Topics in the Theory of One-Dimensional Iterative Networks*

WILLIAM L. KILMER

Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Massachusetts

I. INTRODUCTION AND SUMMARY

This paper is an independent sequel to Kilmer (1963) on general dynamic behavior in one-dimensional iterative logic networks (Fig. 2). It stems from Hennie's (1961) question,¹ which is whether or not every system of one-dimensional iterative logic networks (i.e., family of n-celled networks, n = 1, 2, 3, ...) can have its typical cell decomposed into two partially separate subcells, one subcell passing information only from left to right and the other subcell passing information only from right to left, without essentially reducing the computing capacity of the over-all system.

We prove in this paper that many one-dimensional iterative logic systems have cell types that cannot be equivalently decomposed into partially separate left-to-right and right-to-left subcells, almost regardless of the slightness of subcell separation and/or the weakness of system equivalence required. Our proof involves a slight development of the material in Davis (1958, Sec. 6.2), and a synthesis of an iterative network to calculate successions of Post normal system productions.

We note that the above-mentioned nondecomposition result is opposite to the one obtained by Henrie for systems whose memoryless cell designs are such as to prevent any corresponding n-celled network from ever exhibiting over-all memory in an equilibrium state.² He showed

* This research was supported by Air Force Cambridge Research Laboratories Contract AF 19(604)-6619, under the auspices of the Montana State College Electronics Research Laboratory, Bozeman, Montana. The author is now at the Research Laboratory of Electronics, Massachusetts Institute of Technology, on leave from Montana State College.

¹ Hennie (1961), Sec. 5.3, p. 94, and Sec. 9.1, pp. 170-174.
that in such cases, completely separate left-to-right and right-to-left canonical cell decompositions are always possible, so that the resulting system always produces the same equilibrium cell outputs as the original. The significance of our result is that the complement of the class of Hennie's decomposable systems is precisely the class to which our non-decomposition theorem pertains.

The paper also summarizes several new results on the dynamic behavior of iterative networks of Hennie's decomposable type. We note that if such networks are not decomposed into their Hennie canonical equivalents, they can easily enter a dynamic switching cycle (oscillation) after being perturbed from equilibrium by any single external input change. We also note that all general cycling, bounded transient, and boundary transient problems for such networks are recursively unsolvable, just as was shown previously for network types not decomposable by Hennie. We outline a proof of these results which depends only on a direct correspondence between the dynamic behavior of one of Hennie's decomposable iterative networks, and Kahr, Moore, and Wang's well-known domino construction for the entscheidungsproblem.

II. PART ONE: GENERAL NONDECOMPOSITION THEOREM

A. INTRODUCTION AND DEFINITIONS

Part One concerns computational equivalences between given one-dimensional iterative networks and various possible decompositions of these networks. We need the following definitions:

1. The building blocks of our iterative networks are finite automata, $A$, (i.e., cells) as shown in Fig. 1. All of the variables listed there have finite domains of values, and associate with the following Cartesian functions on their product spaces:

\[
\begin{align*}
    f_\alpha & : \alpha \times \gamma \times \sigma \to \alpha' \\
    f_\beta & : \beta \times \gamma \times \sigma \to \beta' \\
    f_\delta & : \alpha \times \beta \times \gamma \times \sigma \to \delta' \\
    f_\sigma & : \alpha \times \beta \times \gamma \times \sigma \to \sigma
\end{align*}
\]

realized with zero time delay after each $\alpha \times \beta \times \gamma \times \sigma$ change

new $\delta$ following each $\alpha \times \beta \times \gamma \times \sigma$ change realized with unit time delay

It is assumed that all variable value changes occur only at unit time instants. Note that the $\alpha$ and $\beta$ variables interact sequentially through $\sigma$.

\[^3\text{Cf. Kilmer (1963).}\]

\[^4\text{Cf. Kilmer (1963).}\]

\[^5\text{Cf. Kilmer (1963).}\]
Fig. 1. Finite automaton building block of iterative network. $\alpha$ is the memory state variable. $\alpha$ domain of values = $\alpha'$ domain of values. $\beta$ domain of values = $\beta'$ domain of values.

\[
\begin{align*}
\beta_0, \Delta_n, \gamma_{n+1} = \beta_0, \delta_0, \delta_1, \ldots, \delta_n, \gamma_{n+1} \text{ IS OVERALL NETWORK OUTPUT}
\end{align*}
\]

Fig. 2. An $n$-automaton, one-dimensional iterative network corresponding to $A$. All finite automata $A_i$ are identical.

2. To each $A$, we correspond an $n$-automaton, one-dimensional iterative network, i.e., an $nA$-net, as shown in Fig. 2. Each $A_i$th automaton there is an exact copy of $A$. We denote the ordered set of automata inputs $\gamma_1, \gamma_2, \ldots, \gamma_n$ as $\Gamma_n$. A system, $\Sigma_A$, is the entire set of $nA$-nets, for all finite $n$.

3. Now we define an $nA$-net computation. At $t = 0$ we set all of our initial conditions into the net, and fix $\alpha_1, \Gamma_n, \beta_n$ to remain constant.
from then on. We assume that the \( \sigma_i \) change times are all the same. (The generality of this scheme is clear from Hennie (1961, Sec. 5.3).) We also assume throughout Part One that our net runs through a sequence of variable value changes from \( t = 0 \) to \( t = T < \infty \), and then remains in equilibrium from \( T \) on. For each \( n \), we designate the result of an \( nA \)-net computation as the equilibrium \( \delta_{A(n)} \) value out of the \( A(n) \)th automaton of the \( nA \)-net from \( T \) on. The function \( A(n) \) can be chosen arbitrarily, and applies to every \( nA \)-net in \( \Sigma_A \).

4. Next, we define a special type of \( \Sigma_A \) which affords us a framework within which to discuss proposed left-to-right and right-to-left decompositions of given \( \Sigma_A \). Figure 3 shows the scheme. The automaton

\[ \text{FIG. 3. Restricted finite automaton building block of an } \alpha\beta \Sigma_A. \text{ At least one solid arrowhead must be missing, indicating at least a partial separation of the } \alpha \text{ and } \beta \text{ parts in every corresponding or A-net. } \sigma_\alpha, \sigma_\delta, \sigma_\beta \text{ are memory state variables. } \alpha \text{ domain of values } = \alpha' \text{ domain of values. } \beta \text{ domain of values } = \beta' \text{ domain of values.} \]
there operates in finite variable-value domains according to the following rule: At most, no more functional dependencies are allowed than are shown explicitly below:

\[ f_a : \alpha \times \gamma \times \sigma_a \rightarrow \alpha' \] realized with zero time delay after each \( \alpha \times \beta \times \gamma \times \sigma_a \times \sigma_b \times \sigma_3 \) change

\[ f_\beta : \beta \times \gamma \times \sigma_\beta \rightarrow \beta' \] \( \sigma_a \times \sigma_\beta \times \sigma_3 \) change

\[ f_\delta : \alpha \times \beta \times \gamma \times \sigma_a \times \sigma_\beta \times \sigma_3 \rightarrow \delta \] new values following each \( \alpha \times \beta \times \gamma \times \sigma_a \times \sigma_\beta \times \sigma_3 \) change realized with unit time delay

(or symmetrically with respect to \( \alpha \) and \( \beta \)).

These restricted dependencies are reflected in Fig. 3 by requiring that at least one solid arrowhead shown there be missing (as noted in the figure).

We define an \( a_\beta \Sigma_A \) as a \( \Sigma_A \) with Fig. 3-type A.

5. Finally, we define a weak computational equivalence between a \( \Sigma_A \) and an \( a_\beta \Sigma_A \). A \( \Sigma_A \) is equivalent to corresponding \( a\beta \Sigma_A \) if and only if there exist functions \( A'(n) \) and \( A''(n) \) such that the result of every \( n \Sigma_A \) computation is the same as that of its corresponding \( n \Sigma_A'' \) computation. We assume here that the \( \alpha \Sigma_A \) over-all inputs to both \( n \Sigma_A \) are always identical, and that the remaining parts of each initial condition specification are at least in one-to-one correspondence. We emphasize that the equivalence must hold over all possible initial condition specifications for each corresponding pair of networks in \( \Sigma_A \) and \( a\beta \Sigma_A \).

B. Theorem and Proof

We now state our general nondecomposition theorem.

Theorem. Not every \( \Sigma_A \) has an equivalent \( a\beta \Sigma_A \).

Before beginning the proof, we make a precautionary remark. Our theorem, as stated, has a subtle aspect: Its proof must show that no bizarre \( a\beta \Sigma_A \) coding scheme could ever exist in some \( \Sigma_A \) case that might in effect enable an \( a\beta \Sigma_A \) to compute the same thing as \( \Sigma_A \), in the sense of computation which we have chosen.

One further remark: The equivalence relation on which we have chosen to prove our theorem is the weakest natural one for which the theorem holds. Two stronger natural alternatives are covered by lemmas in section (v) of the theorem proof, and in the Appendix it is argued that the two other most natural alternatives, while trivially handled, only
serve to avoid the real problem. The interested reader is referred to the Appendix for further details.

Proof: We prove our theorem in five sections. The first four develop somewhat separate topics, and the fifth combines them to prove the theorem.

(i) In this section we summarize Arbib (1962), since his paper is not easily accessible. A monogenic Post normal system consists of: (1) a finite alphabet of letters; (2) one axiom, consisting of a finite string of letters from the alphabet; and (3) a finite number of production rules of the form \( a_i b \rightarrow b c_i \), where each \( a_i, c_i \) pair is a particular pair of finite strings of letters from the alphabet, and \( b \) is an arbitrary nonempty finite string of letters from the alphabet. At most, one production rule can apply at any stage of the production process, and if at any stage no rule applies, the process terminates.

In the terminology of Davis (1958, Chap. I), Arbib showed that the successive instantaneous descriptions of any Turing machine computation can be exactly reproduced, in the right order, as a result of productions in a monogenic Post normal system. The two Post strings corresponding to any two successive Turing machine descriptions, though,
Fig. 5. Schematic diagram for the inverse tree of productions corresponding to the Turing machines discussed in (ii) of the text. Horizontal lines represent instantaneous Turing machine descriptions. Directed arrows lead from each such description to all of its predecessors.

have sandwiched between them almost all “rotations” of the second description. Figure 4 outlines this relationship schematically.

(ii) After Davis (1958, Chap. 6.2), we now consider only Turing machines whose defining quadruple sets are enlarged so that every computation terminates at the special instantaneous description $h_q'h$. The inverses of the quadruples (i.e., productions) defining any such Turing machine yield a nonmonogenic combinatorial system. That is, instead of defining a unique succession of instantaneous Turing machine descriptions as shown at the left in Fig. 4, they define a predecessor tree (usually infinite) of instantaneous Turing machine descriptions as shown in Fig. 5. In other words, they span the set of all possible computations for the given Turing machine by working backwards along all possible paths from the Turing machine’s common computational ending, $h_q'h$. The inverse system is nonmonogenic because in general

6 That is, terminating sequence of instantaneous descriptions.

7 That is, the tape is blank except for the adjacent symbols $hh$, and the right $h$ is scanned by the reading head, which is in terminal state $q'$.

8 We use the word “tree” here to indicate that there is only one sequence of inverse productions that leads to any particular predecessor string.
Fig. 6. A schematic representation of Arbib's results as modified to apply to the inverse tree of productions of a Turing machine. The heavy horizontal lines represent the exact same strings as in Fig. 5. The light horizontal lines represent successive rotations of the first heavy line string below them.

Each instantaneous Turing machine description can be preceded by several different instantaneous descriptions (bounded by the number of Turing machine quadruples).

Now the fortunate thing is that Arbib's results, summarized in (i) above, are so developed as to make obvious the following corollary: The instantaneous descriptions in the inverse tree of productions corresponding to any Turing machine can be exactly reproduced, in the right relative order, as a result of productions in a (nonmonogenic) Post normal system. Just as before, though, the two Post strings corresponding to any Turing machine description and its predecessor in such a tree have sandwiched between them in the Post system almost all rotations of the predecessor. Figure 6 outlines this relationship.

(iii) Next consider Fig. 5-type trees for universal Turing machines that have been modified to end all of their computations at \( hq'h \). We recall first that each such Turing machine has unboundedly many computations consisting of more than any fixed, finite number \( k \) of successive instantaneous descriptions. Actually, Arbib used quintuples to define his Turing machines, whereas Davis uses quadruples, but this slight mismatch is trivially reconciled.

10 The truth of this is immediately evident, for if it were false the domain of some universal Turing machine would be a recursive set, which is impossible.

11 Davis (1958, terminology).
that occur below any fixed $k$th level of productions (cf. Fig. 5). We also note that for any other fixed integer, $l$, there is always some $(k+j)$th level, $j > 0$, which contains more than $l$ distinct production strings.\textsuperscript{12} We call this pair of facts about $k$ and $l$ the $(k, l)$-relation. It furnishes us with the central items that we need to prove our theorem.

(iv) In this section we show how a cascade of abbreviated Fig. 3-type automata can represent a particular, single-axiom, nonmonogenic-Post normal production process. Consider a cascade of just $\alpha$ parts of Fig. 3-type automata as shown in Fig. 7. Suppose that the axiom string of the Post system to be represented there is fed sequentially, one letter per time unit, into $\sigma_{\alpha_1}$ over its $\alpha_1$ input line. Then $\sigma_{\alpha_1}$ feeds sequentially out, at the same rate, over its $\alpha_2$ output line the string that represents the result of the first Post production on the axiom. Now in general several $a_i b \rightarrow b c_i$ production rules apply to the axiom, so (in Fig. 7) the value of $\pi_1$ specifies which one.

In order to implement this, $\sigma_{\alpha_1}$ must first sense, under control of $\pi_1$, the unique $a_i$ prefix of its $a_i b$ input string that specifies the production it is to represent. After that, $\sigma_{\alpha_1}$ must pass from $\alpha_1$ to $\alpha_2$ the $b$ part of its input string, and then append the proper $c_i$ string. In this way, $\sigma_{\alpha_1}$ produces its total $b c_i$ output to $\alpha_2$. The $\sigma_{\alpha_1}$ block is finite, because it need at most remember a finite number of finite $a_i$, $c_i$ string pair subsets, where each subset corresponds to a separate $\pi_1$ value, plus the $i$ index of a recognized $a_i$ string.

Similarly, if each $\sigma_{\alpha_i}$ in Fig. 7 is identical to $\sigma_{\alpha_1}$, it effects a sequential input-output Post transformation, $a_i b \rightarrow b c_i$, in the same manner. If at any stage the input sequence to some $\sigma_{\alpha_j}$ is such that no production rule applies to it, $\sigma_{\alpha_j}$'s output remains constant at the null value. Thus our Fig. 7 network represents a particular $\pi_1$, $\pi_2$, $\cdots$, $\pi_n$-specified, Post production process if fed the right axiom over $\alpha_1$.

\textsuperscript{12} The truth of this fairly obvious fact is easily demonstrated as follows: Program onto a universal Turing machine the Turing machine that computes in some standard fashion whether any number put onto its tape is a prime, and if it is, prints out that number as its result, and 0 otherwise.
In this section we conclude the proof of our theorem. The idea is to construct a $\Sigma_{A}$ such that any alleged $\alpha \beta \Sigma_{A'}$ equivalent to it can be shown in fact to be inequivalent at sufficiently large $n$ (i.e., network size).

Figure 8 shows the operationally significant aspects of one part of our proof $\Sigma_{A}$. The complete automaton $A$ for the $\eta A$-net, of which Fig. 8 is but the operational remains, is shown as $A_1$ in the figure. We assume there that the $\alpha_1$ value into $A_1$ is a special left boundary constant, and that $\alpha_1$ out of $A_1$ carries the sequence $hq'qh$ during the first 3 time units after $t = 0$, and null after that. Now notice that the $\alpha$ portion of the network in Fig. 8 is the same as the network in Fig. 7 minus its $\pi_i$ inputs. Therefore by assuming that the $\pi_i$ values are supplied internally to the $\sigma_\alpha_i$ blocks in Fig. 8, we can also assume that the $\alpha$ part of Fig. 8 represents a Post production process in the exact same manner as explained for Fig. 7. We also assume that the particular succession of Post productions represented in Fig. 8 is one of those infinitely many alluded to schematically in Fig. 6. We assume, further, that the Turing machine in question in Fig. 6 is universal.

Now what we want in Fig. 8 is each stored $\pi_i$ value to be the same as $\gamma_{n-i+2}$. In this way the decision at the $\sigma_{\alpha_i}$ block, corresponding to that at the "first-choice level" in Fig. 6, is specified by $\gamma_n$ etc. This is easily arranged by sending at successive unit time instants, $\gamma_n$, $\gamma_{n-1}$, $\cdots$ $\gamma_2$, all the way down along the $\beta$ line to $A_1$ and back along the $\alpha$ line to $\sigma_{\alpha_2}$, $\sigma_{\alpha_3}$, $\cdots$, $\sigma_{\alpha_n}$, respectively. This enables each $\gamma_{n-i+2}$ value to arrive...
at $\sigma_\alpha_\gamma$ at or before the time when it must play its $\pi_\lambda$ role (i.e., at $t = i - 2$, for $2 \leq i \leq n$). The logic circuitry required for this is trivial.

Suppose now we specify that Fig. 8's $\delta_j$ equilibrium value, for all $j$, be a parallel display of the recognized $a_\gamma$ prefix on the $a_\beta b$ Post sequence received over $a_\alpha_j$ by the $j$th automaton, provided that such a sequence is received at all, and null otherwise. We note that since our network employs Arbib's scheme, modified as described in connection with Fig. 6, such equilibrium $\delta_j$ exhibit, as $j$ grows, every successive reading head and adjacent tape situation that occurs during the course of the inverse Turing machine computation that our network represents. Thus our equilibrium $\delta_j$ outputs exhibit the full essence of that computation. The logic circuitry required for this is easily arranged because each $j$th automaton can receive at most only one finite, unbroken string of non-null values (i.e., Post-string letter representations).

Now let us review the essential features of our completed Fig. 8 network. If there were no $\beta$-to-$\alpha$ communication there at all, the cascade of $\sigma_\alpha_\gamma$ blocks would have to generate the entire tree of Fig. 6 strings for $[n/2] - 2$ productions out before they could know anything about which possible $a_{\gamma_{[n/2]}}$ string the $\gamma_\alpha_\lambda$, $\gamma_{\alpha_{n-1}}$, $\ldots$, $\gamma_{[n/2]-1}$ values specified. But by the $(k, l)$-relation of section (iii), no matter how complex the $\sigma_\alpha_\gamma$ blocks were, there would be some $[n/2] - 2 = m$ that would require $\sigma_{\alpha_{m+1}}$ to carry more distinct strings to $\sigma_{\alpha_{m+1}}$ than there were $\sigma_{\alpha_{m}}$ block states, which is clearly impossible. Therefore we conclude the following lemma.

**Lemma 1.** If in the theorem, we constrain $\alpha_\beta \Sigma_\lambda_\alpha$ to be a system of networks with no $\alpha$-to-$\beta$ communication or vice versa, and we strengthen the equivalence relation by allowing $A'(n)$ to be chosen as $n$, the system of completed Fig. 8 networks is adequate to prove the weakened theorem.

Now let us continue our construction by forming a composite $nA$-net consisting of one completed Fig. 8 network alongside another, the latter turned upside down and end for end as outlined in Fig. 9. The $\gamma_\alpha$ input to the Fig. 8 portion of the network there is the $\gamma_{\alpha_{n-1}}$ input to the upside-down, end-around Fig. 8 portion of that network. We assume that each composite automaton's equilibrium $\delta_j$ output in Fig. 9 consists of the cross product of the $\delta_j$ outputs of its two Fig. 8 subautomata, and that otherwise the two Fig. 8 portions of Fig. 9 compute separately, as before, from their respective $\gamma_\alpha$ inputs. Thus by considering $A(n)$ equal to 1 and $n$ in Fig. 9, we get the following corollary to Lemma 1.

$[a/b]$ is the largest integer less than or equal to $a/b$. 
Fig. 9. General $\Sigma_A$ network used in nondecomposition theorem proof

**Lemma 2.** If, in the theorem, we strengthen the equivalence relation by allowing $A'(n)$ to be chosen arbitrarily, the system of Fig. 9 networks is adequate to prove the weakened theorem.

We now complete our proof construction for the theorem itself. We simply modify the Fig. 9 network so that upon completion of its computation as described above, a signal is sent into the Fig. 8 portion of its composite first automaton, $A_1$, to generate the sequence $hq'q$ for the second time. The entire Fig. 8 portion of the modified Fig. 9 network then recomputes a new Post production succession according to some non-trivial interpretation of the following new set of Fig. 8-portion $\pi_i$ values: new $\pi_i = \text{Fig. 9-computed composite equilibrium } \delta_i$, $2 \leq i \leq n$.

We denote the Fig. 8-portion $A_i$ equilibrium outputs corresponding to this computation, $\delta_i^*$. Finally, each of these $\delta_i^*$ values is regarded as a $\pi_i$ value for specifying still another Post production succession computation, this time carried out in the upside-down, end-around Fig. 8 portion of the modified Fig. 9 network. The method for this computation is essentially the same as that for computing the $\delta_i^*$ values from the Fig. 9 equilibrium $\delta_i$. We denote the upside-down, end-around Fig. 8 portion $A_j$ equilibrium outputs corresponding to this last computation $\delta_j^{**}$. Also we assume that each automaton's equilibrium output for the resulting network is the cross product of its Fig. 9 equilibrium output, $\delta_j^*$, and $\delta_j^{**}$. 
Now the idea of our construction is as follows: Since our Fig. 9 system, $\Sigma_A$, is adequate for proving Lemma 2, there must always be an $A(n)$th Fig. 9 equilibrium output for which any alleged $a_\beta \Sigma_A$ equivalent could not guarantee an equal $A'(n)$th equilibrium output. With this in mind, a $\delta_i^*$ computation is swept dependently past that old Fig. 9 equilibrium $\delta_A(n)$ value to put all of the new $\delta_j^*$ equilibrium outputs to the right of $\delta_A(n)$ into the same Lemma 2 nonequivalence category as $\delta_A(n)$. Next, a similar computation effects the same nonequivalence recategorization of all of the new $\delta_j^{**}$ equilibrium outputs to the left of $\delta_A(n)$. The result is that none of the $nA$-net total equilibrium outputs can have a corresponding $nA'$-net equilibrium $A_j$ output that is guaranteed to be equal for every possible initial condition specification. This proves our theorem.

III. PART TWO: ON DYNAMIC BEHAVIOR IN HENNIE'S DECOMPOSABLE NETWORKS

A. INTRODUCTION, DEFINITIONS AND RESULTS

In Part Two we consider iterative networks of the type shown in Fig. 10. We assume that all of their finite automata, or cells, are identical just as in Part One; but contrariwise, we assume now that our cells do not have any $\sigma$ memory states, and that there is a unit delay between every cellular $\alpha_i \times \beta_i \times \gamma_i$ input change and every corresponding $\alpha_{i+1} \times \beta_{i-1} \times \delta_i$ output adjustment to that change. We assume that all network cells operate in time synchrony, but remove our Part One restriction that all networks must always compute from their initial conditions on through to equilibrium states. If, in fact, they do, we call the intervening succession of signal changes a transient, but if they do not, we say that they enter a cycle (i.e., oscillation). We define an ordered pair of $\alpha_i, \beta_{i-1}$ values at any given time in a network as the $i$th lateral state.
We say that if corresponding to each $\alpha_1 \Gamma \beta_0$ total input there is one and only one total lateral state and $\delta$ output value configuration for which the over-all network is in equilibrium (i.e., no signal values tending to change), the network is Hennie-decomposable.

In Kilmer (1963), three central dynamic problems were shown to be recursively unsolvable for the class of one-dimensional iterative logic networks that is just the complement of the class of Hennie-decomposable ones. We proceed now to outline a proof that the same three problems are also recursively unsolvable in the class of Hennie-decomposable, but not Hennie-decomposed, networks. They are easily solvable in the class of Hennie-decomposed networks (cf. Kilmer (1961)).

Let us denote our Fig. 10 Hennie-decomposable network $N^*$. We call an $N^*$ a transient $N^*$ if, when starting in equilibrium at $t = -1$ and subjected to a single $\gamma_i$ value change at $t = 0$, it enters a transient instead of a cycle. In case the cell design of a transient $N^*$ is such as to insure that all single $\gamma_i$ changes from equilibrium cause transients involving lateral state changes all the way out to one or both boundaries of every corresponding $N^*$, we call the $N^*$ cell design boundary transient. And in case a transient $N^*$ cell design is such as to insure that no single $\gamma_i$ change from equilibrium can cause transients involving lateral state changes more than a bounded (hence calculable) number of cells to the right and/or left of the $\gamma_i$ change in any corresponding $N^*$, we call the cell design bounded transient.

Our results in Part Two are all stated in one theorem.

**Theorem.** (1) There does not exist a recursive procedure to determine of an arbitrary transient $N^*$ cell design whether or not it is either bounded or boundary transient. (2) There does not exist a recursive procedure to determine of an arbitrary $N^*$ cell design whether or not any corresponding $N^*$ can ever enter a cycle after being disturbed from equilibrium by only one $\gamma_i$ change.

**B. A Proof Outline of the Theorem**

Our purpose in this section is merely to sketch a rough proof outline for our theorem, since we feel that once our point of view has been exposed, the proof scheme will be obvious, and to add further detail would just detract from the paper.

Our general idea is the following: We construct a general $n$-celled Hennie-decomposable network whose dynamic response to a single $\gamma_i$ perturbation from equilibrium directly simulates the succession of
instantaneous descriptions that define a given Turing machine's operation. Then if the Turing machine halts, the corresponding dynamic response is to define either a cycling or bounded transient system, the choice depending upon the network's cell design. On the other hand, if the Turing machine does not halt, the corresponding dynamic response is to define a boundary transient system. In this way, because of the unsolvability of the halting problem for Turing machines, we prove our theorem.

The whole trick is to find an appropriate way to properly embed a given Turing machine operation into the dynamic behavior of a Hennie-decomposable network. The obvious approach is to let the \( i \)th lateral state of the network represent the symbol printed on the \( i \)th tape square of a corresponding (potentially semi-infinite tape) Turing machine, and then "activate" only that lateral state whose corresponding Turing machine square is being scanned by the reading head. The trouble with this approach, though, and variations of it, is that it leads to networks whose transient responses are determined by initial conditions, and which for that reason are not Hennie-decomposable.
On the other hand, Kahr, Moore, Wang's (1962) domino construction for the entscheidungsproblem suggests an embedding approach that meets all proof needs. A trivially modified version of their construction is indicated sketchily in Fig. 11. Each square there is a domino containing a separate symbol (or color) on each edge, and the only rule governing its placement is that the meeting edges between any adjacent pair of dominos must have the same symbols. Kahr et al. chose their domino types to be such that every $k$th diagonal of dominos gives a direct representation, from bottom left towards top right, of the $(k - 1)$th instantaneous description in the operation of the Turing machine from which the domino details were specified. Thus in Fig. 11, the successive $S_{1i}$ are the leftmost symbols on the tape of the successive $i$th corresponding instantaneous Turing machine descriptions.

In our theorem proof, we let the lateral states of a general Hennie-decomposable network represent particular domino designations within corresponding columns of dominos as suggested at the bottom of Fig. 11. More explicitly, we let the $i$th lateral state of our network be $D$ in equilibrium if and only if: (1) $\gamma_i = 0$, for all $j < i$, or else (2) the Turing machine computation represented by the network halts at a corresponding lateral state well to the left of the right end of the network, in which case the $i$th equilibrium lateral state equals $D$ for all $i$. This second case is clarified below. In case (1), the equilibrium lateral states to the right of the leftmost $\gamma_i = 1$ value are $D'$ for the $(i + 1)$th lateral state, and $S_{1i}$ for each $(j + 1)$th lateral state thereafter.

As for the dynamic behavior of our network, suppose it is in the condition described by case (1) above, and that $\gamma_i$ is the leftmost $\gamma$ to have a 1 value in the network at $t = -1$. Suppose also that the network is in equilibrium at $t = -1$. Suppose, further, that at $t = 0$, $\gamma_i$ changes from 0 to 1. If $i > j$, nothing happens. But if $i < j$, every $k$th lateral state of the network, for all $k > i$, is first reset to state $D$, one by one and in succession from left to right. One time unit after this resetting process begins, the following more complicated succession of left-to-right waves of lateral state changes is started. If we let each $T_k$ duration be 4 time units, from $t = 2$ to $t = 5$ the lateral state $i + 1$ changes from $D$ to $D'$ as shown by the $T_1$ arrow above the $i$th cell in Fig. 11. From $t = 6$ to $t = 9$, the lateral state $i + 2$ changes its domino specification as shown by the $T_2$ arrow in Fig. 11. From $t = 10$ to $t = 13$, the lateral states $i + 2$ and $i + 3$ change their domino specifications as shown by the $T_3$ arrows in Fig. 11, etc. for $T_4$, $T_5$, $T_6$, \ldots. This process con-
continues, enabling the lateral states to reach their equilibrium $S_{l_t}$ in a left-to-right manner, at least until a point is reached where the corresponding Turing machine halts.

If such a point is ever reached, our proof network is designed so that the appearance of the lateral halt state either: (1) quenches all of the rest of the network's dynamic activity, and thus defines a bounded-transient system; or else (2) causes an adjacent pair of cells to enter a contradictory switching cycle, and thus defines a cycling system (at least for greater than critical network sizes). In case (1), total quenching is possible because the $T_i$ switches travel at only one fourth the maximum rate of speed along the network.

If in contrast to the previous paragraph, a Turing machine halting place is never reached in the network, the ongoing dynamic activity described above proceeds all the way to the right boundary of the network in every case, and thus defines a boundary-transient system.

The foregoing general dynamics discussion for case (2)-type equilibrium specifications at $t = -1$ follows in almost exactly the same way as just described for case (1)-type equilibrium specifications at $t = -1$. That is, after each perturbation from case (2) equilibrium, the same computation is performed, starting at the perturbation point, as described for case (1).

There is one crucial detail that was omitted in all of the discussions above, and that is covered schematically in Fig. 12. Figure 12 indicates the relative time sequences of network lateral state changes involved in center, right, and left move representations of the corresponding Turing machine reading head. By tracing through the switching sequences there, the reader can see that every time one domino directly influences the specification of another in the next instantaneous domino description, the corresponding time sequence of network lateral state changes is such as to allow a deterministic transfer of this influence between corresponding network lateral states. The "a" arrow labels in Fig. 12 are short notation for $T_i$; the "b" labels, for $T_{i+1}$; the "c" labels, for $T_{i+2}$; and the "d" labels, for $T_{i+3}$. For the most part, it is the details of the "move right" which require the $T_i$ durations to be 4 time units.

C. Sufficiency Conditions for Cycle Avoidance

We report here without elaboration that several distinct sets of conditions have been found for $N^*$ that insure that they can never enter a cycle after being perturbed from equilibrium by a single $\gamma_i$ change.
Fig. 12. Schematic diagram of the relative time sequence of lateral-state changes involved in our network representations of the three types of reading-head moves.

These conditions were developed using the approach taken in connection with Lemma 1 of Kilmer (1962A). They are important because they demonstrate the existence of tractable ways of controlling dynamic behavior in general $N^*$.

APPENDIX. ON THE NONDECOMPOSITION THEOREM

EQUIVALENCE RELATION

1. Suppose the equivalence relation of our nondecomposition theorem were weakened by requiring for each $nA$-net of $\Sigma_A$ only a $g(n)A'$-net of $\alpha\beta\Sigma_{A'}$, for arbitrary $g(n)$, and otherwise the same equivalence as before. Then there would be no theorem, by Kilmer (1963), and Arbib (1962), for a $g(n)A'$-net could always be devised that fed the $nA$-net's $\alpha_i\Gamma_n\beta_n$ and initial condition values sequentially into its $(n + 1)$th automaton. Then the $g(n)A'$-net could compute in Turing machine fashion from left to right, starting at its $(n + 1)$th automaton and finishing at its $g(n)$th automaton, the same recursive function\textsuperscript{14} that the

\textsuperscript{14} Recall our assumption in Section II, A, 3.
\( nA \)-net computed. Every Turing machine computation can be directly represented by a Post normal system, and every Post normal system can be directly represented by a cascade of identical finite automata, each receiving a string of signal values, \( a_i b_i \), sequentially from its left neighbor and feeding a consequent string of signal values, \( b_{c_i} \), sequentially into its right neighbor.

2. Suppose the equivalence relation of the theorem were strengthened by requiring that all of the equilibria \( \delta_i \) from each \( nA \)-net of \( \Sigma_A \) be respectively the same as all of the equilibria \( \delta_i \) out of the corresponding \( nA' \)-net of \( \alpha\beta\Sigma_{A'} \), and otherwise the same equivalence as before. As far as basic computing capacity is concerned, this equivalence relation is essentially covered in Lemma 2. For every difference between the equivalence there and the one here can be overcome by a simple rewiring of network outputs.

3. Suppose, finally, that the equivalence relation were strengthened by requiring: (1) the Lemma 2 equivalence (which is shown to be essentially the same as the theorem equivalence at the end of the theorem proof); and (2) also that the sequential \( A(n) \)th automaton's output from the \( nA \)-net of \( \Sigma_A \) be the same as that out of the \( A'(n) \)th automaton from the \( nA' \)-net of \( \alpha\beta\Sigma_{A'} \) from \( t = 0 \) on. In this case the corresponding theorem is easily proved by constructing a \( \Sigma_A \) that always puts the sequence \( (n \) zeros, followed by a 1, followed by \( n \) zeros, followed by a 1, \( \ldots \) \) out of the rightmost automaton of each of its \( nA \)-nets, starting at \( t = 0 \). Our basic question, though, is not how various \( \Sigma_A \) systems compute, but what, in principle, they can compute. The present sequential equivalence relation is wholly irrelevant to that.

Acknowledgment

The author wishes to acknowledge several valuable discussions with Michael A. Arbib and Warren S. McCulloch concerning the over-all body of work reported here, and complete collaboration with Dr. McCulloch on Part Two, Section C.

Received: June 6, 1963.

References


---

\(^{15}\) As explained in Section II, \( B(i) \).

\(^{16}\) As explained in Section II, \( B(iv) \).

\(^{17}\) This particular example was suggested by S. Winograd, now at IBM Corporation.