Realization of Events by Logical Nets*

IRVING M. COPI, CALVIN C. ELGOT, AND JESSE B. WRIGHT†

University of Michigan, Ann Arbor, Mich.

1. *Introduction*

In *Representation of Events in Nerve Nets and Finite Automata* [3], S. C. Kleene obtains a number of interesting results. The most important of these, his analysis and synthesis theorems (theorems 5 and 3), are obscured both by the complexity of his basic concepts and by the nature of the elements used in his nets. In this paper we give a new formulation and new proofs of Kleene's analysis and synthesis theorems, in which we presuppose no acquaintance with Kleene's work. We use simpler basic concepts and construct our nets out of more familiar and convenient elements (see section 4). The simplified formulation and proofs should make these important results more widely accessible. Some detailed comments on Kleene's ideas are made in section 7.

The problems of analysis and synthesis concern the relationship between structure and behavior of nets. It is characteristic of nets that for any time t the state of any output of the net is completely determined by the states of the inputs of that net between time 1 and time t. Thus it is customary to regard the behavior of an output of a net as a function or "transformation" which assigns to every finite sequence of input states the state which it produces in that output. Any specification of that transformation expresses the behavior of the net output. In this paper we adopt Kleene's method of expressing behavior by means of events, an event being a set of finite sequences of input states. (The interrelations of several different methods of expressing behavior are discussed in the appendix.) To understand how an event can express the behavior of a net output, let the output state *on* be assigned to each sequence of input states in the event and let the output state *off* be assigned to each sequence of input states not in the event. This assignment defines the transformation corresponding to the given event. In this way an event specifies a transformation and thus expresses the behavior of a net output. A net output is said to realize an event if and only if its behavior is given by the corresponding transformation.

These notions can be made more precise in the following way. If we express an input state of a k-input net by means of a k-tuple of zeros and ones, then we can define a *k-event* to be a set of finite sequences of k-tuples of zeros and ones,

* Received September, 1957. This research was supported by funds provided by the Office of Naval Research, Contract Nonr-1224(21), Project NR049-114, and by Project MICHIGAN, a Tri-Service contract (DA-36-039sc-57654) administered by the U. S. Army Signal Corps. Funds for the writing of this report were provided by the Office of Naval Research under Contract Nonr-1224(21).

† The authors wish to thank Don W. Warren for his significant contributions in the early stages of the writing of this paper. They also wish to thank J. Richard Buchi, Arthur W. Burks, and John H. Holland for their helpful criticism and suggestions.

$k \geqq 0$. And we define an *event* to be a k-event for some $k \geqq 0$. A k-event E is *realized* by an output of a k-input net just in case for every time l the output is on at l if and only if a sequence in E of length l is applied to the inputs over the interval from time 1 to time l. Certain events (see sections 2 and 3) are defined to be "regular" and it is proved in section 5 that all regular and only regular events can be realized by the outputs of nets. This provides a characterization of exactly those kinds of behavior which can be realized by nets. Regular events are described by formulas called "regular expressions". The proof of our synthesis theorem (2) gives an effective procedure for finding a net and an output of that net which realizes a given regular event, and the proof of our analysis theorem (1) gives an effective procedure for finding the regular event which is realized by a given output of a given net.

In section 6 we apply these analysis and synthesis algorithms (theorems 1 and 2) to illustrative examples.

2. *Regular Sets and Regular Expressions*

The notion of "regularity" is central to our entire discussion. In the present section we define "regularity" for expressions and (derivatively) for sets, and prove our first lemma.

Given any non-empty finite ordered set A of n objects, we may wish to consider sets of finite sequences of those objects. To do so it is convenient to introduce a language L_A containing three operator symbols "$\vee$", "$\cdot$", and "$*$"; an alphabet of $n + 1$ letters $\Lambda, A_1, A_2, \cdots, A_n$; and *formulas* defined by the following inductive rule:

1. Each single letter of L_A is a formula.
2. If Ω_1 and Ω_2 are formulas of L_A, then so are $(\Omega_1 \vee \Omega_2)$, $(\Omega_1 \cdot \Omega_2)$, and Ω_1^*.
3. Nothing is a formula of L_A unless its being so follows from these rules.

The language L_A has the following interpretation in terms of sets of sequences of objects of A:

1. The letter Λ denotes the empty set, and each letter A_i denotes the set whose only member is the ith element of A, regarded as a sequence of length 1.
2. If Ω_1 and Ω_2 are formulas, then
   a) the formula $(\Omega_1 \vee \Omega_2)$ denotes the union of the sets denoted by Ω_1 and Ω_2;
   b) the formula $(\Omega_1 \cdot \Omega_2)$ denotes that set which contains every sequence of length $l_1 + l_2$ ($l_1 > 0$ and $l_2 > 0$) whose first section, consisting of its first l_1 terms, is a sequence in the set denoted by Ω_1, and whose last section, consisting of its last l_2 terms, is a sequence in the set denoted by Ω_2 (obviously $(\Omega \cdot \Lambda) = (\Lambda \cdot \Omega) = \Lambda$ for any Ω);
   c) the formula Ω_1^* denotes the union of the sets denoted by Ω_1, $(\Omega_1 \cdot \Omega_1)$, $((\Omega_1 \cdot \Omega_1) \cdot \Omega_1)$, $\cdots$.

The formulas of L_A are defined to be *regular expressions*, and a set of sequences of objects from A will be called a *regular set* if and only if it is denoted by a regular expression.

Where R is any binary relation defined on a given set S, a sequence of objects from S, $a_1a_2 \cdots a_l$, is an *R-sequence* if and only if for every i ($i = 1, 2, \cdots, l - 1$) a_iRa_{i+1}. (Note that any sequence of length 1 is an R-sequence.) Now we are able to state and prove

LEMMA 1. If R is a binary relation defined on a finite set S containing a and b, then the set of all R-sequences beginning with a and ending with b is regular.

PROOF: By induction on n, the number of objects in the set S.

α) $n = 1$. Here $a = b$.

Case 1. If $\overline{aRa}$, then the only R-sequence beginning with a and ending with b is a itself regarded as a sequence of length 1. Here the set of all R-sequences beginning with a and ending with b is the unit set $A = \{a\}$, which is regular.

Case 2. If aRa, then the set of all R-sequences beginning with a and ending with b is $\{a, aa, aaa, \cdots\}$, which is the regular set A^*.

β) $n > 1$. Here we assume the lemma true for any non-empty set containing fewer than n objects, and consider any set containing n objects with relation R defined over that set.

Case 1. $a = b$. Any R-sequence beginning with a and ending with a can be written $a\alpha_1a\alpha_2a\alpha_3 \cdots a\alpha_ra$. Here $a\alpha_ka$ can be simply aa, in case aRa, otherwise α_k is an R-sequence containing no occurrence of a. Let $e_1, \cdots, e_g$ ($g \geqq 0$) be the objects e other than a such that aRe, and let $f_1, \cdots, f_h$ ($h \geqq 0$) be the objects f other than a such that fRa. Now any R-sequence α_k begins with an e, ends with an f, and contains no occurrence of a. By the induction hypothesis, the set of all R-sequences beginning with a particular e, ending with a particular f, and containing no occurrence of a is regular. Let $B_1, B_2, \cdots, B_{gh}$ be these regular sets of R-sequences. Now if aRa, the set of all possible R-sequences $a\alpha_k$ is $A \vee A\cdot(B_1 \vee B_2 \vee \cdots \vee B_{gh})$, which reduces to A if there are no B's, that is, if $gh = 0$. But if $\overline{aRa}$, the set of all possible R-sequences $a\alpha_k$ is $A\cdot(B_1 \vee B_2 \vee \cdots \vee B_{gh})$, which reduces to the empty set if there are no B's, that is, if $gh = 0$. In either case the set is regular: call it C. From the construction of C it is clear that every sequence in C, in C^*, and in $C^*\cdot A$ is an R-sequence. Hence every sequence in $A \vee C^*\cdot A$ is an R-sequence beginning with a and ending with b, and every R-sequence beginning with a and ending with b is in $A \vee C^*\cdot A$. The set of all R-sequences beginning with a and ending with b is a regular set, since it is denoted by a regular expression.

Case 2. $a \neq b$. Any R-sequence beginning with a and ending with b can be written $a\alpha_1a\alpha_2a\alpha_3 \cdots a\alpha_ra\beta b$. Here each α_k is as in case 1, and $a\beta b$ can be simply ab in case aRb, otherwise β is an R-sequence containing no occurrence of a. By the argument of case 1, the set of all possible R-sequences $a\alpha_k$ is the regular set C. By the inductive assumption the set, B, of R-sequences not containing a from an e to b is regular. If $g = 0$, then B is empty. Let $D = A\cdot B$. Then every sequence in D is an R-sequence; every sequence in $D \vee C^*\cdot D$ is an R-sequence beginning with

a and ending with b, and every R-sequence beginning with a and ending with b is in $D \vee C^{*} \cdot D$. Hence the set of all such R-sequences is regular since it is denoted by a regular expression.

3. *Sequences of k-tuples*

In the present section we consider sets of sequences of objects of a particular kind, namely sets of sequences of k-tuples of ones and zeros. For a fixed $k \geq 0$, there are 2^k distinct k-tuples of ones and zeros. (By convention there is exactly one 0-tuple.) Where A^k is the set of all k-tuples of ones and zeros, the language L_{A^k} (also written L^k) will have its alphabet consist of the $1 + 2^k$ letters Λ, A_1, A_2, $\cdots$, A_{2^k}. Every regular expression or formula of L^k will denote a regular set of sequences of k-tuples of ones and zeros.

There are obvious interrelations between any two languages L^k and $L^{k'}$. Let us suppose for the sake of definiteness that $k < k'$. Where A^k is the set of all k-tuples and $A^{k'}$ is the set of all k'-tuples, since $k < k'$, each k-tuple of A^k will consist of the first k terms of a k'-tuple of $A^{k'}$. Given any k'-tuple, the *corresponding k-tuple* is that k-tuple which consists of the first k terms of the given k'-tuple. In general there will be $2^{k'-k}$ distinct k'-tuples that have the same corresponding k-tuple. The set $A^{k'}$ can be mapped onto the set A^k by mapping each k'-tuple of $A^{k'}$ onto the corresponding k-tuple of A^k.

This mapping can be extended to sequences of k'-tuples and sets of sequences of k'-tuples. Given any sequence of k'-tuples, the *corresponding sequence of k-tuples* is the sequence of corresponding k-tuples, that is, the sequence whose jth member is the corresponding k-tuple of the jth member of the original sequence. And given any set of sequences of k'-tuples, the *corresponding set of sequences of k-tuples* is the set of corresponding sequences of k-tuples.

We can now state and prove

LEMMA 2. If a given set of sequences of k'-tuples is regular then so is the corresponding set of sequences of k-tuples ($k < k'$).

PROOF: If the given set of sequences of k'-tuples is regular then it is denoted by a formula of $L^{k'}$. Now we map the formulas of $L^{k'}$ onto formulas of L^k according to the following rules:

1. Let the letter in $L^{k'}$ that denotes the empty set of k'-tuples be mapped onto the letter in L^k that denotes the empty set of k-tuples.
2. Let each letter in $L^{k'}$ that denotes a unit set containing a single k'-tuple regarded as a sequence of length 1 be mapped onto the letter in L^k that denotes the corresponding unit set containing the single corresponding k-tuple regarded as a sequence of length 1.
3. Where formulas Ω_1' and Ω_2' of $L^{k'}$ are mapped onto formulas Ω_1 and Ω_2 of L^k, then let the formulas $(\Omega_1' \vee \Omega_2')$, $(\Omega_1' \cdot \Omega_2')$, and $\Omega_1'^{*}$ of $L^{k'}$ be mapped onto formulas $(\Omega_1 \vee \Omega_2)$, $(\Omega_1 \cdot \Omega_2)$, and Ω_1^{*} of L^k.

Now if a given set of sequences of k'-tuples is denoted by a formula Ω' of $L^{k'}$, then by the mapping defined above the corresponding set of sequences of k-tuples is denoted by the formula Ω of L^k. Hence if the given set of

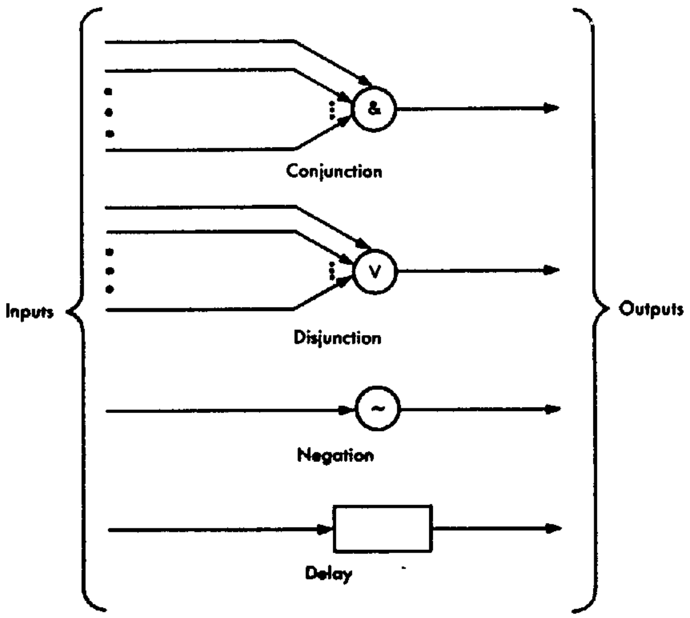

FIG. 4a

sequences of k'-tuples is regular so is the corresponding set of sequences of k-tuples.

4. *Nets*

We shall consider nets constructed out of four kinds of elements: conjunctions, disjunctions, negations, and delays. Each element has exactly one output, each negation and delay has exactly one input, and each conjunction and disjunction can have any number $n \geqq 1$ of inputs (see figure[1] 4a). The output of an element will sometimes be referred to as an "element output", and an input of an element will sometimes be referred to as an "element input."

At any discrete moment of time t ($t = 1, 2, 3, \cdots$) each input and each output is in one of two states, 1 or 0, *on* or *off*. Where i is an input or an output, that i is on (off) at time t is written $i(t) = 1$ (0). The output of a conjunction is on at time t if and only if all of its inputs are on at time t, the output of a disjunction is on at time t if and only if at least one of its inputs is on at time t. The output of a negation is on at time t if and only if its input is off at time t. These three kinds of elements, called switching elements, realize the indicated truth functions with no temporal delay. The output of a delay is on at time $t + 1$ if and only if its input was on at time t; at time 1 every delay output is off.

A combination of elements is a *net* if and only if it can be constructed from elements according to the following inductive rule:

1. Any element is a net.

[1] Figure numbers are related to section numbers.

2. If N_1 and N_2 are distinct nets, the first containing an output o and the second containing an input i which is not an output, then the result of identifying (connecting) o and i is a net.
3. If N is a net containing a delay output d and an input i which is not an output, then the result of identifying d and i is a net.
4. If N is a net containing inputs i_1 and i_2 which are not outputs, then the results of identifying i_1 and i_2 is a net.
5. If N_1 and N_2 are distinct nets then N_1 and N_2 taken together constitute a single net.

A *net output* of a net is any element output contained in that net. A *net delay output* of a net is an output of any delay element contained in that net. Each element output in a net is a distinct net output of that net, since an element output can be identified only with an input that is not an output. Since no two element outputs can ever be identified, a "net output" could equally well be defined as the result of identifying an element output with zero or more inputs.

A *net input* of a net is any input which is contained in that net which is not an output. Any net with one or more distinct net inputs can have its distinct net inputs ordered and labeled $i_1, i_2, i_3, \cdots$, and we shall assume that every such net has had its distinct net inputs so ordered and labeled. A *k input net* is a net with exactly $k \geqq 0$ distinct net inputs.

An *input state* of a k input net is an arbitrary assignment of ones and zeros to its inputs. Since the net inputs are ordered, each input state can be represented by a k-tuple of ones and zeros, and each of the 2^k distinct k-tuples represents a distinct input state of the net. We shall sometimes speak of the k-tuples as being the input states they represent.

If a k input net contains q delay outputs we order and label its net inputs and delay outputs $i_1, i_2, \cdots, i_k, d_1, d_2, \cdots, d_q$. A *complete state* of the net is an arbitrary assignment of ones and zeros to the net inputs and delay outputs. Since the net inputs and delay outputs are ordered, each complete state can be represented by a k'-tuple of ones and zeros ($k' = k + q$), and each of the $2^{k'}$ distinct k'-tuples represents a distinct complete state of the net. We shall sometimes speak of the k'-tuples as being the complete states they represent. Note that not all complete states are necessarily assumed by the net.

It should be remarked that the state of any output of a net at time t is (uniquely) determined by the complete state of the net at time t (cf. theorems VIII and IX of Burks and Wright [2]).

The interrelations between complete states and input states are fairly clear. Each input state of a net consists of the first k terms of a complete state of that net. Given the complete state of a net it will be convenient to refer to the input state contained in it as the input state *derived* from that complete state, or more briefly as its *derived input state*. The complete state of a k input net is a k'-tuple, and the derived input state of that net is the corresponding k-tuple—in the sense of "corresponding" explained in section 3.

We say that an input state or k-tuple is *applied* to a net at time t just in case the jth net input is on at time t if and only if the jth term of the k-tuple is a one.

A sequence of input states is called an *input sequence*. We say that an input sequence is *applied* to a net over the l' consecutive moments of time $l - l' + 1$, $l - l' + 2, \cdots, l$ if and only if the jth input state of the input sequence is applied to the net at time $l - l' + j$ ($j = 1, 2, \cdots, l'$).

An input sequence is said to be *derived* from a sequence of complete states if and only if both sequences contain the same number of terms (say l) and the jth term of the input sequence is derived from the jth term of the sequence of complete states ($j = 1, 2, \cdots, k$). And a set σ of input sequences is said to be derived from a set σ' if and only if every sequence in σ is derived from a sequence in σ' and every sequence in σ' has a derived sequence in σ.

Since a complete state usually includes states of delay outputs as well as states of net inputs, the number of complete states that a net can assume at any time $t > 1$ is in general greater than the number of input states it can have at that time. But that is not true for time $t = 1$. Let us use the term *initial complete state* for any complete state that a net can assume at time $t = 1$. Since every delay output is in state 0 at time $t = 1$, any initial complete state of a net is (uniquely) determined by its derived input state. And since the state of each delay output of a net at time $t + 1$ is determined by the complete state of the net at time t, the complete state of a net at time $t + 1$ is wholly determined by its complete state at time t and its input state at time $t + 1$. These remarks suffice to establish

LEMMA 3. Any given input sequence applied to a net over the first l moments of time determines a (unique) sequence of complete states which the net assumes during the first l moments of time.

Of the $2^{k'}$ distinct complete states of a net $a_1, a_2, \cdots, a_{2^{k'}}$, we say that a_i stands in the *direct-transition relation* to a_j if and only if the last q terms of a_j (the states of the q delay element outputs) are those that would be produced at time $t + 1$ if the net were in complete state a_i at time t (cf. Burks and Wang [1], p. 31). Next we define a sequence of complete states $a_1a_2 \cdots a_l$ to be a *transition sequence* of complete states if and only if for every i ($i = 1, 2, \cdots, l - 1$) a_i stands in the direct-transition relation to a_{i+1}. (Any complete state regarded as a sequence of length 1 is a transition sequence.) Since the direct-transition relation is a binary relation defined on the finite set of complete states of a given net, transition-sequences of complete states are R-sequences of the kind discussed in section 2. It is obvious that any sequence of complete states assumed by a net over any l' consecutive moments of time must be a transition sequence of complete states.

5. *Analysis and Synthesis Theorems*

THEOREM 1. Any set of input sequences realized by a net output is regular.

PROOF: For any k input net there are exactly 2^k distinct input states and 2^k distinct initial complete states: $a_1, a_2, \cdots, a_{2^k}$. Let $b_1, b_2, \cdots, b_m$ ($0 \leqq m \leqq 2^{k'}$) be all the distinct complete states in which output o is on. Let σ' be the set of all transition sequences of complete states beginning with an a_i

and ending with a b_j. It is clear that $o(l) = 1$ if and only if the sequence of complete states assumed by the net over the first l moments of time belongs to σ'.

Where σ is the set of input sequences derived from σ', it follows by lemma 3 that the input sequence applied to the net over the first l moments of time belongs to σ if and only if the sequences of complete states assumed by the net over the first l moments of time belongs to σ'. Hence $o(l) = 1$ if and only if the input sequence applied to the net over the first l moments of time belongs to σ.

By lemma 1, taking as R the direct-transition relation, and as S the set of all complete states, the set of all transition sequences beginning with a particular a_i and ending with a particular b_j is regular. There are $m \cdot 2^k$ such regular sets of transition sequences, and their logical sum or disjunction, which is σ', is a regular set also. Since σ is derived from σ', lemma 2 assures us that σ is regular also.

Hence $o(l) = 1$ if and only if the input sequence applied to the net over the first l moments of time belongs to a regular set of input sequences. Which means that any set of input sequences realized by a net output is regular.

THEOREM 2. Any regular set of input sequences is realized by a net output.

PROOF: To prove theorem 2 it will be convenient to establish a somewhat stronger result in whose formulation we use the term "S-unit". An *S-unit* (starter) is defined to be a zero input net constructed from a delay element, a negation element, and a 2 input disjunction element, by identifying the negation output with one of the disjunction inputs, identifying the disjunction output with the delay input, and identifying the delay output with the negation input and with the other disjunction input. Output s of the S-unit is on only at time 1.

In what follows, every k input net N that has an output which realizes a regular expression of L^k will contain exactly one S-unit. And the output s of that S-unit will be identified with some input of an element of N not in the S-unit. Hence if the S-unit is detached from N the input which *was* identified with s is no longer identified with any output, and becomes the $k + 1$st net input i_{k+1} of the resulting $k + 1$ input net N'.

Now we can state the stronger result to be proved: Any formula Ω of any L^k is realized by an output o of a k input net N containing exactly one S-unit, whose detachment produces a $k + 1$ input net N' such that $o(l) = 1$ if and only if there is an l' such that

1) Ω denotes a regular set of input sequences which contains a sequence of l' k-tuples that is applied to the first k net inputs of N' over the l' consecutive moments of time $l - l' + 1, l - l' + 2, \cdots, l$; and
2) $i_{k+1}(l - l' + 1) = 1$. Our proof is by induction on the number of symbols in the formula, counting each occurrence of a letter or of $\vee$ or of $*$ as a single symbol.

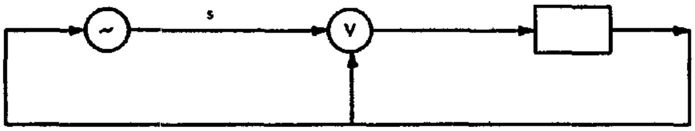

FIG. 5a

α) $n = 1$. Here the formula is one of the $1 + 2^k$ letters of L^k, and the regular set of sequences of k-tuples it denotes is either the empty set or a unit set containing a single k-tuple.

Case 1. If the formula is Λ it denotes the empty set, and is realized by the output o of a conjunction element whose $k + 2$ inputs are the k inputs of the net, the output of a negation element whose input is the first net input, and the output s of an S-unit. Since $o(t) = 0$ for every t, o realizes Λ.

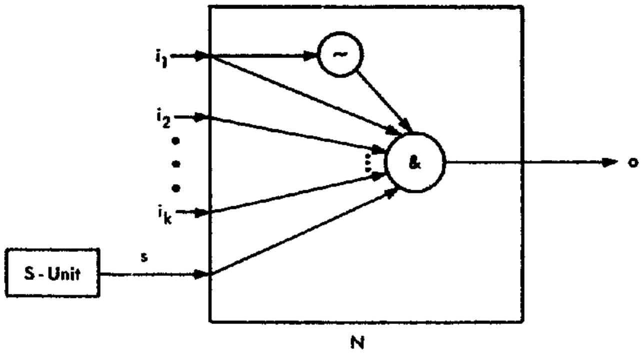

FIG. 5b

Case 2. If the formula is an A_i it denotes a unit set containing a single k-tuple, and is realized by the output o of a $k + 1$ conjunction element whose jth input is the jth net input i_j if the jth term of the k-tuple is 1, otherwise the output of a negation element whose input is the jth net input i_j, and whose $k + 1$st input is the output s of an S-unit. The following figure illustrates N where the k-tuple is $10 \cdots 10$:

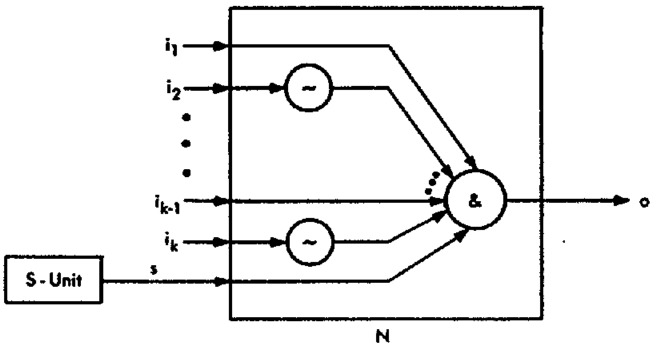

FIG. 5c

β) Here we assume the theorem true for any formula containing fewer than n symbols, and consider any formula Ω containing n (>1) symbols. Here $\Omega = (\Omega_1 \vee \Omega_2)$ or $\Omega = (\Omega_1 \cdot \Omega_2)$ or $\Omega = \Omega_1^*$.

Case 1. $\Omega = (\Omega_1 \vee \Omega_2)$. By the induction hypothesis, since Ω_1 and Ω_2 contain fewer than n symbols, they are realized by outputs o_1 and o_2 of k input nets N_1 and N_2 each containing exactly one S-unit, etc. Now Ω is realized by the output o of the k input net N constructed out of N_1', N_2', an S-unit, and a 2 input disjunction element whose output is o: by identifying o_1 with one input of the disjunction element and o_2 with the other, by identifying the output s of the S-unit with the $k + 1$st input of N_1' and the $k + 1$st input of N_2', and identifying input i_j of N_1' with input i_j of N_2' for $j = 1, 2, \cdots, k$.

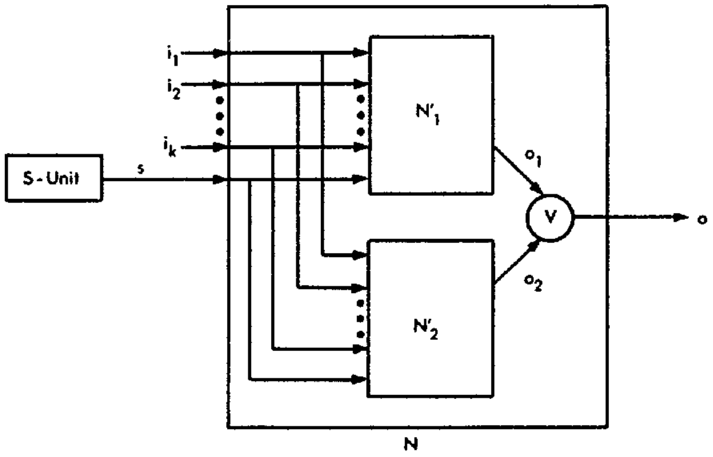

FIG. 5d

Case 2. $\Omega = (\Omega_1 \cdot \Omega_2)$. By the induction hypothesis, since Ω_1 and Ω_2 each contain fewer than n symbols, they are realized by outputs o_1 and o_2 of k input nets N_1 and N_2 each containing exactly one S-unit, etc. Now Ω is realized by the output o_2 of the k input net N constructed out of N_1', N_2', an S-unit, and a delay element: by identifying the output o_1 of N_1' with the input of the delay element, identifying the output s of the S-unit with the $k + 1$st input of N_1', identifying the delay output

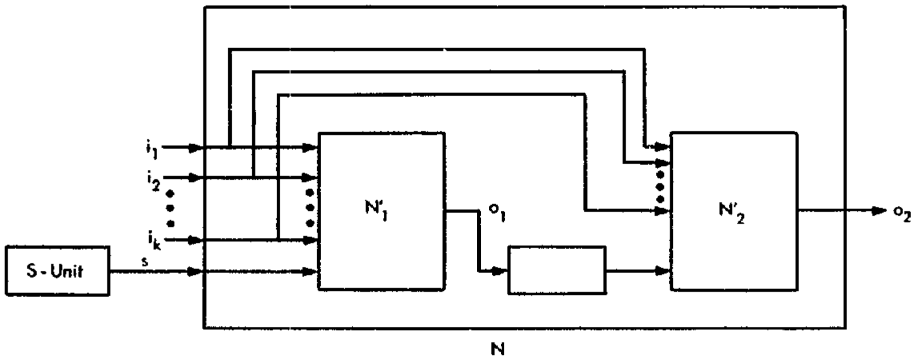

FIG. 5e

with the $k + 1$st input of N_2', and identifying input i_j of N_1' with input i_j of N_2' for $j = 1, 2, \cdots, k$.

Case 3. $\Omega = \Omega_1^*$. By the induction hypothesis, since Ω_1 contains fewer than n symbols, it is realized by output o_1 of k input net N_1 containing exactly one S-unit, etc. Now Ω is realized by the output o_1 of the k input net N constructed out of N_1', an S-unit, a delay element, and a 2 input disjunction element: by identifying the disjunction output with the $k + 1$st input of N_1', by identifying the output s of the S-unit with one input of the disjunction element and the delay output with the other, and identifying the output o_1 with the input of the delay element.

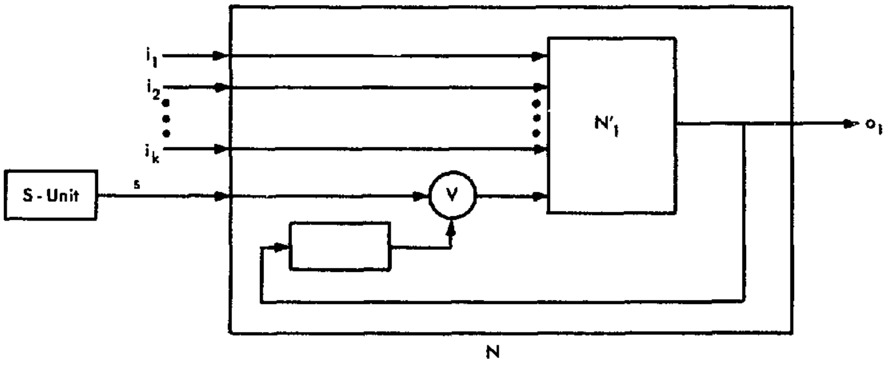

Fig. 5f

6. *Examples*

A. *Synthesis.* We use the algorithm embodied in the synthesis theorem to construct a net whose designated (in these cases its rightmost) output realizes $[(A \vee (B\cdot B)) \vee ((B\cdot A^*)\cdot B)]^*$, where $A = \{0\}$ and $B = \{1\}$.

If an output of a net N realizes F, we shall say that F' is realized by the corresponding output of the net N' formed by detaching the S-unit from N. Figures 6j and 6k are unabbreviated representations of the nets represented by figures 6g and 6i, respectively.

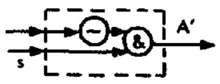

Fig. 6a

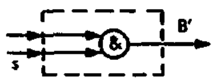

Fig. 6b

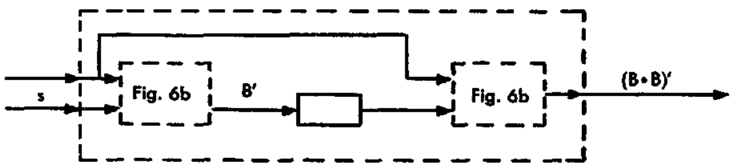

Fig. 6c

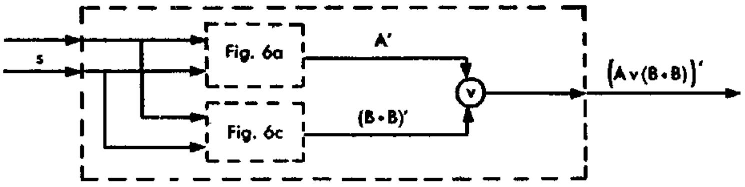

FIG. 6d

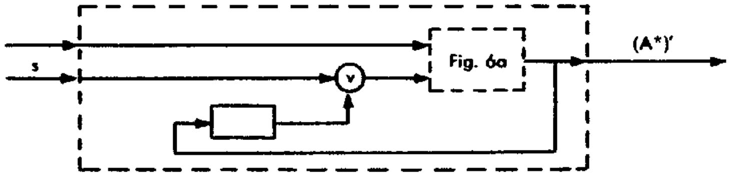

FIG. 6e

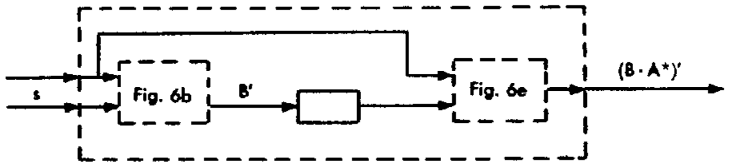

FIG. 6f

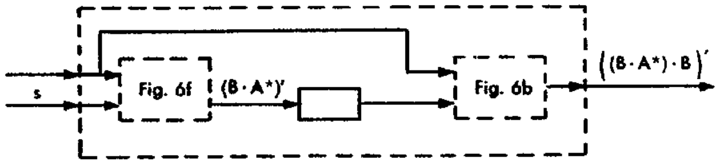

FIG. 6g

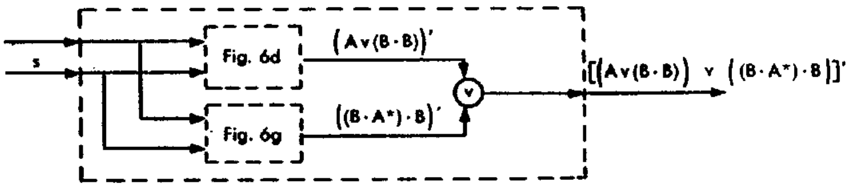

FIG. 6h

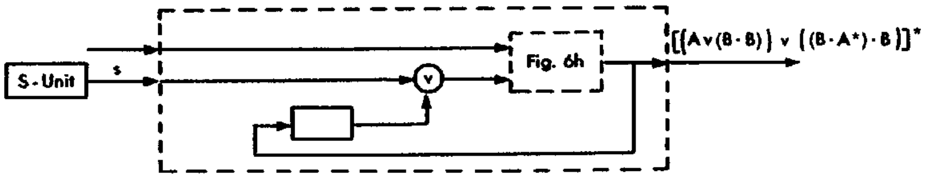

FIG. 6i

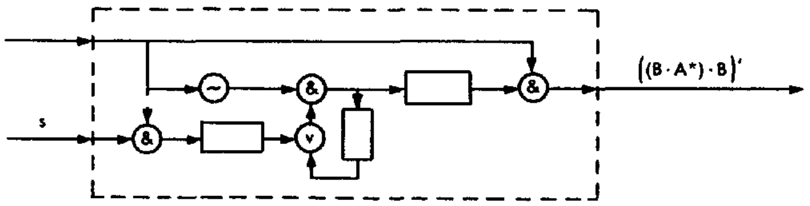

FIG. 6j

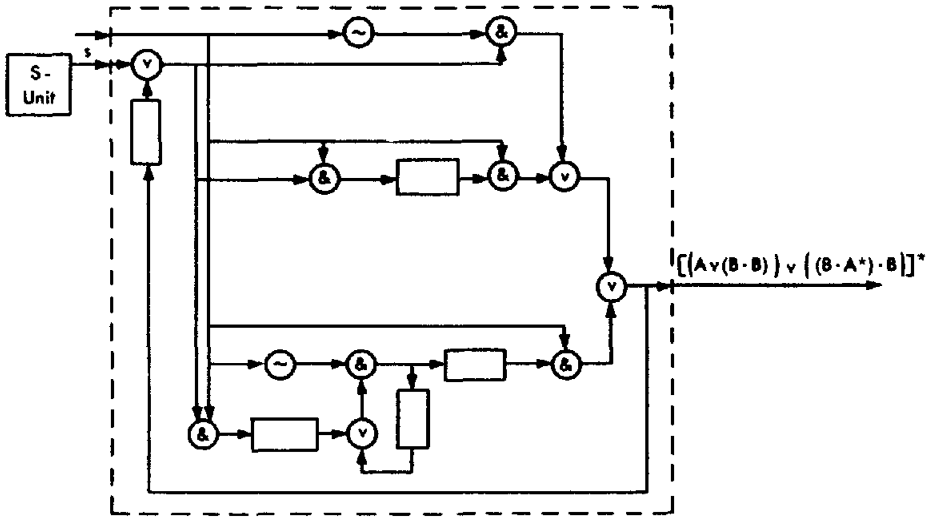

FIG. 6k

B. *Analysis.* We wish to find a regular expression which is realized by the designated output of the net below.

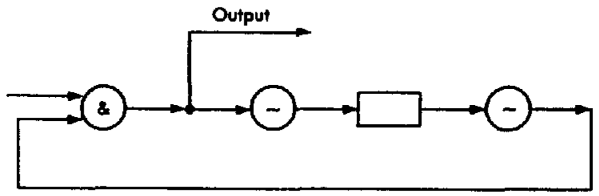

FIG. 6l

If we let the first bit represent the state of the input and the second bit represent the state of the delay output, then the direct transition relation (on complete states) is given by the table on page 194. The initial complete states are 00 and 10, while there is only one complete state in which the designated output is active, namely, 10. We, therefore, wish to find all *R*-sequences from 00 to 10 and also from 10 to 10.

In this particular example, it is easy to find the regular set by inspection.

R	
00	01
00	11
01	01
01	11
10	00
10	10
11	01
11	11

However, if we wish to use the algorithm embodied in lemma 1, we observe that the set of R-sequences from 11 to 10 in (i.e., constructed from elements of) $\{10, 11\}$ is empty since the "D" of the lemma is empty. For the same reason the set of R-sequences from 01 to 10 in $\{01, 10\}$ is empty. It follows that the set of R-sequences from 01 (11) to 10 in $\{01, 10, 11\}$ and from 00 to 10 in $\{00, 01, 10, 11\}$ is empty while from 10 to 10 in $\{00, 01, 10, 11\}$ is $\{10\} \vee \{10\}^* \cdot \{10\}$. (The "C" of the lemma is $\{10\}$.)

Applying now the projection of lemma 2, we obtain $\{1\} \vee \{1\}^* \cdot \{1\}$. The designated output of the net above realizes this regular expression.

7. *Comments*

Basic to Kleene's concept of event is his notion of tagged table. A table may be tagged in one of two ways, initial or non-initial. A set E of tagged k-columned tables represents a k-event. A k-columned table, m, of length l, is an *occurrence* of E if (1) m with the tag initial is an element of E, or (2) there exists an $r \geqq 1$ such that the table, tagged non-initial, consisting of rows $r, r + 1, \cdots, l - r + 1$ is an element of E. Two sets, E, F, of tagged k-columned tables are *equivalent*, i.e., represent the same event, if the set of all occurrences of E is the same as the set of all occurrences of F. An *event* (in the sense of Kleene) may be defined as an equivalence class of sets of tagged tables.

Three binary operations are defined on sets of tagged tables. It is not true in general, however, that, e.g., $E \cdot F$ is equivalent to $E' \cdot F'$ whenever the pairs E, E' and F, F' are equivalent. Thus these operations on sets do not carry over to events.

Kleene's regular expressions denote sets of tagged tables. Two regular expressions are *equal* if they denote the same set. Two regular expressions are *equivalent* if they denote equivalent sets.

Since the main complication in Kleene's theory is due to the concept of tagged tables, we sought to determine whether it was possible to obtain the theory without such a concept. This concept seemed to play an essential role in the synthesis theorem (Kleene's theorem 3, our theorem 2). We found that the concept can be eliminated as a primitive one and defined in terms of the concepts we adopt. The result is a great simplification in the notion of event, the fusion of the concepts equality and equivalence of regular expressions, and simpler operations $\vee$, $\cdot$, $*$.

Another complication in Kleene's development is contributed by the unit delay built into the elements of his nets. By separating logical units from delay units one obtains more flexible, in a sense stronger, nets. As a result our synthesis theorem is, in a sense, weaker than Kleene's; but it brings the essential nature of the result into sharper focus.

We use a singulary "*" operation rather than a binary one because the operation Kleene uses seems "essentially" singulary and because the singulary operation simplifies the algebra of regular events. It should be noted that the singulary and binary star operations are interdefinable.

APPENDIX

In general, there are $k \geqq 0$ inputs and $n > 0$ outputs in a net or finite automaton. Each input and each output may assume a finite number of states. If the states of every input for every moment of time (given by the natural numbers) are specified for a net, then the state of every output at every moment of time is determined. Assuming that the inputs (outputs) are ordered and that the ith input (output) is capable of assuming $a_i(b_i)$ states the input (output) over time t, $1 \leqq t < l + 1$, may be described by a k (n)-rowed matrix whose ith row entries denote any of $a_i(b_i)$ states, and with l columns, where l is a positive integer or infinity. The behavior of the net or automaton may then be described in the following alternative ways:

(a) A transformation (mapping, function) from k-rowed infinite matrices to n-rowed infinite matrices with the property P: whenever the first l columns of two k-rowed matrices are the same, the associated n-rowed matrices have the same first l columns.

(b) A transformation which associates with k-rowed, l-columned (k is fixed; l varies over all the positive integers) matrices n-rowed, l-columned matrices and which has property P.

(c) A transformation which associates with k-rowed finite matrices, n-rowed one-columned matrices.

(d) An n-tuple of (ordered) partitions of all the k-rowed finite matrices. The ith, $1 \leqq i \leqq n$, (ordered) partition consists of a b_i-tuple of sets.

If T_a is a transformation of type (a) then a transformation of type (b), T_b, may be uniquely associated with it by defining $T_b(x_m) = (T_a(x))_m$ where x is an infinite matrix whose first m columns is x. With each T_b may be associated a transformation T_c, whose domain is the same as the domain of T_b, and whose value when applied to a m-columned matrix is the mth column of $T_b(x)$. With each T_c an n-tuple of (ordered) partitions of all the k-rowed finite matrices may be associated as follows:

The jth set, $1 \leqq j \leqq b_i$, of the ith entry, $1 \leqq i \leqq n$, of the n-tuple consists of all finite k-rowed matrices x such that the ith row of $T_c(x)$ is the jth output state. It is clear with a little reflection that these four modes of expressing behavior are in biunique correspondence.

Burks and Wright [2] use method (a) to express behavior with $n = 1$, all $a_i = 2$, $b_1 = 2$ and the infinite k-rowed matrices regarded as k-tuples of infinite sequences (of zeros and ones). Burks and Wang [1] use method (a) (among others) with all a_i, $b_i = 2$ but with the infinite matrices regarded as infinite sequences of k (n)-tuples. E. F. Moore [4] uses method (b) with $k = 1$ and $n = 1$ but with a_1 and b_1 arbitrary. S. C. Kleene [3], suitably interpreted, uses method (d) with $n = 1$, $b_1 = 2$ and the $a_i = 2$. G. H. Mealy [5] uses method (b) with the b_i, $b_j = 2$ and with the finite k (n)-rowed matrices interpreted as finite sequences of k (n)-tuples. We use, in this paper, (d) with $n = 1$, $b_1 = 2$ and the $a_i = 2$ and interpret the finite k-rowed matrices as finite sequences of k-tuples.

REFERENCES

[1] A. W. Burks and H. Wang, The logic of automata, *Jour. Assoc. Comp. Mach.*, Part I: *4* (1957), 193-218; Part II: *4* (1957), 279-297.

[2] A. W. Burks and J. B. Wright, Theory of logical nets, *Proc. IRE 41* (1953), 1357-1365.

[3] S. C. Kleene, Representation of events in nerve nets and finite automata, *Automata Studies*, edited by C. E. Shannon and J. McCarthy, Princeton University Press, 1956, pp. 3-41.

[4] E. F. Moore, Gedanken-experiments on sequential machines, *Automata Studies*, edited by C. E. Shannon and J. McCarthy, Princeton University Press, 1956, pp. 129-153

[5] G. H. Mealy, A method for synthesizing sequential circuits, *The Bell System Technical Journal*, *34* (1955), 1045-1079.