Conclusion

It has been shown that algebraic methods are readily applicable to threshold-logic synthesis of three-argument switching functions. Indeed, most of the representations derived above are undoubtedly minimal. When (5) is used in its general form and when certain other theorems not stated above are also used, it is possible to derive threshold-operator representations for functions of more than three arguments. For such functions, however, it becomes increasingly difficult to derive representations that have a reasonable claim to being minimal. The difficulty usually lies in knowing which of many possible paths to take in dealing with the problem.

W. H. Hanson
Univac Div. of Sperry Rand Corp.
St. Paul, Minn.

On the Construction of Sequential Machines from Regular Expressions*

Summary

Methods have been described [1], [2] for constructing sequential machines directly from regular expressions. In the construction, every machine is assumed to have an input terminal, an output terminal and a special starting input terminal. The methods described [1], [2] involve a recursive procedure. For example, if machines M_E and M_F realize the regular expressions E and F respectively, then the machine realizing the concatenation, $R=EF$, is obtained [1], [2] by connecting the output of M_E to the starting terminal of M_F.

In this paper it is shown that a submachine generated by the algorithm in the middle of the synthesis procedure cannot be replaced in general, by an equivalent machine. It is felt that the proofs of the original algorithms [1], [2] are rather brief and do not point out the difficulties which might arise if proper precautions are not taken. To remedy this situation, this paper examines the construction in detail. The notion of recurrent realization of a regular expression is introduced, and a theorem is proved that the construction is valid if the proper precautions are taken.

Introduction

We are concerned with regular expressions over the alphabet $\{0, 1\}$, defined recursively as follows [2]–[4]:

1) 0, 1, λ and ϕ are regular expressions.
2) If P and Q are regular expressions, then so are $(P+Q)$, (PQ) and P^*.
3) Nothing else is a regular expression unless its being so follows from a finite number of applications of steps 1) and 2).

Regular expressions denote sets of sequences; λ is the set consisting of the sequence of zero length (or the empty sequence), ϕ is the empty set of sequences, $(P+Q)$ denotes the union of the sets denoted by P and Q, (PQ) denotes the concatenation of P and Q and P^* can be defined as an infinite sum

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

(Note that Copi, Elgot and Wright [1] define P^* without including λ, *i.e.*, $P^*=P+PP+\cdots$. Consequently, their construction differs slightly from ours which follows the work of Arden [2].)

The sequential machines under consideration are fixed, finite, deterministic and synchronous. The machines are always started in a prescribed initial state at $t=0$. Also, every machine has a starting terminal on which a starting pulse [2] appears at $t=0$. The symbols of an input sequence occur at times $t=1, 2, \cdots, p$.

A machine is said to accept a sequence s of symbols, if and only if, when the machine is in its starting state at $t=0$, when the starting pulse appears at $t=0$ and when the input symbols at times $1, 2, \cdots, p$ constitute s, then the output at the end of the sequence is $Z=1$. A machine is said to *realize* a regular expression R if and only if it accepts all the sequences denoted by R, and only such sequences.

The Recursive Construction

Fig. 1 shows the symbols used to represent AND gates, OR gates, inverters and delays. The construction described in the literature [1], [2] is recursive in nature and proceeds by induction. First, the machines realizing the regular expressions 0, 1, λ and ϕ can be built as shown in Fig. 2. The starting pulse (occurring only at $t=0$, by assumption) can be identified with λ, the set of sequences consisting of the sequence of zero length. For the details, see Arden [2] and Brzozowski [4].

The constructions of Fig. 2 form the basic step of the induction. Now assume that machines realizing the regular expressions E and F have been constructed. Then the machines of Fig. 3 are used [1], [2] to realize the regular expressions $E+F$, EF and E^*. A machine realizing the regular expression R is denoted by M_R.

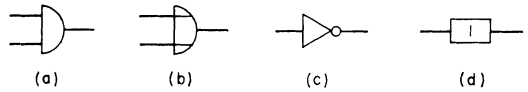

Fig. 1—Symbols for logical devices: (a) AND gate, (b) OR gate, (c) Inverter, (d) Unit delay.

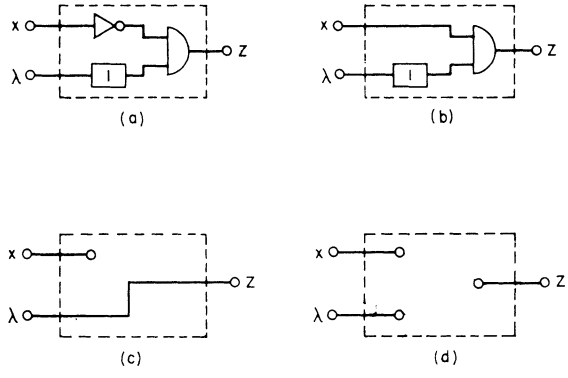

Fig. 2—Machines realizing the regular expressions: (a) $R=0$, (b) $R=1$, (c) $R=\lambda$, (d) $R=\phi$.

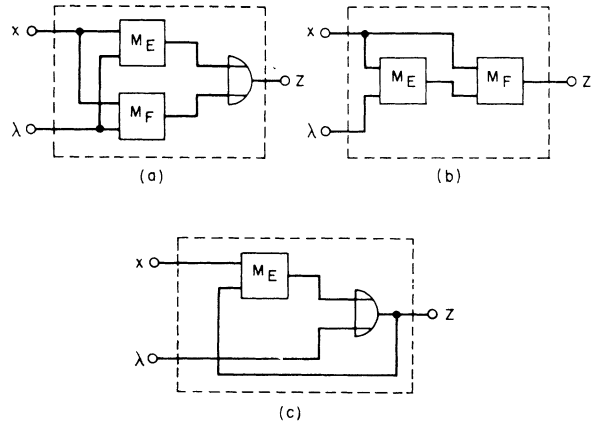

Fig. 3—Machines realizing: (a) $R=E+F$, (b) $R=EF$, (c) $R=E^*$.

* Received October 18, 1962; revised manuscript received March 25, 1963.

The Difficulties

Consider the machine M_λ shown in Fig. 4. Assuming that the feedback loop has an initial value of 0, it is easily verified that the network realizes the regular expression $R=\lambda$; *i.e.*, an output $Z=1$ occurs if and only if there has been a starting pulse at $t=0$. Note, however, that after the occurrence of the starting pulse at $t=0$, the output of M_λ is always $Z=0$.

Suppose now we construct a machine N_F by concatenating the machine M_λ and a machine M_F, realizing the regular expression F. (This is illustrated by Fig. 3(b) where M_E is replaced by M_λ.) The starting pulse applied to N_F at $t=0$ is transmitted directly through M_λ to the starting terminal of M_F. Thus, if M_F realizes F, N_F also realizes F.

The difficulty arises when we construct a machine to realize EF by using a machine M_E realizing E, followed by N_F realizing F, as shown in Fig. 5. Suppose, for example, that $E=1+11$ and $F=0$. EF is then $10+110$. If we apply the sequence 110 to M, that sequence is not accepted. This follows because E occurs at $t=1$ and $t=2$, but, due to the presence of M_λ, the output of M_E at $t=2$ is not transmitted through M_λ to the starting terminal of M_F. Thus, the machine M does not realize EF, in general, although it is the concatenation of machines M_E and N_F, realizing E and F respectively.

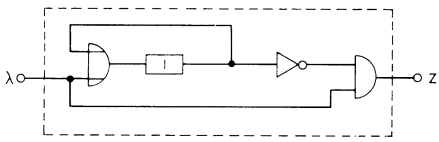

Fig. 4—Machine M_λ.

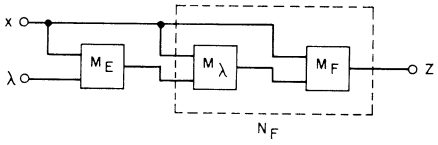

Fig. 5—Machine M.

The Solution

Definition: A machine M (with input x and a starting terminal) *recurrently realizes*[1] a regular expression E if and only if M has the following property: An output $Z=1$ is produced by M at time t if and only if there has been a signal on the starting terminal at some time $t-d$ *and* the input sequence at times $t-d+1, \cdots, t$ is a member of E.

It is clear that if M recurrently realizes E then M also realizes E (*i.e.*, accepts all the sequences of E and only such sequences, when the starting terminal is energized only once at $t=0$). The converse, however, is not true, in general, as exemplified by the machine M_λ. For the recurrent realization of a regular expression it is assumed that a signal on the starting terminal may be applied an arbitrary number of times.

Theorem: If machines M_E and M_F recurrently realize the regular expressions E and F respectively, then the machines M_{E+F}, M_{EF} and M_{E^*} of Fig. 3 recurrently realize the regular expressions $E+F$, EF and E^* respectively.

Proof: The proof is trivial for M_{E+F} by inspection of the diagram of Fig. 3(a). Consider now M_{EF}. If $Z=1$ at time t, then there must have been a sequence $f\in F$ at times $t-d_1+1, \cdots, t$ and a signal at $t-d_1$ on the starting terminal of M_F, because, by assumption, M_F recurrently realizes F. But this implies an output from M_E at time $t-d_1$ and, consequently, a sequence $e\in E$ at times $t-d_2+1, \cdots, t-d_1$ and a signal at $t-d_2$ on the starting terminal of M_E. Summing up, we see that the presence of $Z=1$ at time t implies the presence of a sequence $ef\in EF$ at times $t-d_2+1, \cdots, t-d_1 \cdots, t$ and a signal on the starting terminal of M_{EF} at time $t-d_2$. Conversely, if a signal is applied on the starting terminal of M_{EF} at $t-d_2$ and is followed by a sequence $s\in EF$, it is clear that the output will be $Z=1$ at the end of s. Hence, M_{EF} recurrently realizes EF. A similar analysis applied to the machine M_{E^*} of Fig. 3(c) shows that M_{E^*} recurrently realizes E^*, if M_E recurrently realizes E. Thus, the theorem is true.

We are now in a position to describe the correct construction for any regular expression. It is easily verified that the machines of Fig. 2(a)–(d) *recurrently* realize the regular expressions 0, 1, λ and ϕ respectively. If these are used as the initial building blocks then the recursive procedure is applicable, for, if M_E and M_F *recurrently* realize the regular expressions E and F, then the machines of Fig. 3 *recurrently* realize $E+F$, EF and E^*, according to the theorem. Thus, the construction is valid provided that each machine M_R *recurrently* realizes R.

Conclusion

We have examined in detail the construction of sequential machines from regular expressions. It was felt that the original papers [1], [2] describing the construction had not foreseen the difficulties described above and had not proved the validity of the construction. It has been shown that the construction is indeed valid if the property of recurrent realization is preserved.

J. A. Brzozowski
Dept. of Elec. Engrg.
Princeton University
Princeton, N. J.
At present:
University of Ottawa
Ottawa, Ontario, Canada
J. F. Poage
Dept. of Elec. Engrg.,
Princeton University
Princeton, N. J.
At present:
Bell Telephone Labs., Inc.
Holmdel, N. J.

References

[1] I. M. Copi, C. C. Elgot and J. B. Wright, "Realization of events by logical nets," *J. Assoc. Comp. Mach.*, vol. 5, pp. 181–196; April, 1958.
[2] 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.
[3] 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.
[4] J. A. Brzozowski, "A survey of regular expressions and their applications," IRE Trans. on Electronic Computers, vol. EC-11, pp. 324–335; June, 1962.
[5] E. F. Moore, "Gedanken-experiments on sequential machines," in "Automata Studies," C. E. Shannon and J. McCarthy, Eds., Princeton University Press, Princeton, N. J., Study 34, pp. 129–153; 1956.
[6] H. Yamada, "Disjunctively linear logic nets," IRE Trans. on Electronic Computers, vol. EC-11, pp. 623–639; October, 1962.

[1] In the work of Yamada [6], who discusses similar problems from a different viewpoint, machines having "disjunctively linear behavior" are ensured to have the property of recurrent realization.

A Note On Threshold Device Error Analysis*

Threshold Device Model

The threshold device to be considered in this communication is a device with n-inputs $(x_1, \cdots, x_n)$ and one output (Z_q). (See Fig. 1.) The inputs and output are assumed to be binary variables which can have either of two values ('1' or '0').

The following assignment of physical values will be made to the logical input values: ('1', '0') = (b, c) where b and c are real numbers such that $b>c$.

The element of Fig. 1(a) will be called the *summer*. Call $a_1, \cdots, a_n$ the *weights* and T the *threshold*. The operation of the summer is characterized by (1).

$$Z_s = x_1a_1 + \cdots + x_na_n - T \qquad (1)$$

where T and $a_i (1\leq i\leq n)$ are real numbers and the operations are ordinary real number addition and multiplication.

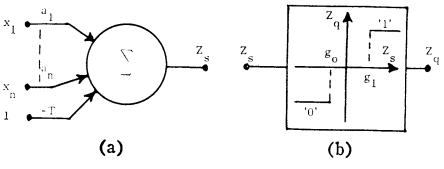

Fig. 1—A model of the threshold device.

The element of Fig. 1(b) will be called the *quantizer*. The operation of the quantizer is characterized by (2a) and (2b). For $g\geq 0$ [$g=g_1-g_0$ in Fig. 1(b)],

$$Z_q = \begin{cases} \text{'1'} & \text{for } Z_s > g_1 \\ \text{unspecified} & \text{for } g_1 \geq Z_s \geq g_0 \\ \text{'0'} & \text{for } g_0 > Z_s. \end{cases} \qquad (2a)$$

For $g<0$,

$$Z_q = \begin{cases} \text{'1'} & \text{for } Z_s > g_1 \\ \text{'0'} & \text{for } Z_s \leq g_1 \end{cases} \Bigg\} \text{If the last value of } Z_q = \text{'1'}$$
$$\begin{cases} \text{'1'} & \text{for } Z_s \geq g_0 \\ \text{'0'} & \text{for } Z_s < g_0 \end{cases} \Bigg\} \text{If the last value of } Z_q = \text{'0'}. \qquad (2b)$$

Since input signals, voltage bias, branch conductances, etc., will vary in practice, the following variations are defined:

d = weight variation (expressed as a fraction of the weight)

* Received May 25, 1962; revised manuscripts received September 13, 1962 and December 10, 1962.