inputs. T—threshold of the neuron; $z(t) = [x_1(t-1)+x_2(t-1)] \,\&\, [w_1'(t-1)] \,\&\, w_2'(t-1)]$, $R = i^*(4+8+12)i$ with zero delay, $R = i^*(4+8+12)$ with unit delay. (b) Representation of the neuron in terms of AND gates, OR gates, inverters and delays.

Arden also presents the following theorem which is a generalization of a theorem of Kleene:[2] "Given a complete class of definite devices, every circle-free automaton represents a definite event and every definite event is realizable by a circle-free automaton."

The theorem states that every definite event can be realized by a circle-free automaton but it may be possible to synthesize it by an automaton with circles. For example, the automaton shown in Fig. 6 is not circle-free and produces an output for all sequences ending with a 1, with unit delay.

The discussion above was restricted to devices which realize non-initial definite events. In order to construct

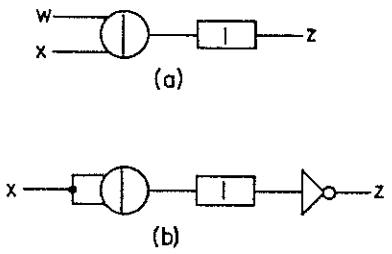

Fig. 4—Sheffer stroke circuits. (a) Sheffer stroke with unit delay; $z(t)=w(t-1)\,|\,x(t-1)=w'(t-1)+x'(t-1)$. (b) Construction of unit delay.

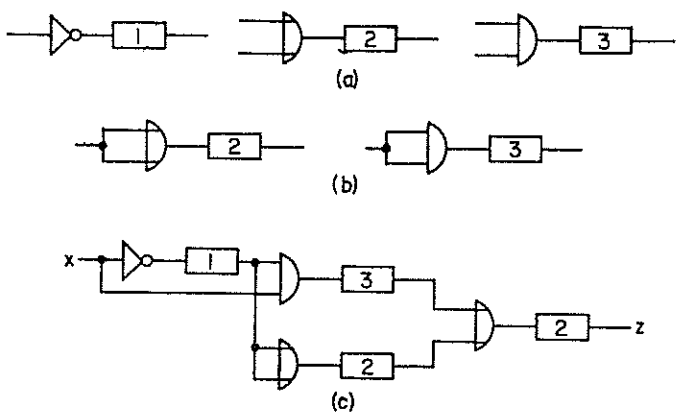

Fig. 5—Illustrating delayed logic. (a) A complete set of devices: Inverter, delay 1; OR, delay 2; AND, delay 3. (b) Realization of delays 2 and 3. (c) Example of synthesis, $R=i^*(0+01)$, with delay 5.

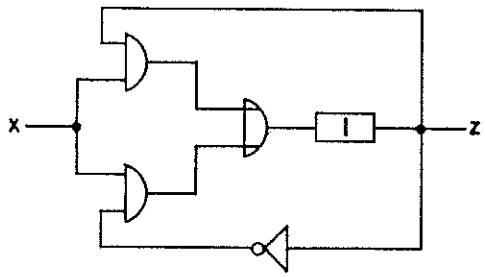

Fig. 6—An automaton with circles; $z(t)=x(t-1)$.

automata which realize initial events, a special starting pulse, λ, is introduced. This pulse occurs only at $t=0$ and corresponds to starting the sequential circuit. It is given the same symbol as the sequence of zero length because, mathematically, both perform the same functions. Fig. 7(a) shows the automaton which realizes the event λ by producing an output pulse at $t=0$. Since all input sequences are assumed to start at $t=1$, the application of the λ pulse to an automaton enables it to begin responding to the inputs. In practice the starting pulse may be replaced by the action of turning the power on in order to start a sequential circuit operating. However, for consistency, we shall assume that all the sequential circuits have a starting pulse. To formalize this, let us define a *machine* to mean an automaton with the starting pulse and the devices required for its proper use. These concepts will be illustrated by an example. It has been shown that an AND gate realizes the non-initial event, $R=i^*3$. If it is desired to realize

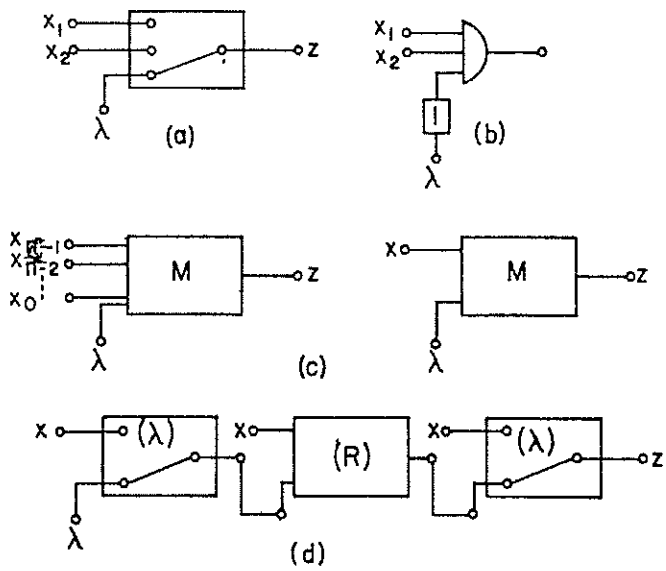

Fig. 7—Illustrating the use of λ. (a) Automaton realizing the event $R=\lambda$. (b) Automaton realizing $R=3$. (c) Models for machines. (d) Illustrating a property of λ.

the initial event $P=3$, this can be done with the aid of the starting pulse as shown in Fig. 7(b).

Fig. 7(c) shows the model of a machine. There are n input terminals, 1, 2, · · · , n and one special λ terminal on which the starting pulse appears. Sometimes, for brevity, only one input terminal will be shown, labelled x and representing an n-tuple of inputs. The variable x may take on any value in the input alphabet, whereas the x_i are binary variables.

The sequence of zero length was given the property of a multiplicative identity, where by multiplication is meant concatenation. This same property is given to the starting pulse as is illustrated in Fig. 7(d). If a machine M_1 realizes an event R_1 and another machine M_2 realizes an event R_2, the event R_1R_2 is realized by connecting the output of M_1 to the λ terminal of M_2. This will be further discussed in Section VI, but this remark and Fig. 7(d) show that $\lambda R=R\lambda=\lambda R\lambda=R$.

At this point it is perhaps in order to remark about the notation. If it were to be rigorous, we would be forced to introduce a host of new symbols and a great deal of confusion. Thus an input symbol is one thing, a sequence consisting of that input symbol is another, and the set of sequences consisting only of this one sequence of unit length is still another. We would also need a different symbol for the event consisting of that set of sequences and another for the regular expression and again another for the machine realizing the event. Similar considerations hold for λ, which may be interpreted as a sequence of zero length, a set of sequences consisting of only one member λ, a starting pulse, the event λ, the machine realizing that event, etc. This paper is of an explanatory nature and it is felt that such rigor is unnecessary and very confusing. Therefore, if no ambiguity can possibly arise, in the interest of simplicity, notational subtleties will be disregarded.

In concluding the discussion of definite events, it should be pointed out that every definite event and the corresponding definite machine can also be uniquely characterized by Boolean functions of the inputs at times $p-q+1, \cdots, p-1, p$. For example, the elementary initial event $R=10$, realized by a one-input machine is described by the Boolean equations $z(2) = x(1) \,\&\, x'(2)$, $z(t) = 0, t \neq 2$; the output z is 1 at $t=2$, if and only if, the input x was 1 at $t=1$ and 0 at $t=2$. Similarly, the non-initial event $P=i^*(00+11)$ is given by $z(t) = x'(t) \,\&\, x'(t-1) + x(t) \,\&\, x(t-1)$.

VI. Realization of Regular Events—Instantaneous Logic

A set of devices is capable of performing instantaneous logic, if any Boolean function can be realized with zero delay. Any set of devices, which is complete for Boolean algebra and produces outputs without delays, will suffice. For convenience it is assumed that AND gates, OR gates and inverters are available; if other devices are used, these gates can always be constructed. In order to obtain a set of devices complete for definite events, the existence of a unit delay is also assumed.

We now proceed to outline the proof of the following: *Theorem.*[2,6] Given a complete set of devices, capable of instantaneous logic, every regular event can be realized with zero delay by a machine with n direct inputs and a starting input λ.

The theorem will be proved by construction, and we begin by considering the realization of an elementary initial event. Let the sequence which is to produce an output be $a_1\, a_2 \cdots a_q$. Each symbol a_j corresponds to an n-tuple of input signals and the block labelled f_j in Fig. 8(a) consists of AND gates, OR gates and inverters and produces an output of 1 at time t, if and only if, the inputs correspond to the given n-tuple, a_j. For example, if $a_j=(01101)$, then $f_j = x_4' \,\&\, x_3 \,\&\, x_2 \,\&\, x_1' \,\&\, x_0$. Each function f_j is delayed by $q-j$ units of time, thus giving the correct time relationship. The starting pulse, λ, delayed by q units of time, ensures that no output can be produced unless λ has been received at $t=0$ and was followed by the correct sequence $a_1 a_2 \cdots a_q$ at times $1, 2, \cdots, q$.

Next, an initial event which is the sum of r elementary initial events is realized by simply combining the outputs of machines realizing the elementary events P_j by means of an OR gate. This is shown in Fig. 8(b).

An elementary non-initial event $R=i^*P$ may be realized by constructing a machine to realize the initial event P and ensuring that the λ terminal has a pulse applied to it at all times, because the sequences in i^* can be of any length. Fig. 9(a) illustrates this construction. Similarly, a non-initial event consisting of r elementary non-initial events, P_j, is realized with the aid of an OR gate, as shown in Fig. 9(b).

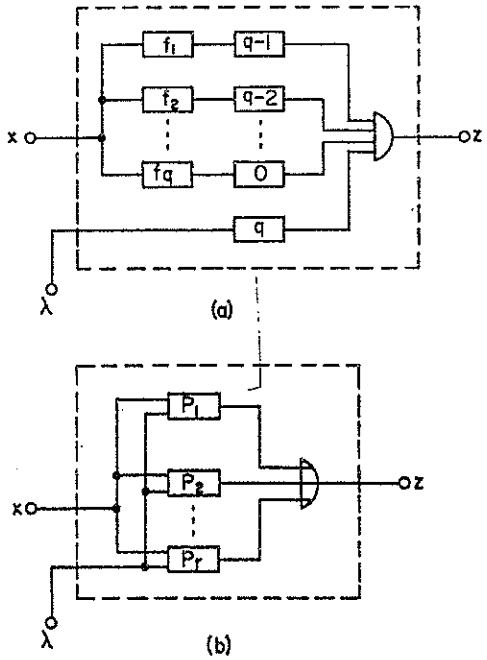

Fig. 8—Realization of initial events. (a) Elementary initial event. (b) Initial event.

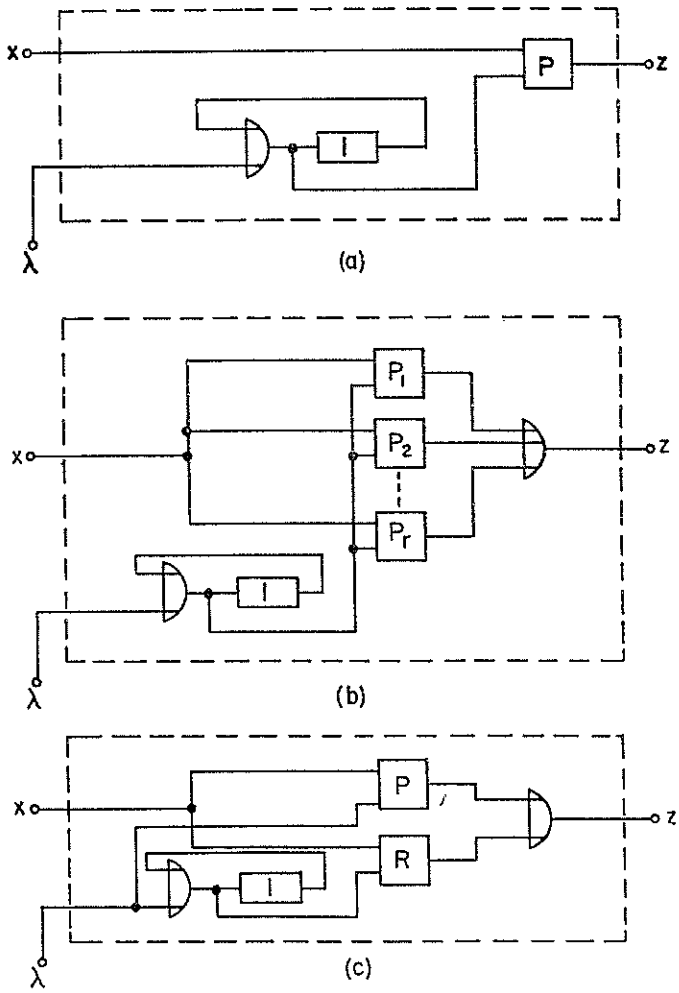

Fig. 9—Realization of non-initial and composite events. (a) Elementary non-initial event. (b) Non-initial event. (c) Composite event.

The construction of a machine to realize a composite event $M=P+Q=P+i^*R$, is given in Fig. 9(c), and the desired output is achieved by providing correct signals to the λ terminals.

It has been shown that every definite event can be realized without delay with instantaneous logic. As was remarked above, regular events can be constructed by applying the regular operations to definite events. Suppose machines have been constructed to realize the regular events E and F. The realization of the events $E+F$, EF, and E^* is shown in Fig. 10. This completes the construction.[13]

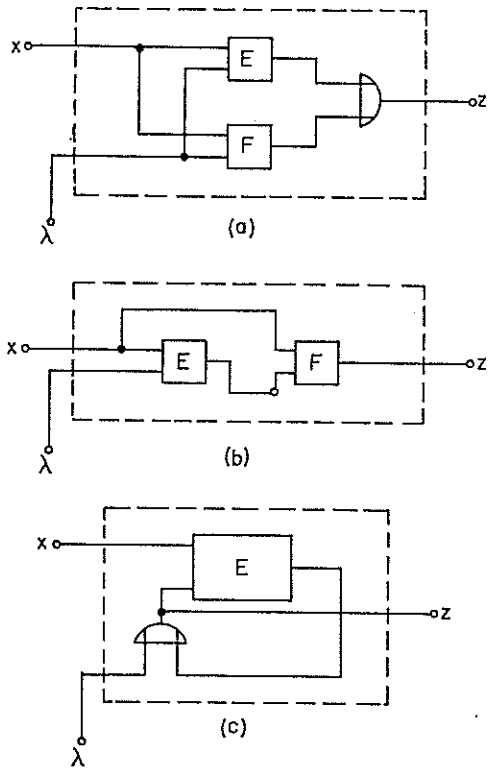

Fig. 10—Realization of regular events.
(a) $R=E+F$. (b) $R=EF$. (c) $R=E^*$.

VII. Realization of Regular Events—Delayed Logic

If devices capable of instantaneous logic are not available, regular events cannot be realized with zero delay. However, Kleene[2] has shown that, if every definite event can be realized with delay d using some complete set of devices, then every regular event can also be realized with delay d. The importance of this theorem in practical applications is questionable, since it is not known which sets of devices are sufficient to realize all definite events with delay d. The theorem and its proof is also presented by Arden[6] and will not be pursued here.

[13] For a proof of the validity of the construction see: J. A. Brzozowski, "Regular Expression Techniques for Sequential Circuits," Ph.D. Dissertation, Dept. of Elec. Engrg., Princeton University, Princeton, N. J.; 1962.

VIII. State Diagrams and Regular Expressions

The action of every finite automaton can be described by means of a state diagram. McNaughton and Yamada[4] have shown that it is possible to obtain the regular expression describing the automaton, directly from the state diagram by means of the algorithm described below. A very similar method has also been presented by Lee;[5] our discussion follows more closely the work of McNaughton and Yamada.[4]

Consider a state diagram with m internal states $q_1, q_2, \cdots, q_m$. It is assumed that one of these states q_s is initial, *i.e.*, the machine is in state q_s at $t=0$. The outputs are associated with internal states. Let the output states be $q_{z_1}, q_{z_2}, \cdots, q_{z_l}$. The algorithm is effected by constructing auxiliary regular expressions $a^k{}_{ij}$ defined recursively as follows:

1) $a_{ij}{}^0$ is a symbol (or a sum of several symbols) a_r of of the input alphabet, if there is a transition from state i to state j labelled a_r and it is the symbol ϕ, denoting the empty set of sequences, if no direct transition exists from state i to state j. Thus $a_{ij}{}^0$ is the set of input sequence which takes the state diagram from state i to state j without going *through* any of the states.
2) $a^k{}_{ij}=a^{k-1}{}_{ij}+a^{k-1}{}_{ik}(a^{k-1}{}_{kk})^*a^{k-1}{}_{kj}$, for $k>0$; is the set of input sequences[4] which take the state diagram from state i to state j without going through any state designated by a number higher than k.

The set of all input sequences which produce an output is given by

$$R = a^m{}_{sz_1} + a^m{}_{sz_2} + \cdots a^m{}_{sz_l},$$

where s is the number of the initial state, the z_r are the numbers of the output states and m is the total number of states.

The algorithm will be illustrated by a simple example shown in Fig. 11. The initial state 1 is indicated by an arrow, and the output state 2 is labelled 2/1. The required regular expression is therefore $R=a_{12}{}^2$. We have

$$a^2{}_{12} = a^1{}_{12} + a^1{}_{12}(a^1{}_{22})^*a^1{}_{22},$$
$$a^1{}_{12} = a^0{}_{12} + a^0{}_{11}(a^0{}_{11})^*a^0{}_{12},$$
$$a^1{}_{22} = a^0{}_{22} + a^0{}_{21}(a^0{}_{11})^*a^0{}_{12},$$
$$a^0{}_{11} = 0, \quad a^0{}_{12} = 1, \quad a^0{}_{21} = 1, \quad a^0{}_{22} = 0.$$

Hence,

$$\begin{aligned} R &= (1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) \\ &= [(\lambda + 00^*)1][\lambda + (0 + 10^*1)^*(0 + 10^*1)] \\ &= 0^*1(0 + 10^*1)^*. \end{aligned}$$

The final form of the regular expression depends on the manner of numbering the states. If the numbers of the states in Fig. 11 are interchanged, we get $R=(0+10^*1)^*10^*$. The two expressions are equivalent but, in general, no easy methods for determining whether two regular expressions represent the same sequences are available.

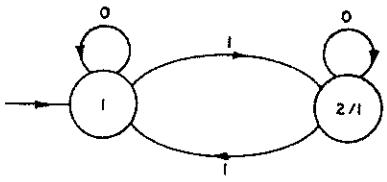

Fig. 11—State diagram of an automaton.

A different method for obtaining a regular expression from the state diagram was presented by Arden.[6] Let the set of all sequences taking the diagram from the starting state q_s and leaving it in state q_k be denoted by x_{sk}. Then the following equations hold:

$$\begin{aligned}
x_{s1} &= x_{s1}a^0_{11} + x_{s2}a^0_{21} + \cdots + x_{sm}a^0_{m1},\\
x_{s2} &= x_{s1}a^0_{12} + x_{s2}a^0_{22} + \cdots + x_{sm}a^0_{m2},\\
&\cdots\cdots\cdots\cdots\cdots\\
x_{ss} &= x_{s1}a^0_{1s} + x_{s2}a^0_{2s} + \cdots + x_{sm}a^0_{ms} + \lambda,\\
&\cdots\cdots\cdots\cdots\cdots\\
x_{sm} &= x_{s1}a^0_{1m} + x_{s2}a^0_{2m} + \cdots + x_{sm}a^0_{mm}, \qquad (1)
\end{aligned}$$

where a_{ij}^0 has been defined above. Note that in this interpretation the starting pulse λ can be used to *put* the machine in the starting state at $t=1$. For this reason the equation for the starting state includes the sequence λ. In another interpretation, if the machine *is* in the starting state at $t=0$, it will remain in that state, if we apply the sequence of zero length (*i.e.*, do exactly nothing). Arden[6] has also shown that equations of this type can be solved, if it is possible to solve the equation

$$x = xa + b, \qquad (2)$$

and proved that, if a does not contain λ the solution is $x=ba^*$. It is seen that in solving for x_{sk}, the coefficient multiplying x_{sk} on the right side of any equation of (1) will never contain λ, because it is a^0_{kk}. Therefore, the equations can always be solved and the method will be illustrated using the example in Fig. 11. The equations are

$$x_{11} = x_{11}0 + x_{12}1 + \lambda, \qquad (3)$$

$$x_{12} = x_{11}1 + x_{12}0. \qquad (4)$$

Solving (4) for x_{12} with $a=0$ and $b=x_{11}1$, gives

$$x_{12} = (x_{11}1)0^*. \qquad (5)$$

Substituting for x_{12} from (5) into (3) yields

$$x_{11} = x_{11}0 + x_{11}10^*1 + \lambda = x_{11}(0 + 10^*1) + \lambda. \qquad (6)$$

Solving (6) with $a=0+10^*1$, $b=\lambda$, gives

$$x_{11} = \lambda(0 + 10^*1)^* = (0 + 10^*1)^*. \qquad (7)$$

Finally, x_{12} is obtained by substituting for x_{11} from equation (7) into (5):

$$x_{12} = x_{11}10^* = (0 + 10^*1)^*10^*. \qquad (8)$$

As was the case in the previous method, different forms of regular expressions can be obtained by using a different order of elimination in the solution of the equations. In this case the output is associated with state 2 and, for the above example, $R=x_{12}=(0+10^*1)^*10^*$.

The related problem of finding a state diagram corresponding to a given regular expression has also been discussed by McNaughton and Yamada,[4] and the following algorithm illustrates their method. With each occurrence of a symbol of the input alphabet in a regular expression is associated a *position* directly to the right of the symbol. Thus, the positions of the symbol 0 are numbered 1, 2, $\cdots$ from left to right, and similarly the positions of the remaining symbols are also numbered. A position j is *initial*, if there exists a sequence contained in the regular expression which begins with the symbol associated with the jth position. Similarly a position is *terminal*, if there is a sequence ending with the symbol associated with that position. A *transition* is an ordered pair of positions i and j, such that if an input sequence reaches the symbol in the ith position, it can be followed by the symbol in the jth position and the new sequence is still an initial portion of a sequence in the regular expression. An *allowable sequence* of positions is any sequence beginning with an initial position, ending with a terminal position and such that there is a transition between any position and the next one. For example, consider the expression $R=(0+1)(00+01)^*(0+1)$. The positions are indicated as follows: $(0_1+1_1)(0_20_3+0_41_2)^*(0_5+1_3)$. Positions 0_1 and 1_1 are initial, 0_5 and 1_3 are terminal and the sequences $0_10_20_30_20_30_5$, $0_10_41_20_20_31_3$ are examples of allowable sequences of positions. The validity of the algorithm is shown by assuming an initial state and one state for every possible set of positions giving 2^p+1 states for an expression with p positions. To illustrate this, consider $R=(0_1+1_1)^*1_2$. The $2^3=8$ sets of positions are (ϕ), (0_1), (1_1), (1_2), (0_11_1), (0_11_2), (1_11_2), $(0_1, 1_1, 1_2)$. The allowed transitions are $[0_10_1]$, $[0_11_1]$, $[0_11_2]$, $[1_10_1]$, $[1_11_1]$, $[1_11_2]$. Assume the present state is q_i corresponding to a set of positions $(Pi_1, Pi_2, \cdots, Pi_x)$. If the input is 0, the next state q_j is that corresponding to the largest set of positions $(0_{j1}, 0_{j2}, \cdots, 0_{jr})$, provided there is a transition from at least one P_{i_n} for every 0_{j_m}. If there is no such set of positions $(0j_1 \cdots 0j_r)$ the diagram goes to state ϕ, corresponding to the empty set of positions. All transitions leaving state ϕ return to state ϕ. Analogous statements apply if an input of 1 is received. To carry out the construction of the state diagram, assume the diagram is in the initial state q_s. If a 0 is received, the next state is that corresponding to the set of positions (0_1), since (0_1) is an initial position. On the other hand, if a 1 is received, the diagram goes to the state (1_11_2), because the positions 1_1 and 1_2 are both initial (see Fig. 12). Note that neither the set (1_1) nor (1_2) is chosen because (1_11_2) is a larger set. If a 0 is received in state (0_1), the next state is also (0_1), because (0_1) is the only set of states with 0's

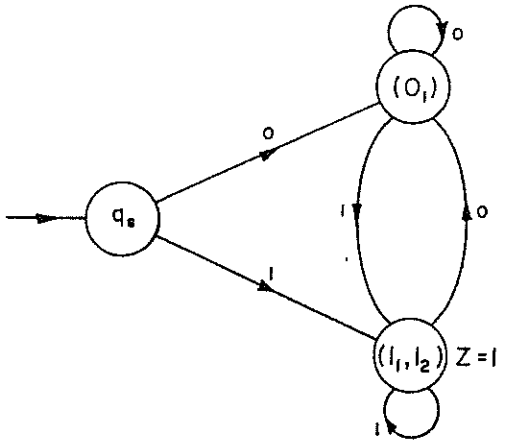

Fig. 12—State diagram for $R=(0+1)^*1$.

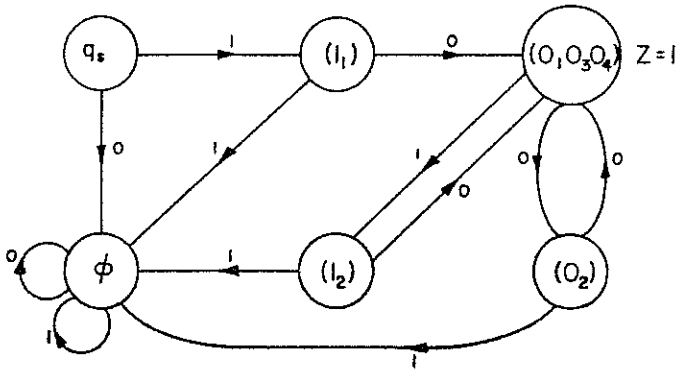

Fig. 13—State diagram for $R=1(00+01)^*0$.

in all of its positions and there is a transition $[0_1 0_1]$. If 1 is received in state (0_1), the next state is $(1_1 1_2)$, since we have the transitions $[0_1 1_1]$, $[0_1 1_2]$. Finally, a 1 applied to state $(1_1 1_2)$ returns the diagram to the state $(1_1 1_2)$ since transitions $[1_1 1_1]$ and $[1_1 1_2]$ are present. The remaining sets of positions correspond to inaccessible states. The state $(1_1 1_2)$ is an output state since the position 1_2 is terminal.

It should be pointed out that only sets of positions with all entries 0 or all entries 1, need be considered, and often many of the states are inaccessible. Thus the number of states is usually much smaller than 2^p+1.

As another example,[4] consider $R=1_1(0_1 0_2+0_3 1_2)^*0_4$. Here 1_1 is the only initial position and 0_4 the only terminal position. The transitions are

$$[1_1 0_1][1_1 0_2][1_1 0_4]$$
$$[0_1 0_2]$$
$$[0_3 1_2]$$
$$[0_2 0_1][0_2 0_3][0_2 0_4]$$
$$[1_2 0_1][1_2 0_3][1_2 0_4].$$

An initial state q_s is assumed. Then, if the input 1 is obtained the state diagram goes to state (1_1), and if 0 is obtained, to state ϕ, which is an "exit" or "dead" state, because no sequence in the regular expression can begin with a 0. Next, if in the state (1_1) an input of 0 is received, the state diagram goes to state $(0_1 0_3 0_4)$ and an output is produced because 0_4 is a terminal position. The process is carried out until all transitions have been shown and results in the diagram of Fig. 13.

A different approach to obtaining a state diagram from a regular expression has been presented by Lee.[5] He has shown that, if we have the flow tables[9,11] (or, equivalently, the state diagrams) for the automata realizing the regular expressions P and Q, then we can obtain the flow tables corresponding to the regular expressions $P+Q$, PQ and P^*. For details of the procedure the reader is referred to his paper.[5]

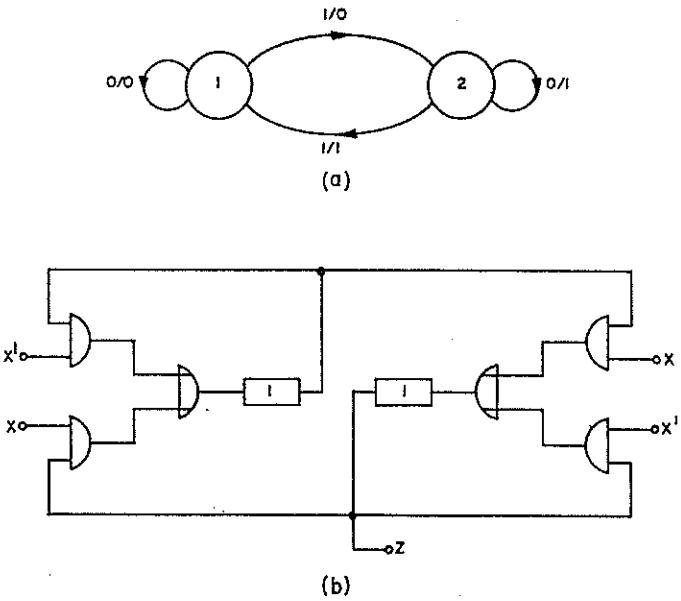

Fig. 14—Example of synthesis from state diagram. (a) State diagram. (b) Realization.

Arden[6] presents a direct synthesis procedure from an internal state diagram. Consider the example in Fig. 14. We can describe the action of the circuit as follows. The circuit will go to or remain in state 1, if it is in state 1 and the input is 0, or if it is in state 2 and the input is 1. State 2 can be similarly described. The realization follows immediately, if the present state is represented by the output of a delay element. This is a direct method but certainly not minimal and corresponds to the "one hot" assignment, *i.e.*, one memory element per state.

IX. Regular Events and Finite Automata

It was shown that every definite event is realizable by a circle-free machine and that every circle-free machine realizes a definite event. At this point, a corresponding statement can be made about regular events and finite automata. Since every finite automaton is representable by a state diagram and to every state diagram there is a corresponding regular expres-

sion, *every finite automaton realizes a regular event.* Conversely, a finite state diagram can be constructed corresponding to any regular expression. Since an event is regular, if and only if, it is described by a regular expression, *every regular event can be realized by a finite automaton.*

These results have been proved by Kleene[2] and point out the general applicability of the regular expression language. A proof is not required at this point because the correspondence between state diagrams and automata has been demonstrated constructively in Section VIII.

X. Extended Regular Expressions

It has been pointed out by McNaughton and Yamada[4] that it is very useful to enrich the language of regular expressions to include the operation of intersection (&), and negation or complementation (′).

The intersection of two sets of sequences P and Q is denoted by P & Q and is the set consisting of all sequences common to P and Q. The complement P' of a set of sequences P consists of all those sequences not in P. The added operators increase the ability of writing regular expressions from a word description. For example,[4] suppose it is desired to synthesize a sequential circuit which will produce an output either if there have never been three consecutive 0's in the input sequence or if there have been three consecutive 1's not followed by three consecutive zeros. The expression describing this behavior is easily seen to be $R=(i^*000i^*)'+i^*111(i^*000i^*)'$.

There is a difficulty introduced by including intersection and negation, because not much is known about methods of transforming regular expressions from one form to another and expressions including (&) and (′) are not easy to handle. Certain methods relating such expressions to state graphs are discussed by McNaughton and Yamada[4] but will not be treated here.

It is easy, however, to construct machines which realize events containing complementation and intersection. Thus, if machine M realizes an event described by the regular expression R, a machine to realize R' is obtained by inverting the output of M. Similarly, if M_1 and M_2 realize the events P and Q, respectively, the machine which realizes P & Q is constructed by combining the outputs of M_1 and M_2 by means of an AND gate. This is illustrated in Fig. 15.

XI. Improper State Diagrams

Ott and Feinstein[14] have taken a different approach for representing a regular expression by a state diagram. An unconventional or *improper* state diagram can be used to represent a regular expression and hence a finite automaton. The state diagram is improper because an automaton may assume more than one state simultaneously. The set of states assumed by the automaton at any given time is called its *configuration* at that time. The method will be briefly described by an example.[14]

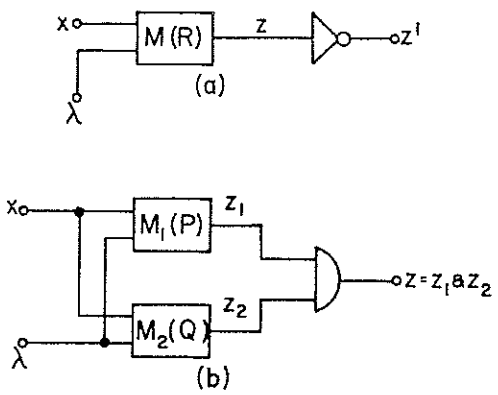

Fig. 15—Realization of R' and P & Q. (a) R'. (b) P & Q.

Consider the regular expression $R=(0+1(01)^*)^*$. It is assumed that the automaton which is to realize R is placed in the starting state 1 at $t=0$. Here R is of the form P^*, where $P=(0+1(01)^*)$; this fact will be represented as shown in Fig. 16. The interpretation given to the graph on Fig. 16 is as follows: If the automaton is in state 1, then it is also in states 2 and 3, and, if it is in state 2, it is also in state 3. The λ transitions merge the states in the forward direction and keep them separate in the reverse direction. Being in state 1, means that the automaton can now receive any sequence of P and that an output of 1 is produced because of the transition λ/1. Next we note that $P=Q+S$ where $Q=0$ and $S=1(01)^*$. The inclusion of this information into the improper state graph of Fig. 16 results in the graph shown in Fig. 17, which merely indicates that P consists of two parallel paths. The expression S is then decomposed into $S=UV$ where $U=1$ and $V=(01)^*$. This decomposition is shown in Fig. 18. Finally, U is similarly decomposed until the components are single input symbols; the resulting complete improper state diagram is shown in Fig. 19.

It can be seen that the improper state diagram is a graphical representation of the structure of a regular expression. The behavior of the corresponding automaton can be obtained, if the diagram is properly interpreted. For example, in the automaton of Fig. 19, if a 1 is received, the automaton will go to state 4. But being in state 4 implies also being in states 5, 2 and 3 since state 4 is connected to these by the sequence of zero length. Thus an output is produced upon reaching state 4, because the transition from state 2 to 3 is labelled with $z=1$.

A neural net realization can be obtained directly from an improper state diagram;[14] however this realization is far from minimal and resembles somewhat Arden's[6] construction.

[14] G. H. Ott and N. H. Feinstein, "Design of sequential machines from their regular expressions," *J. ACM*, vol. 8, p. 585–600; October, 1961.

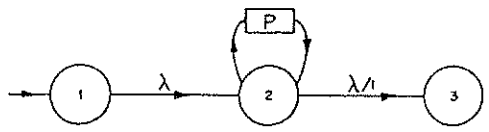

Fig. 16—$R=P^*$.

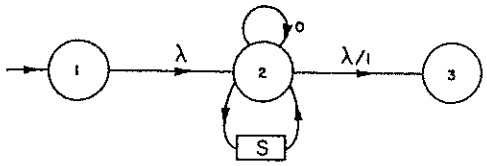

Fig. 17—$P=Q+S$.

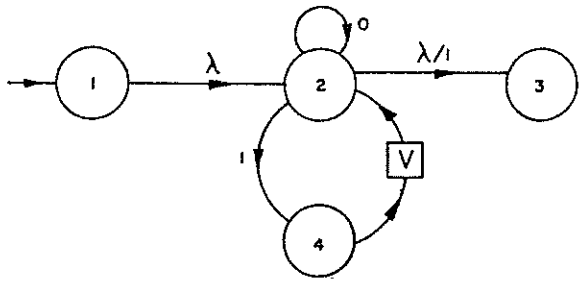

Fig. 18—$S=UV$.

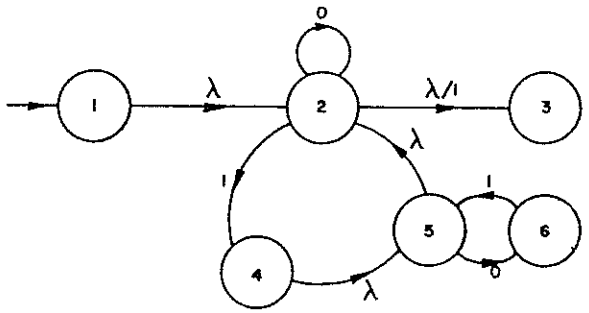

Fig. 19—Improper state diagram for $R=[0+1(01)^*]^*$.

There are certain difficulties with this type of diagram: First, the conversion to a proper state diagram is not very straightforward and, secondly, the minimization of an improper state diagram is difficult.

XII. Applications to Coding

Neumann[15] has used regular expressions in connection with variable-length codes. The regular expressions used correspond to a restricted class of finite automata with the following characteristics:

1) The automata are strongly connected, *i.e.*, every state can be reached from every other state.
2) The output $z=1$ occurs only during certain transitions leading to the initial state.

Given such a finite automaton, a code word is defined as any sequence $s=a_1a_2\cdots a_r$ which takes the automaton from its starting state back to the starting state and for which the output sequence is $00\cdots01$. For example, consider the state diagram of Fig. 20. State A is the initial state and an output $z=1$ occurs during the transition from state B to state A under the input 1. The code words generated by this automaton can be denoted by a regular expression. First we note that the sequences denoted by the expression $(1+00)^*$ are precisely all those sequences which can be applied to the automaton without ever producing an output, and returning the automaton to the starting state. In order to produce an output at the end of a sequence, the expression $(1+00)^*$ is followed by 01. Thus the set of code words is denoted by the regular expression $(1+00)^*01$. The first few code words are: 01, 101, 0001, 1101, 00101, 10001, etc.

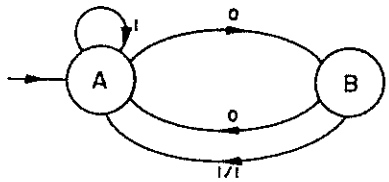

Fig. 20—State diagram for code example.

Properties of such codes are described by Neumann[15] but will not be discussed here. The purpose of mentioning this work was to point out the usefulness of the regular expression language in describing such codes.

Conclusion

An attempt has been made to present the theory and applications of regular expressions in a unified way, combining the results of several authors and showing similarities and differences in their treatments. The notation has been simplified in order to present to the reader a clearer picture of the ideas involved, and the terminology has been correlated, whenever possible, with that used in sequential circuit theory.

Acknowledgment

The author wishes to express his gratitude to Prof. E. J. McCluskey, Jr., of the Electrical Engineering Department, Princeton University, for his useful comments and suggestions.

[15] P. G. Neumann, "Efficient error-limiting variable-length codes," to be published in IRE Trans. on Circuit Theory.