The Logic of Automata—Part II*

ARTHUR W. BURKS AND HAO WANG

University of Michigan, Ann Arbor, Mich.

3. TRANSITION MATRICES AND MATRIX FORM NETS

3.1. *Transition Matrices*

The transition part of a net controls the passage of the net from state to state and is therefore the heart of the net. In this subsection we introduce "the transition matrix," a table which describes a net by showing the various ways in which it may pass from one delay state to another.

We use a characterizing table with M input (state) words I, N internal-state words S, and $M \times N$ rows, each of the form $\ll I, S>, S'>$, to define N^2 *direct-transition expressions* I_{ij} as follows. I_{ij} is a disjunction of all those I_k such that $\ll I_k, S_i>, S_j>$ is a row of the characterizing table; if there are no such I_k, then I_{ij} is ϕ ("the false"). That is, I_{ij} is a disjunction of all those input words (if any) which can cause the net to pass from state S_i at time t to state S_j at time $t + 1$; it is allowed that i equals j. It is clear that each I_{ij} is a disjunctive normal form expression of the input function variables.

Note that the direct-transition expression "ϕ" is distinct from the direct-transition expressions "0", "00", "000", etc.; the former means that no direct transition between the two states is possible, while the latter mean that such a transition is brought about by making all the inputs zero.

A *direct transition* from S_i to S_j in a net is a passage from state S_i at t to state S_j at $t + 1$. Such a transition is possible only in the case where $I_{ij} \not\equiv \phi$. We say that $<I_k, S_i>$ (or $\widehat{I_k S_i}$) at time t *directly produces* S_j at time $t + 1$ only in the case where the net makes a direct transition from S_i to S_j under the influence of input I_k at t.

A *transition* from S_i to S_j in a net is a passage from state S_i at t to state S_j at $t + w$ $(w > 0)$. Such a transition is possible only where there exists a sequence of direct-transition expressions, none of which are ϕ, of the form I_{ia_1}, $I_{a_1a_2}, \cdots, I_{a_wj}$. We say that $\widehat{I_{ia_1} S_i}(t), I_{a_1a_2}(t + 1), \cdots, I_{a_wj}(t + w - 1)$ *produces* S_j at $t + w$ if and only if there is a transition from $S_i(t)$ to $S_j(t + w)$ under the direction of the listed inputs; this is a transition of w *steps* (or, alternatively, a transition of *length* w).

It is convenient to arrange the information in an $M - N$ characterizing table in a *direct-transition matrix of order* N by arranging the N^2 direct-transi-

* Received December, 1956. Part I, including References, appeared in the April issue, pp. 193-218. This research was supported by the United States Air Force through the Air Force Office of Scientific Research of the Air Research and Development Command under Contract No. AF 18(603)-72.

tion expressions in square array. The following is a *direct-transition matrix schema* of order four:

$$\begin{bmatrix} I_{00} & I_{01} & I_{02} & I_{03} \\ I_{10} & I_{11} & I_{12} & I_{13} \\ I_{20} & I_{21} & I_{22} & I_{23} \\ I_{30} & I_{31} & I_{32} & I_{33} \end{bmatrix}.$$

It is clear that a direct-transition matrix presents the same information as a characterizing table, but in a different way. For many purposes this form is more convenient, because it reflects the fact that the basic behavior of an automaton consists of a sucession of transitions from one state to another. (Since a transition matrix is equivalent to a characterizing table, the formulae given in section 2.3 for the number of $M - N$ automaton characterizing tables apply here also.)

The information contained in a complete table can also be expressed in matrix form. Since for an abstract automaton the complete table is the characterizing table, the matrix derived from the complete table is a direct-transition matrix.

We give an example of a transition matrix. A matrix for a four-stage cyclic counter is

$$\begin{array}{c} \\ S_0 \\ S_1 \\ S_2 \\ S_3 \end{array} \begin{array}{c} \begin{array}{cccc} S_0 & S_1 & S_2 & S_3 \end{array} \\ \begin{bmatrix} I_0 & I_1 & \phi & \phi \\ \phi & I_0 & I_1 & \phi \\ \phi & \phi & I_0 & I_1 \\ I_1 & \phi & \phi & I_0 \end{bmatrix} \end{array}.$$

(We have added the S's as a mnemonic aid, but given a conventional ordering of them, they need not be written in.) Thus, an input I_0 (e.g., 0) causes the counter to stay in its given state, while an input I_1 (e.g., 1) causes it to advance to the next state (modulo 4). All other entries are ϕ's since they represent cases where direct transitions are impossible.

3.2. *Matrix Form Nets*

In this subsection we will present a net form closely related to the transition matrix. Our presentation is in two steps; the first is to construct a decoded normal form net.

Consider for a moment a coded normal form net in relation to its coded characterizing table. Each S is associated with a D, and in general any arbitrary number of bits of D may be unity. Consider in contrast a decoded normal form characterizing table. Exactly one bit of each S is unity, which suggests associating each state S primarily with a single junction. That can be done for an N state decoded normal form table as follows. Let $S = \overbrace{B_0 B_1} \cdots \overbrace{B_{N-1}}$ as be-

fore, and form a net with N delay elements so that $D = E_0 \frown E_1 \frown \cdots \frown E_{N-1}$. Of the 2^N delay words D, we use only $N + 1$, namely, $D_0(= 000 \cdots)$ and the N words having exactly one bit which is unity and all other bits zero ($100 \cdots$, $010 \cdots$, etc.). We next construct a junction C which is the output of a disjunctive element ("or") fed by E_0 and by the input-independent transformation $100 \cdots$. Hence,

$$C(0) \equiv 1$$

$$C(t) \equiv E_0(t), \quad \text{for all} \quad t > 0.$$

We now associate B_0 with C, and each B_i with E_i for $0 < i < N$; that is, we equate $B_0 \frown B_1 \frown \cdots \frown B_{N-1}$ and $C \frown E_1 \frown \cdots \frown E_{N-1}$. Hence we have N junctions (C, E_1, $\cdots$, E_{N-1}) such that each state S is associated primarily with one junction: namely, that junction which is active when the net is in that state. These junctions are called the *state junctions* of the net. See figure 2, where the state junctions are labeled C, E_1, and E_2, and are also labeled with the states (S_0, S_1, and S_2) which they "represent."

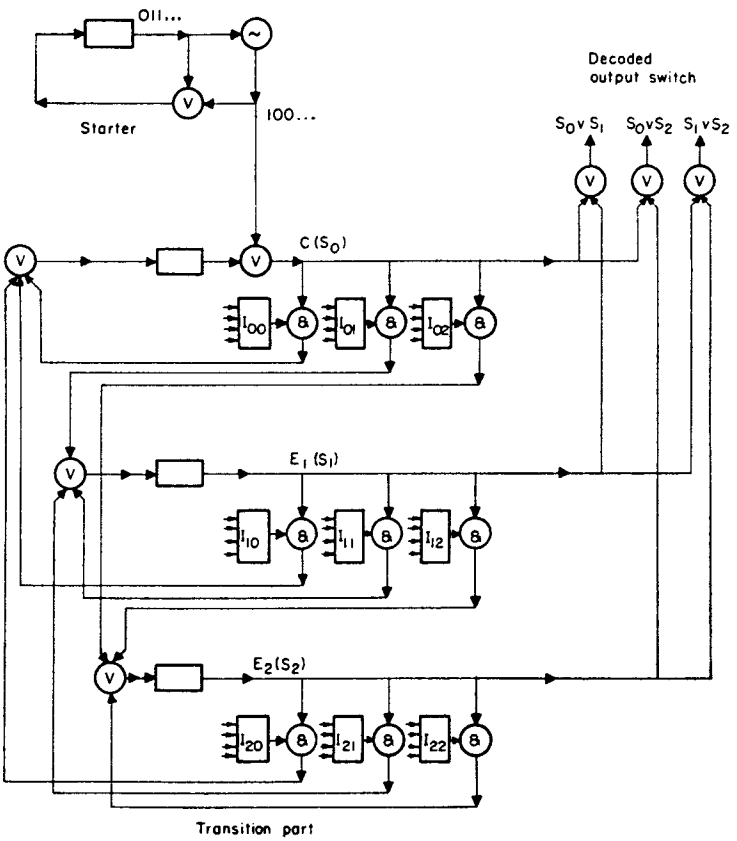

FIG. 2. Decoded normal form net

We now wish to construct a net containing wires C, E_1, $\cdots$, E_{N-1}, so connected that at each time exactly one of them is active (in state one) while all others are zero, and with the following inductive property.

(A) Junction C (representing S_0) is active at time 0.

(B) For any time t, if the junction labeled S_i (i.e., C for $i = 0$, E_i for $i > 0$) is active at time t, the net input at time t is I_k, and $\ll I_k, S_i>, S_j>$ is a row of the characterizing table, then the junction labeled S_j will be activated at time $t + 1$.

To realize condition (A), we construct a *starter* (see fig. 2) to produce the input-independent transformation $100\cdots$. The starter output is then disjoined with E_0 to produce C. This will insure that $C(0) \equiv 1$, and that $C(t) \equiv E_0$ for $t > 0$. The starter, which adds another delay element to the net, may be constructed without a cycle (see section 4.3).

The realization of (B) is more complicated. Since there is a state junction for each state (C for S_0, E_{i+1} for S_{i+1}), the concept of a direct-transition word is useful here. At each state junction S_i we build N switches such that:

$I_{i0}\frown S_i$ directly produces S_0 (i.e., activates C at the next moment of time);
$I_{i1}\frown S_i$ directly produces S_1 (i.e., activates E_1 at the next moment of time);
$\vdots$
$I_{i,N-1}\frown S_i$ directly produces S_{N-1} (i.e., activates E_{N-1} at the next moment of time).

A net to accomplish this purpose is shown in figure 2, which is actually a *net schema* rather than a net, since the I_{ij} are not specified.

The boxes labeled with the direct-transition expressions I_{ij} are called *direct-transition switches*. (Note that these direct-transition switches are different from the direct-transition switch of figure 1, though both kinds of switches play the same basic role in a net.) A direct-transition switch I_{ij} has an output at time t if and only if the input to the net is represented by a disjunct of I_{ij}. Every such switch can be made of a disjunction driven by conjunctions of positive and negative inputs. For example, let I_{10} be 0101 $\vee$ 0110 $\vee$ 1111, which may be written as $\bar{A}_0A_1\bar{A}_2A_3 \vee \bar{A}_0A_1A_2\bar{A}_3 \vee A_0A_1A_2A_3$, and the latter is readily realized by a switch. Of course the number of inputs may vary from net to net; we will usually show four inputs in our figures (as we do in fig. 2).

If a particular I_{ij} is ϕ, the switch I_{ij} and the conjunction it drives may be deleted. If a net always passes directly from a state S_i to a state S_j, no matter what the inputs are (or because there are no inputs), then we can run a direct line from the S_i junction to the input of the delay element driving the S_j junction. An input-independent net thus becomes a string of delays (corresponding to the initial part of the input-independent transformation), driven by a starter and driving a cycle of delays (corresponding to the periodic part of the function); cf. Theorem XIII and the accompanying figure of Burks and Wright [1], p. 1363.

We call a net of the form of figure 2 a *decoded normal form net*. We will first

explain why we call this form "decoded" in contrast to the "coded" form described earlier, and then discuss the exact relation between decoded and other nets. The terminology is justified by the fact that in a coded normal form net the numbers representing delay states (i.e., the D's) appear in coded form, while in the decoded normal form net the numbers representing the delay states (the D's, except that E_0 is replaced by C at time zero) appear in decoded form, in the sense in which the terms "coded" and "decoded" are used in switching theory. A *decoding switch* is a switch with the same number of output junctions $C_0, \cdots, C_{N-1}$ as there are admissible input words $I_0, \cdots, I_{N-1}$, and so connected that when the input state is I_n the output junction C_n is active and all other outputs are inactive. (This is a special case of the nets discussed in Burks, McNaughton, *et al.* [13]). The information on the inputs is in *coded* form, while that on the outputs is *decoded*. In our coded and decoded normal form nets, however, the coding and decoding applies to delay outputs rather than to switch outputs.

Given any automaton net, we can construct a decoded normal form net which models that automaton net in the sense of having junctions which behave the same. Let us begin with the transition part of the given automaton net. Suppose it has n delay output junctions $F_0, F_1, \cdots, F_{n-1}$ and N admissible states $S_0, S_1, \cdots, S_{N-1}$ ($N \leqq 2^n$). We next construct the transition part of a decoded normal form net with junctions $C, E_1, \cdots, E_{N-1}$, such that the i-th bit from the left of $\overbrace{C E_1 \cdots E_{N-1}}$ is unity when the original automaton is in state S_{i-1}. Any function F_j is equivalent to a disjunction $S_{a_1} \vee S_{a_2} \vee \cdots \vee S_{a_k}$ of just those states for which F_j has the value one. Hence by disjoining the appropriate state junctions of the decoded normal form net we can obtain a junction F_j' such that $F_j'(t) \equiv F_j(t)$ for all t. Figure 2 shows a *decoded output switch* which realizes $S_0 \vee S_1$, $S_0 \vee S_2$, and $S_1 \vee S_2$. The "single-disjunct disjunctions" S_0, S_1, and S_2 are already represented in figure 2, and the input-independent outputs ($\bar{S}_0$ & $\bar{S}_1$ & $\bar{S}_2$) and ($S_0 \vee S_1 \vee S_2$) (i.e., $000 \cdots$ and $111 \cdots$) are readily obtained from the net if needed.

This shows how to construct junctions of a decoded normal form net which behave the same as the delay output junctions of the original net. Any other junction of the original net whose behavior at time t does not depend on the state of the inputs at time t can be treated in the same way. For the remaining junctions we can build an output switch fed both by the decoded output switch and the net inputs. Alternatively, the decoded output switch can be replaced by a switch driven by the state junctions and the net inputs. Switches of these various kinds are allowed as parts of decoded normal form nets. We can now incorporate these results in a theorem.

THEOREM 2: For any well-formed net with junctions $C_0, C_1, \cdots$, one can construct a decoded normal form net with junctions $C_0', C_1', \cdots$ such that $C_i(t) \equiv C_i'(t)$ for all i and t.

The transition part of a decoded normal form net can be drawn in matrix form to bring out its relation to the transition matrix. In figure 3 this is done for a transition matrix of order 4; the result is called a *matrix box*. The dis-

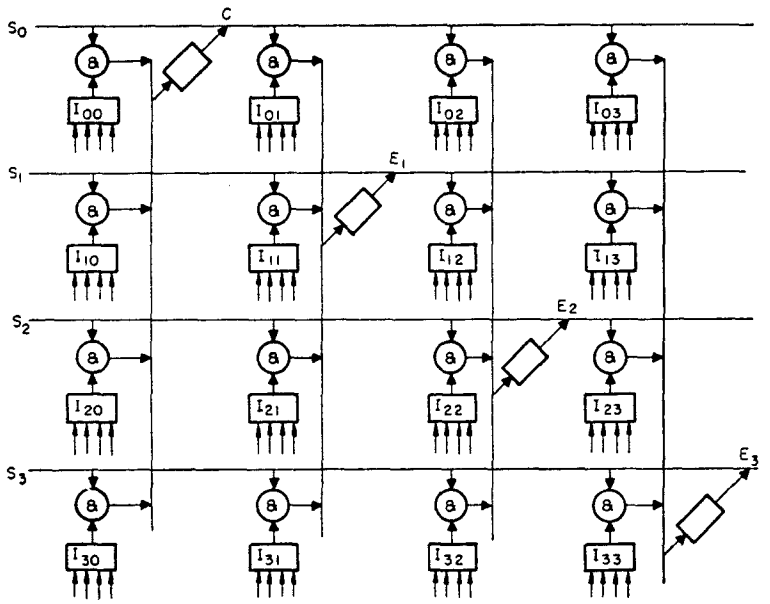

FIG. 3. Matrix box of order 4

junction elements of figure 2 are omitted by the convention that several wires can drive a line (see Burks and Copi [2], p. 307). A normal form net with the transition part put in matrix-box form is called a *matrix form net.* Figure 4 is an example which lacks an output switch.

A particular net of order four may be obtained from the schema of figure 3 by substituting the appropriate switches for the direct-transition-switch schemata. We illustrate this with the four-stage cyclic counter whose transition matrix was given in section 3.1. If we let input $I_0 \equiv 0$ and input $I_1 \equiv 1$ (so the counter counts pulses rather than the absence of pulses), we get the matrix box of figure 4. Each $I_{ii} \equiv I_0 \equiv 0$, so we replace these direct-transition-switch schemata by negation elements. For each i, j such that $j = i + 1$ modulo four, $I_{ij} \equiv 1$, so we replace these direct-transition-switch schemata by single input lines. All other transition-switch schemata correspond to ϕ's in the transition matrix defining the counter, so these are dropped; e.g., no direct transition from S_2 to S_1 is possible, so there is no direct coupling from S_2 to ϵ_1. It is manifest from figure 4 that the counter stays in its prior stage when the input is zero, but advances to the next stage (modulo four) when the input is one.

3.3 *Some Uses of Matrices*

The discovery that matrices may be used to characterize automata nets opens up a number of interesting lines of investigation. In the present subsection we will discuss a few applications of matrices to the analysis of automata nets.

A direct-transition matrix (whose elements are direct-transition expres-

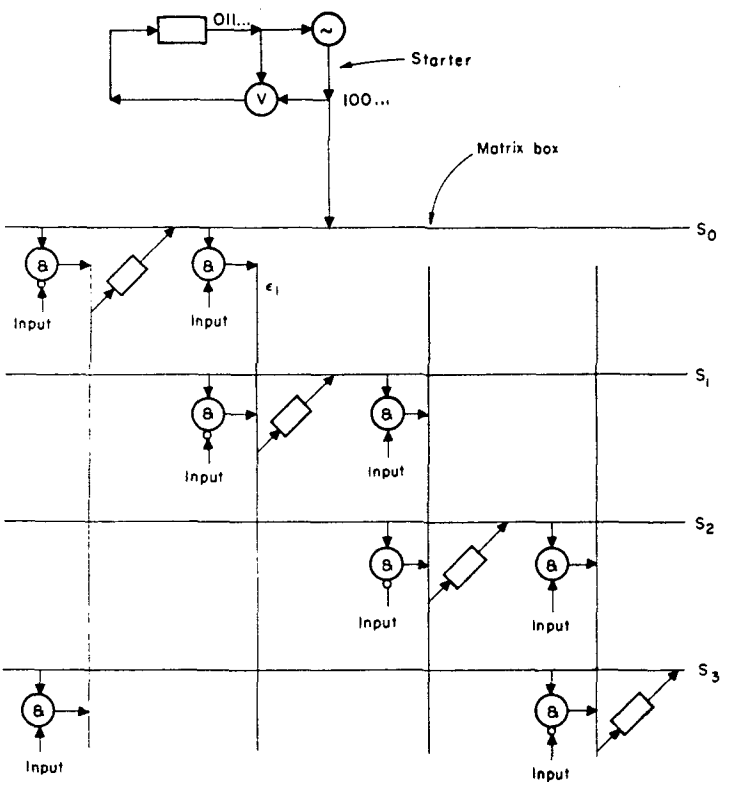

FIG. 4. Matrix form binary counter

sions) characterizes the direct transitions of an automaton. We will first establish some properties of this matrix, and then generalize the concepts of direct-transition expression and direct-transition matrix to cover transitions of arbitrary length.

Each row of a direct-transition matrix is a partition of M input words I_0, I_1, $\cdots$, I_{M-1} into N columns S_0, S_1, $\cdots$, S_{N-1}. Hence the disjunction of a row contains all the admissible input words. If these are all the possible words, the disjunction of a row is a tautology. If there are inadmissible input words, then the hypothetical whose consequent is the disjunction of the matrix row and whose antecedent is a disjunction of all the admissible input words is a tautology. In this case we can say that the matrix row sums to a tautology relative to the admissibility conditions, or that it is a *tautology* in an extended sense of this word. (Note that this relative sense of "tautology" is relevant in minimality problems; in minimizing a switch we are not looking for a switch logically equivalent to the given one, but rather for a switch logically equivalent to the given one relative to the admissibility conditions on the inputs.) A single element j of a row i may be a tautology, in which case all other elements in the row are ϕ; this means that whenever the automaton is in state S_i it makes a direct transition to state S_j, no matter what the input is. No input

word can occur more than once in a row, else the automaton would not be deterministic.

The disjunction of the elements of a column is not in general a tautology, but cases where it is are of special interest as they are related to the concept of backward determinism.

DEFINITION 5: An abstract automaton is *backwards deterministic* if and only if for each finite sequence $I(0), I(1), \cdots, I(t), S(t+1)$, there is a unique sequence $S(0), S(1), \cdots, S(t)$ satisfying the complete table. A direct-transition matrix is backwards deterministic if and only if for each I_k and S_j there is at most one state S_i such that $I_k \frown S_i$ directly produces S_j.

We give an example of a backwards-deterministic matrix of order three.

$$\begin{bmatrix} I_0 \vee I_1 & \phi & I_2 \\ \phi & I_0 \vee I_1 \vee I_2 & \phi \\ I_2 & \phi & I_0 \vee I_1 \end{bmatrix}.$$

Another example of interest is

$$\begin{bmatrix} I_0 & I_1 & I_2 & I_3 \\ I_1 & I_2 & I_3 & I_0 \\ I_2 & I_3 & I_0 & I_1 \\ I_3 & I_0 & I_1 & I_2 \end{bmatrix}.$$

Besides being backwards deterministic, this matrix has the property that a direct transition is possible from any state to any other state. We call such a matrix directly strongly connected; see definition 6 below.

THEOREM 3: *The disjunction of every column of a direct-transition matrix is a tautology if and only if that matrix is backwards deterministic. An abstract automaton is backwards deterministic if and only if its direct-transition matrix is backwards deterministic.*

We prove first that having every column sum (logically) to a tautology is a necessary and sufficient condition for a direct-transition matrix to be backwards deterministic. If every column sums to a tautology, every input word must occur at least once in each column. But no input word could occur twice in a column, because in the $N \times N$ matrix every input word must appear exactly once in a row and there are exactly N occurrences of each input word. If an input word occurred twice in the same column, then at least one of the $(N-1)$ remaining columns must miss that word and could not be a tautology. Hence for a given state S_j at time $t+1$, and a given input word I_k at time t, there is only one state S_i which together with I_k could have directly produced S_j. Therefore, having every column sum to a tautology is a sufficient condition for a direct-transition matrix to be backwards deterministic. The proof that it is a necessary condition is obtained by reversing the considerations just used. In a backwards-deterministic matrix no input word can occur twice in the same

column. But each row contains exactly one occurrence of each input word and there are exactly N occurrences of each input word in the matrix. Hence, no input word is missing in any column, because otherwise it must occur twice or more in at least one other column. Therefore, every column must sum to a tautology.

We show next that if a direct-transition matrix is backwards deterministic, the abstract automaton is backwards deterministic. Consider a finite sequence $I(0), I(1), \cdots, I(t), S(t+1)$. It follows from the results of the preceding paragraph that there is exactly one $S(t)$ which together with $I(t)$ directly produced $S(t+1)$. Iterating this argument, we see that there is a unique sequence $S(0), S(1), \cdots, S(t)$ satisfying the complete table (for the given $I(0), \cdots, I(t), S(t+1)$). To prove the second part of the theorem in the other direction, we note that if a matrix is not backwards deterministic, there will be some I_k and some S_j such that there are two distinct states S_a and S_b, either of which will, together with I_k, directly produce S_j. Hence for the sequence $I(0) = I_k$ and $S(1) = S_j$, there are two sequences [namely, $S(0) = S_a$ and $S(0) = S_b$] satisfying the complete table.

In section 2.1 we remarked that in the presence of our deterministic assumption an infinite past would be inconvenient. The first part of theorem 3 may be used to justify this statement. In order to describe the behavior of a net over a certain period of time $t, t+1, \cdots, t+w$, we would naturally need to know the inputs $I(t), I(t+1), \cdots, I(t+w)$ and the internal state S at one of these times (or perhaps at $t+w+1$). Now if every net were backwards deterministic, it would not matter for which time S was known. But for a net which is not backwards deterministic we must know $S(t)$ to determine $S(t+1), \cdots, S(t+w)$. Hence we might as well pick a time $t = 0$ as a standard reference point for our analysis and always work forward from this time; we therefore allow t to range over the nonnegative integers only. [We could define backwards deterministic on the basis of each infinite sequence $\cdots, I(-7), \cdots, I(0), \cdots, I(t), S(t+1)$ determining an infinite sequence $\cdots, S(-7), \cdots, S(0), \cdots, S(t)$ and conduct the discussion in terms of this definition. Theorem 3 then holds with the following exception: The direct-transition matrix of a backwards-deterministic automaton may fail to be backwards deterministic with regard to states which cannot be recovered. For example, a backwards-deterministic automaton can have a transition matrix in which both $S_a \widehat{I_k}$ and $S_b \widehat{I_k}$ directly produce the same state S_j, but in which no state and input combination directly produces S_a or S_b.]

We turn now to the task of generalizing the notion of direct-transition expression to cover nondirect transitions. Consider an example. Suppose it is possible to go from state three of an automaton to state seven with either a sequence I_4, I_6, or with I_2. We could write this as $I_4 I_6 \vee I_2$, but it must be understood that juxtaposition here represents a noncommutative type of conjunction, since I_6 followed by I_4 may not carry the net from state three to state seven. We will sometimes use a special operation, called concatenated conjunction, to express the order-preserving conjunction needed here. Thus the

above may be written $I_4 \frown I_6 \vee I_2$. However, the *concatenated-conjunction* symbol, $\frown$, may be omitted if the context makes clear what is intended. The non-commutative nature of concatenated conjunction can be brought out by making the role of time explicit: $I_4 \frown I_6$ is short for $I_4(t) \cdot I_6(t + 1)$, while $I_6 \frown I_4$ is short for $I_6(t) \cdot I_4(t + 1)$. Clearly $I_4(t) \cdot I_6(t + 1)$ is not equivalent to $I_6(t) \cdot I_4(t + 1)$.

Concatenated conjunction can be used to build up transition words from direct-transition expressions. For example, we might have the transition expression $I_{3,2} \frown I_{2,5} \frown I_{5,7} \vee I_{3,7}$, or, more briefly, $I_{3,2}I_{2,5}I_{5,7} \vee I_{3,7}$. In a concrete case it might reduce to the transition expression $I_4 \frown (I_6 \vee I_9) \frown I_5 \vee (I_3 \vee I_6)$, or, more briefly, $I_4(I_6 \vee I_9)I_5 \vee (I_3 \vee I_6)$, which is of course equivalent to the transition expression $I_4 \frown I_6 \frown I_5 \vee I_4 \frown I_9 \frown I_5 \vee I_3 \vee I_6$, or, more briefly, $I_4I_6I_5 \vee I_4I_9I_5 \vee I_3 \vee I_6$. For this expansion we use a distribution principle for concatenated conjunction: $(p \vee q) \frown (r \vee s) \equiv (p \frown r \vee p \frown s \vee q \frown r \vee q \frown s)$.

We can now define the general concept of transition expression. A *transition expression* is a disjunction of concatenated conjunctions of direct-transition expressions, provided that if any concatenated conjunction contains a ϕ, it may be replaced by ϕ. Thus $I_{2,5}I_{5,3} \vee I_{2,3}$ might become $(I_3 \vee I_7) \frown \phi \vee \phi$, which would reduce to ϕ. We allow direct-transition expressions as special cases of transition expressions.

DEFINITION 6: A *transition matrix* of order N is an $N \times N$ array whose elements are transition expressions. Two transition matrices of order N can be combined by the following operations, where $\alpha(a, b)$, $\beta(a, b)$ are transition expressions for transitions from state a to state b.

Matrix disjunction: $[\alpha(a, b)] \vee [\beta(a, b)] = [\alpha(a, b) \vee \beta(a, b)]$.

Matrix concatenated conjunction: $[\alpha(a, b)] \frown [\beta(a, b)] = [\gamma(a, b)]$, where $\gamma(a, b) = \sum_{i=0}^{N-1} \alpha(a, i) \frown \beta(i, b)$, where $\sum$ represents disjunction.

Matrix (concatenated) power: $M^1 = M$

$$M^n = M^{n-1} \frown M.$$

Sum of matrix powers: $\sum_{i=1}^{n} M^i = M \vee M^2 \vee \cdots \vee M^n$.

The *characteristic matrix* $C(M)$ of a transition matrix M is obtained by replacing the elements of M with zeros or ones, according to whether the elements do or do not reduce to ϕ, i.e., according to whether transitions from state a to state b are not or are possible by M. A direct-transition matrix M is *directly strongly connected* if and only if every element of $C(M)$ is unity. *Inequality between characterizing matrices* is defined by $C(M) \leqq C(N)$ if and only if for each element $\alpha(a, b)$ of $C(M)$ and the corresponding element $\beta(a, b)$ of $C(N)$, $\alpha(a, b) \leqq \beta(a, b)$, i.e., $\alpha \supset \beta$.

THEOREM 4: *Let M be a direct-transition matrix of order n. A transition from state S_i to state S_j in exactly w steps is possible if and only if the element $\alpha(i, j)$ of $C(M^w)$ is one. A transition from S_i to S_j in w or less steps is possible if and only if the element $\alpha(i, j)$ of $C(\sum_{k=1}^{w} M^k)$ is one. A transition from state S_i to state S_j is possible if and only if* (a) *for $i \neq j$, the element $\alpha(i, j)$ of $C(\sum_{k=1}^{n-1} M^k)$ is one*; (b) *for $i = j$, the element $\alpha(i, j)$ of $C(\sum_{k=1}^{n} M^k)$ is one. A net is strongly con-*

nected if and only if every element of $C(\sum_{k=1}^{n} M^k)$ *is unity. If* α *and* β *are positive integers, then*

$$C\left(\sum_{k=1}^{\alpha} M^k\right) \leqq C\left(\sum_{k=1}^{\alpha+\beta} M^k\right)$$

$$C\left(\sum_{k=1}^{n} M^k\right) = C\left(\sum_{k=1}^{n+\alpha} M^k\right).$$

To prove this theorem, we first examine the structure of the elements of M^w, where M is the direct-transition matrix. Each element $\alpha(i, j)$ is a disjunction of concatenated conjuncts, each of the form $I_{ia_1}I_{a_1a_2}\cdots I_{a_wj}$. Clearly a transition from S_i to S_j in exactly w steps is possible if and only if at least one of these concatenated conjunctions does not reduce to ϕ, i.e., if and only if the element $\alpha(i, j)$ of $C(M^w)$ is unity. A matrix $\sum_{i=1}^{w} M^k$ has as its elements transition expressions covering transitions in 1, 2, $\cdots$, or w steps, and hence the elements of $C(\sum_{i=1}^{w} M^k)$ are one or zero according to whether a transition from S_i to S_j can or cannot be made in w or less steps. It remains for us to show that beyond a certain power (n for $i = j$, $n - 1$ for $i \neq j$), raising M to a higher power does not add to the possible transitions that can occur, but only to the way in which they occur. Consider two distinct states, S_i and S_j. There are only $n - 2$ other states to pass through. The automaton's being in one of these states S_k for more than one moment of time does not increase the possibility of getting to S_j from S_i, since whatever can be accomplished from a later occurrence of S_k can be accomplished from the first occurrence of S_k. The argument is similar for possible transitions from S_i to S_i, with the difference that here we must consider $n - 1$ other states.

4. Cycles, Nets, and Quantifiers

4.1. *Decomposing Nets*

In this section we discuss cycles in nets and their bearing on the application of logic to net analysis. As a first step we discuss the elimination of unnecessary cycles from nets.

A well-formed net (w.f.n.) may have unused switching-element input wires. This is especially likely to be the case for a coded normal form net constructed from a characterizing table, for not all bits of D and I need influence a given delay input junction. By inspection of the characterizing table of a w.f.n., we can tell which bits are irrelevant to a switch output C_i. A particular bit A_j of $\widehat{ID}$ is irrelevant to C_i if and only if for each pair $\widehat{ID}$ identical in every position A_j the value of C_i is the same. Using this criterion we can eliminate all the unused switch input wires by replacing the original switching elements with other elements which behave the same for all inputs and on which every switch input has an influence. The same process can be applied to an output table and an output switch.

It should be noted that the above process is a minimization technique, i.e., a technique for producing a simpler net which realizes the same transformations as the original net. In section 2.3 we showed how to minimize the number of delay elements by using a coded normal form. Other minimization methods are implicit in the results of preceding sections. For example, if two junctions of a net behave the same (cf. the decision procedure of section 2.2), one may be eliminated. Note that for these minimization procedures we can work from complete tables, characterizing tables, and output tables; we need not refer to the net diagrams at all.

However, our main interest at present is not in minimality in general, but in minimality only insofar as it relates to the number and nature of cycles in a net. For example, every normal form net with at least one delay element will have cycles, while the corresponding net with no irrelevant switching-element inputs may have either fewer cycles or perhaps no cycles at all. For this reason we shall hereafter consider only such nets. Our next task is to define a measure of the complexity of the cycles of a net.

A sequence of junctions $A_1, A_2, \cdots, A_n, A_1$ (possibly with repetitions) constitutes a *cycle* if and only if each A_j is an input to an element whose output is A_k, where $k \equiv j + 1$ modulo n. Thus a junction occurs in a cycle if it is possible to start at that junction, proceed forward (in the direction of the arrows) through switching elements and delay elements, and ultimately return to the junction. A junction which does not occur in a cycle has *degree* zero, as does an input-independent junction. It should be noted that this definition assigns degree zero to some junctions occurring in cycles, i.e., to all input-independent junctions which occur in cycles. The reason will become clear in the next subsection. For the same reason we require a further modification of the net before degrees are assigned to the remaining junctions of it. That modification is to replace all cycles containing both input-independent and non-input-independent junctions by cycles containing only one of these kinds of junctions. Let C be an input-independent junction occurring in a cycle with a non-input-independent junction E. Break the cycle at C by deleting the element whose output wire is joined to junction C; to make the net behave the same, we connect C to the output of a subnet which realizes the input-independent transformation originally realized by C. Such a subnet may be so constructed that it has only one cycle, and such that every junction in it is an input-independent junction (Burks and Wright [1], Theorem XIII, p. 1363). Thus, given any net N, we can find an equivalent net N' with no more cycles than N and which has no cycles containing both input-independent and non-input-independent junctions. We say that a net with no irrelevant switching-element inputs and with no cycles containing both input-independent and non-input-independent junctions is in *reduced form*. We assign degrees to all the junctions of N' (and hence derivatively to all junctions of N) as follows.

The degree of a non-input-independent junction which occurs in a cycle is the maximum number of distinct delay elements it is possible to pass through by traveling around cycles in which the junction occurs. Figure 5 shows a net

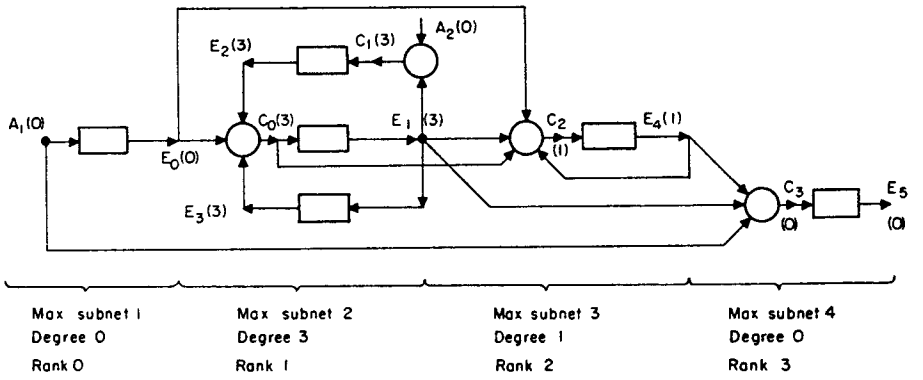

FIG. 5. Net of degree 3

with the degree of each junction in parentheses. (We stipulate that in fig. 5 the switching functions are so chosen that no junction is input-independent, and the net is in reduced form.) Note that in order to get to both E_2 and E_3 from C_0, it is necessary to pass through E_1 twice.

The *degree of a net* is the maximum of the degrees of its junctions. Figure 5 is of degree 3. A net is *entirely connected* if and only if its degree is greater than zero and the number of delay elements in it is equal to its degree. This notion should be compared with the analogous notion of "strongly connected," defined in section 2.3. We define *directly entirely connected* analogously to the notion "directly strongly connected" of section 3.3; that is, in a directly entirely connected net it is possible to start at any delay output junction and proceed forward to any delay input junction, passing only through switching elements. One of these sets of notions concerns states; the other set concerns the bits used to represent states.

Figure 5 is not entirely connected, but it may be completely decomposed into two nets of degree zero (the net $A_1 - E_0$ and the net $E_4 - E_1 - A_1 - C_3 - E_5$) and two entirely connected subnets ($E_0 - C_0 - E_1 - A_2 - C_1 - E_2 - E_3$ and $E_0 - C_0 - E_4 - C_2$).

A *maximal entirely connected subnet associated with a net junction*, say F, is the net formed of all junctions which occur in a cycle with F, together with the elements between these junctions, and the switch input junctions of all switches whose output junctions are in a cycle with F. Subnet 2 of figure 5 (the net $E_0 - C_0 - E_1 - A_2 - C_1 - E_2 - E_3$) is a maximal entirely connected subnet associated with the junctions E_2, C_1, C_0, E_1, E_3. The part of this subnet which results by deleting the delay element between E_1 and E_3 is an entirely connected subnet of figure 5, but it is not maximal.

Since any two junctions of a net either do or do not occur in the same cycle, each element of a net either belongs to a subnet of degree zero (i.e., is not in a cycle) or belongs to a unique, maximal, entirely connected subnet of the original net. (Note in this connection that "occurring in the same cycle" is a transitive relation.) A given net element may belong to several subnets of

degree zero; e.g., the delay $C_3 - E_5$ of figure 5 belongs to subnet 4 and to the subnet consisting of itself.

There are various ways to group the elements connected to junctions of degree zero into maximal subnets, of which we will give one. Let A be a junction of degree zero and B be any other junction of the net. Proceeding forward from B to A along a certain path, we can pass through n ($n \geqq 0$) maximal entirely connected subnets before arriving at A. Note that n is bounded, for if we could pass through a given maximal entirely connected subnet M and then (always proceeding forward in the direction of the arrows) later come back to M and pass through it again, it would not be the case that M is maximal. Since there are a finite number of junctions in the net and a finite number of paths from each to A, there is a maximum such number N to be associated with A. Then group into a *maximal subnet of degree zero* all the elements lying between junctions with the same maximal numbers N, together with the input wires of all switches whose output junctions are assigned the number N. Subnet 4 of figure 5 is a maximal subnet of degree zero.

It is by now clear that any net in reduced form can be uniquely and effectively decomposed into maximal entirely connected subnets and maximal subnets of degree zero, i.e., into *maximal subnets* of various degrees. Figure 5 is uniquely decomposed into the four subnets shown there. To decompose a net, one need only find the degrees of the junctions, one by one, remove all input-independent junctions which occur in cycles with non-input-independent junctions, determine the classes of junctions belonging to the same cycles, and then determine the maximal subnets of degree zero.

A *rank* can then be assigned to each maximal subnet inductively. A maximal subnet which has no net inputs or whose only inputs are net inputs is of rank 0. A maximal subnet which has at least one input from another maximal subnet of rank r and no inputs from maximal subnets of rank greater than r, is of rank $r + 1$. See figure 5 for an example of ranks. There may, of course, be several maximal subnets of the same rank. It is clear that every maximal subnet has a unique rank, for there cannot be two such subnets driving each other, else they would not be maximal (cf. Theorem IX of Burks and Wright [1]). It is worth noting that if each maximal subnet of a net is replaced by a single box with inputs and outputs, the result is a diagram without cycles. The following structure theorem summarizes these results.

THEOREM 5: *Any net in reduced form may be uniquely decomposed into (one or more) maximal subnets, each of which has a unique degree and rank.*

We conclude this subsection with a conjecture: For any degree d, there is some transformation not realized by any net of degree d. This means that there is no maximal degree such that any transformation can be realized by a net of this degree. Our grounds for making this conjecture are as follows. Consider counters with one input, designed to produce an output modulo m. When m is a power of two, one can construct a sequence of binary counters, each of degree one, each driving its successor, except the last one, which drives nothing but produces the desired output, and the whole net will be of degree 1. When

m is not a power of two, the standard way of constructing the desired counter is to take a counter modulo a power of two, sense with a switch when it reaches $m - 1$, and use that information to clear the counter back to zero. But such a feedback loop produces a net of arbitrarily high degree. Considerations of this sort lead us to believe that the conjecture is true.

4.2. *Truth Functions and Quantifiers*

We have already indicated (section 2.2) the close correspondence between switching nets (switches) and the theory of truth functions (the propositional calculus, Boolean algebra). That correspondence permits us to assign variables to switch inputs and to associate with each switch output a truth-functional expression which is a truth function of the input variables for that switch. Thus we can represent the output of a switch as an explicit function (in particular, a truth function) of its inputs.

It is natural to seek analogs for well-formed nets in general. We will give an analog for nets of degree zero and then discuss the problem for nets of arbitrary degree.

Consider first nets with delays but without cycles. For these we can express each output as an explicit function of the inputs by using the theory of truth functions enriched with the delay operator δ. Thus in figure 5 $E_0 \equiv \delta A_1$. That this can always be done for noncyclic nets can be proved from the formation rules (with rule 5 deleted, of course); the considerations involve generalizations of those connected with the concept of rank in Burks and Wright [1], p. 1361 ff.

We next mention two theorems for delay nets without cycles. The first concerns shifting a delay operator across a logical connective; for example,

$$\delta(A \ \& \ B) \equiv \delta A \ \& \ \delta B.$$

To prove this formula, we apply the definition of δ to both sides:

$$\delta[A(0) \ \& \ B(0)] \equiv \delta A(0) \ \& \ \delta B(0) \equiv 0$$

$$\delta[A(t+1) \ \& \ B(t+1)] \equiv \delta A(t+1) \ \& \ \delta B(t+1) \equiv A(t) \ \& \ B(t).$$

The legitimacy of this operation is connected to the fact that conjunction is a positive truth function (i.e., has the value zero when all its arguments are zero). In general, if P is a positive truth function, the following holds:

$$\delta P(A_1, A_2, \cdots) \equiv P(\delta A_1, \delta A_2, \cdots).$$

Both $\vee$ and $\not\equiv$ are also positive truth functions. A negative truth function is one which has the value one when all arguments are zero; $\sim$, $|$, $\downarrow$, $\equiv$, and $\supset$ are examples. To develop analogous principles for these, we need an operator δ' defined by

$$\delta' A(0) \equiv 1$$

$$\delta' A(t+1) \equiv A(t).$$

If N is a negative truth function, we have

$$\delta' N(A_1, A_2, \cdots) \equiv N(\delta A_1, \delta A_2, \cdots).$$

If the negative function is not tautologous (i.e., not true for all values of its variables), then

$$\delta N(A_1, A_2, \cdots) \equiv N(\delta^{a_1} A_1, \delta^{a_2} A_2, \cdots),$$

where each δ^{a_i} is either δ or δ'. For example, $\delta \sim A \equiv \sim\delta' A$. Note that two formulae which are the same except for the absence or presence of primes on deltas describe two functions which differ only initially; after some fixed time which is determined by the number of deltas involved they are equivalent. Shifting deltas across truth-functional connectives is equivalent to shifting all delay elements to inputs, so the resultant net consists of delays followed by a switch. This theorem can often be used to simplify expressions and nets. For example, consider a net which realizes $\delta(\delta A \not\equiv A) \not\equiv (\delta A \not\equiv A)$. Applying the theorem, we get $(\delta\delta A \not\equiv \delta A) \not\equiv (\delta A \not\equiv A)$, which by the theory of truth functions reduces to $\delta\delta A \not\equiv A$.

The second theorem concerns input-independent transformations. By a result of section 2 every such transformation is periodic, and hence is of the form $\widehat{\phi\,\alpha\,\alpha\,\alpha}\cdots$, where ϕ and α are binary words. For example, in $1010100100100\cdots$ ϕ is 1010 and α is 100. We call the length of α in bits the periodicity of the transformation, assuming that α is of minimum length. The periodicity of our example is three (not six, or nine, or etc.). The second theorem states that the class of input-independent transformations realized by cycle-free nets is equivalent to the class of periodic transformations of period one. We omit a detailed proof. The essential point in showing that every input-independent transformation realized by a cycle-free net is of period one lies in the fact that an automaton without cycles cannot remember anything for more than a fixed period of time. To show that every periodic transformation of period one can be realized by a noncyclic net, we can use part of the construction of the figure for Theorem XIII of Burks and Wright [1]. With this construction we can realize any transformation of the form $\phi 0000\cdots$. To realize a transformation of the form $\psi 1111\cdots$, we feed $\bar{\psi} 0000\cdots$ through a negation element, where $\bar{\psi}$ is the bitwise complement of ψ.

Consider next input-independent nets, i.e., nets all of whose internal junctions realize input-independent transformations. These nets may have cycles. Nevertheless, we can express the behavior of a net output as an explicit function of the inputs (in a vacuous sense) without using quantifiers. To do so it suffices to state the times at which the junctions are active. Thus, for $F(t) \equiv 111010101\cdots$, we have $F(t) \equiv [(t = 1) \vee (t \equiv 0 \bmod 2)]$. We can now let an input of a noncyclic net be driven by an input-independent junction, and by making an appropriate substitution still obtain an expression for the output as an explicit function of the inputs. Thus, given

$$C(t) \equiv A_0(t) \;\&\; \delta A_1(t),$$

we can identify A_1 with F above and obtain

$$C(t) \equiv A_0(t) \;\&\; \delta[(t = 1) \vee (t \equiv 0 \bmod 2)]$$
$$\equiv A_0(t) \;\&\; [(t = 2) \vee \{(t > 0) \;\&\; (t \equiv 1 \bmod 2)\}].$$

We can further extend our theory of truth functions to include expressions like those just used. By adding $t = a$, $t > a$, $(t - a) \equiv c \bmod b$, where t is a variable and a, b, and c are integers, we can describe any periodic function (using, of course, the truth-functional connectives). We call the theory obtained by adding these symbols and the operator δ the *extended theory of truth functions*. It is clear from the preceding discussion that the following theorem holds.

THEOREM 6: *For every junction of a net of degree zero, we can effectively construct a formula of the extended theory of truth functions which describes the behavior of the junction as an explicit function of the behavior of the inputs.*

This theorem provides the motivation for our decision in the preceding subsection to classify input-independent junctions occurring in cycles along with non-input-independent junctions not occurring in cycles, for both can be handled by our extended theory of truth functions. A much more difficult problem is to find formulae which describe the behavior of junctions of degree greater than zero as explicit functions of the net inputs. The natural place to seek such formulae is quantification theory, the next step beyond truth-function theory in the usual development of symbolic logic.

The theory of quantifiers uses, in addition to the truth-functional connectives, the quantifiers "(x)" ("all x"), "$(\exists x)$" or "$(\mathrm{E}x)$" ("some x"), etc. The functional expressions of net theory "$A(t)$", "$B(t + 3)$", etc., are clearly monadic propositional functions or predicates. An essential feature of a deterministic net is that an output $C(t)$ cannot depend on any inputs for times greater than t; hence the quantifiers used must be bounded. These bounds may be expressed by predicates such as "$x < t$" and "$x \leqq y < t$," which are basically dyadic (the second is triadic but is easily reduced to dyadic predicates). Hence the required form of quantification theory involves monadic predicates and bounded quantifiers ranging over the nonnegative integers.

Figure 6A shows a very simple cyclic net; it is described by the bounded quantifier expression

$$E(t) \equiv (\mathrm{E}x){:}x < t \cdot A(x), \tag{4.2-1}$$

which states that E is active at t if and only if A has been active at some prior

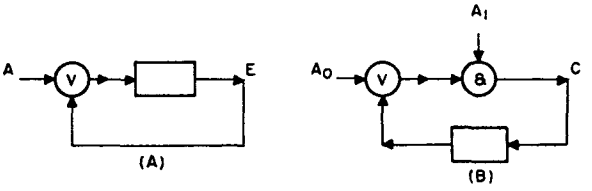

FIG. 6. Two simple nets

time. The slightly more complicated cyclic net shown in figure 6B is described by the quantifier expression

$$(4.2\text{–}2) \qquad C(t) \equiv (\mathrm{E}x)::x \leqq t \cdot A_0(x):\cdot(y):x \leqq y \leqq t \cdot \supset A_1(y),$$

which asserts that C is active at time t if and only if there is some nonlater time x at which A_0 was active and such that at that time and all later times A_1 was active. It is easy to give examples of quantifier formulae for much more complicated nets with cycles. Whether or not formulae of this type can be found for arbitrary w.f.n. is an open question.

It should be noted that in the above examples the quantifier expressions do describe the output as an explicit function of the inputs; i.e., the only function variables on the right are input variables. That is analogous to using a truth-functional expression to describe a switch output as a truth function of its inputs alone. It stands in contrast to the recursive methods for describing net behavior used previously, in which the output was expressed as a function not only of the input junctions of the net but also (in general) of the internal junctions (at an earlier time). In some cases such a recursive formulation is the natural way of specifying the behavior of a desired circuit. On the other hand, it is often simpler and more direct to specify the behavior of a net in terms of the inputs alone by means of quantifiers and simple arithmetic predicates like "is odd," "is between m and n," etc. Hence it is of interest to develop a form of quantification theory that will facilitate this method of characterizing an automaton and to find both effective (in the purely theoretical sense) and practical ways of passing from formulae in the calculus to the corresponding automaton nets and vice versa.

The problem of finding a quantifer formula for a net characterized recursively may be viewed as one of converting recursive definitions into explicit ones. As we have remarked in section 2.2, the transformation realized by each delay output of a net is primitive recursive relative to the net inputs. Theoretically one can use the well-known procedures for converting primitive recursive functions (cf. Hilbert and Bernays [17], pp. 412–421) to obtain the desired result. As it turns out, however, this method produces quantifier expressions in which some quantified variables range not over time but over the history of the states of the delay outputs. The quantifier expressions so obtained are intuitively always no more and actually less transparent than the corresponding recursive characterization.

4.3. *Nerve Nets*

We will close this paper with a few remarks about nerve nets and cycles in nets. A nerve net is a special case of a well-formed automaton net, in which each neuron consists of a positive switching element driving a delay element. Hence our general results apply to nerve nets. Not all transformations realized by well-formed nets can be realized by nerve nets.

According to theorem 2, every transformation realized by a w.f.n. can be realized by a decoded normal form net. By the results of section 4.1 the starter of a

decoded normal form net may be constructed without cycles. Hence we can construct a decoded normal form net whose cycles pass through conjunctions and delays only. Hence every transformation realized by a w.f.n. can be realized by a w.f.n. in which the only positive switches occur in cycles. A neural net is a net in which only positive switches occur in cycles. It differs from a decoded normal form net in two basic respects: First, it has no starter, and second, every switch is combined with a delay. Hence, if a starter is added to the system of nerve nets, every automaton transformation can be realized by a nerve net, except that the nerve-net output may be later in time because each neuron has a delay built into it. Usually the total time lag can be made two, because a disjunctive normal form expression, e.g., $(p \cdot \bar{q}) \vee (\bar{p} \cdot q)$, is a disjunction of conjuncts (see, for example, Kleene [11], theorem 3).

Kleene [11] has investigated the logic of nerve nets in some detail. He analyzes nets in terms of the kinds of events (input histories) they can detect, and he establishes the result that an event can be detected by a net if and only if the event is regular (theorems 3 and 5). The reader is referred to page 22 of Kleene [11] for a definition of "regular"; we note here merely that an important ingredient of the notion of regularity is periodicity. For example, an input of the form $\alpha \frown \alpha \frown \cdots \alpha \frown \phi$, with an indefinite number of α's, is regular. It is easy to construct a net which will be active at time t if and only if the history of its input is of the form $\alpha \frown \alpha \frown \cdots \frown \alpha \frown \phi$, for an indefinite number of α's; cf. the discussion of section 4.2 on periodic transformations.

The pervasiveness and importance of cycles in the analysis of automata and nerve nets are worth emphasizing. When cycles are permitted in automata nets, these nets become much more powerful, and, correspondingly, the logic required to treat them becomes much more complicated. There are many ways in which nets can involve cycles. We have just noted that by Kleene's results an important aspect of any input history which can be detected or distinguished by automata is the periodicity ingredient in its regularity. By our results of the previous subsection the internal structure of an automaton is analyzable into cycles; and by earlier results (see section 2.2) any output which is independent of the inputs is periodic, and hence cyclic in character. The relations between these various cyclic aspects of automata remain to be investigated. It would be of interest to have a theory which shows how they are interconnected.