THE UNIVERSITY OF MICHIGAN

COLLEGE OF LITERATURE, SCIENCE, AND THE ARTS
Department of Philosophy

Technical Report

SEQUENCE GENERATORS AND DIGITAL COMPUTERS

A. W. Burks

J. B. Wright

ORA Project 03105

under contract with:

DEPARTMENT OF THE NAVY
OFFICE OF NAVAL RESEARCH
CONTRACT NO. Nonr 1224(21)
WASHINGTON, D. C.

administered through:

OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

February 1961

In addition to support received from the Office of Naval Research, this research was supported by the U. S. Army Signal Corps through Project MICHIGAN (Contract No. DA-36-039-SC-52654) and by the U. S. Army Office of Ordnance Research (Contract No. DA-20-018-ORD-16971).

Some of the material in this paper was presented at the International Conference on Information Processing, UNESCO, Paris, 15-20 June 1959. An abstract and report of the discussion was published in the Proceedings, UNESCO, Paris, 1960, p. 425.

This paper grew out of some researches on well-behaved nets (see Sec. 3.1); Hao Wang participated in these early investigations and supplied an essential part of the proof of Lemma 3.3-1 for the case of well-behaved nets.

J. Richard Büchi has made many helpful suggestions during the course of our work.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1. SEQUENCE GENERATORS

The basic concept of this paper, that of sequence generator, is a generalization of the concepts of digital computer, finite automaton, logical net, and other information-processing systems. In this subsection, we will define sequence generator and some related concepts and will illustrate them immediately thereafter.

Definitions: A sequence generator $\Gamma = (S, G, R, P^1,\ldots, P^n)$ consists of a set S (whose elements are called complete states), a set G (whose elements are called generators), a binary relation R (called the direct transition relation), and functions $P^1,\ldots, P^n$ (called projections), for some $n = 0, 1, 2, 3,\ldots$, satisfying the conditions: (1) S is finite, (2) G is a subset of S, (3) R is defined on S, and each P^i (for $i = 1, 2,\ldots, n$) is also defined on S. The values of the function P^i, which may be entities of any kind, are called P^i-states.

A sequence generator may be represented by a finite-directed graph whose vertices denote complete states and whose arrows indicate when the direct transition relation holds between two states. In our diagrams,we will use rectangles at those vertices which represent generator states and circles at vertices representing complete states which are not also generators; the names of complete states and of P-states are written in the circles and rectangles (see Figs. 1(b), 2(b), 3, etc.). Though our diagrams are closely related to the usual state diagrams (transition diagrams) employed to represent automata (see, for example, Moore, 1956, p. 134), there are very significant differences. The vertices (nodes) of our diagrams represent complete states, while in the usual state diagrams, the nodes represent internal states. This difference results

from the fact that in sequence generators complete states are basic and input and output states are derived from complete states by means of projections, while in the usual approach complete states are derived by compounding internal states and input states. (The latter process is explained in Section 1.2; we will discuss the relation of the two approaches further in Section 2.1.)

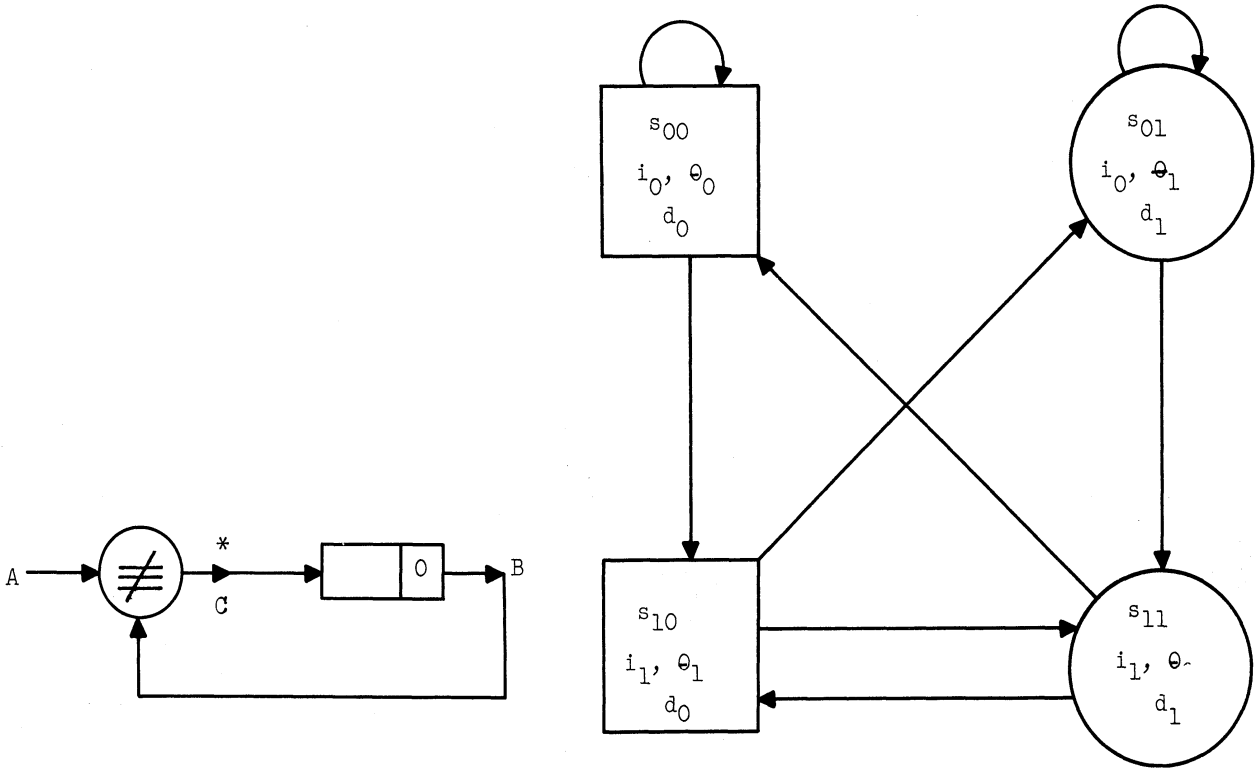
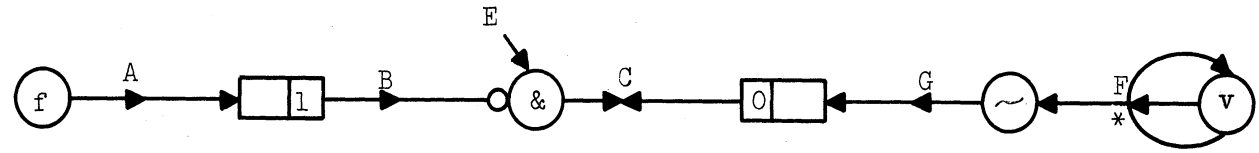
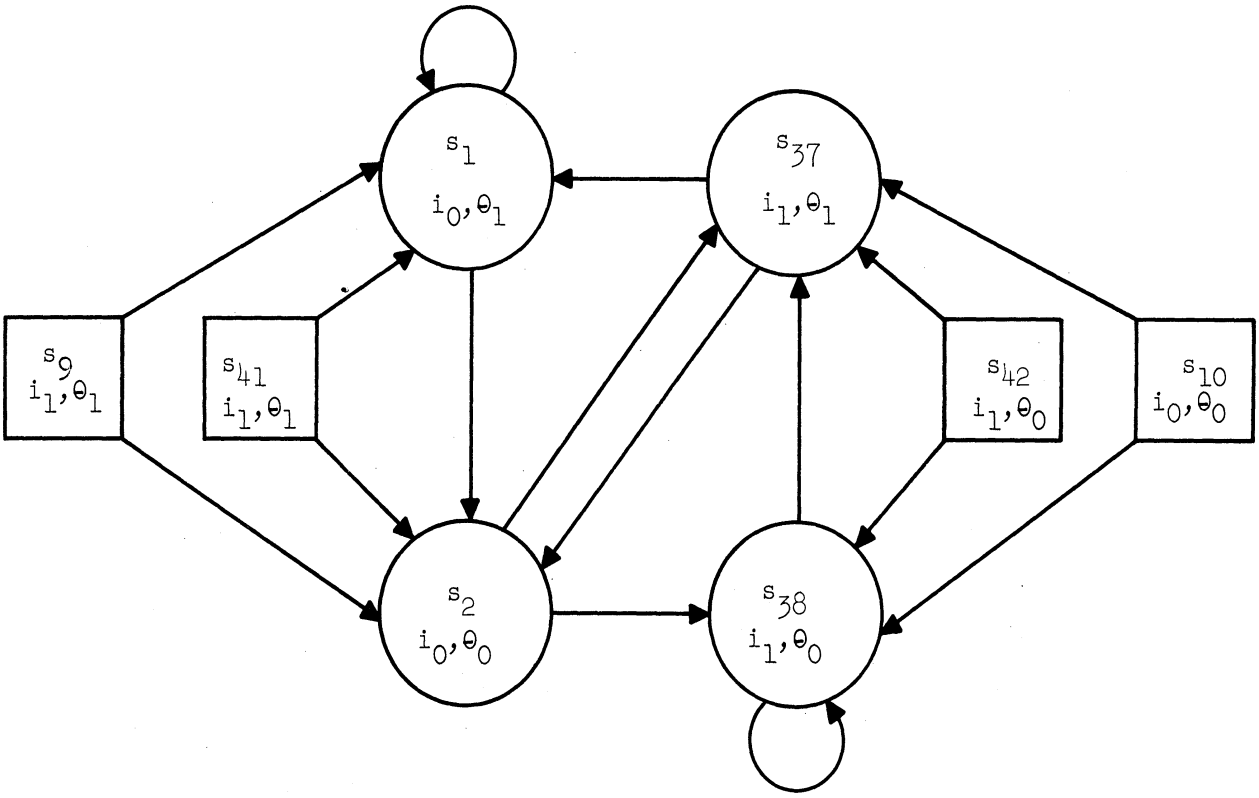


Fig. 1. (a) Binary counter; (b) Three-projection sequence generator $\Gamma = (S, G, R, I, \Theta, D)$ associated with the binary counter (a).

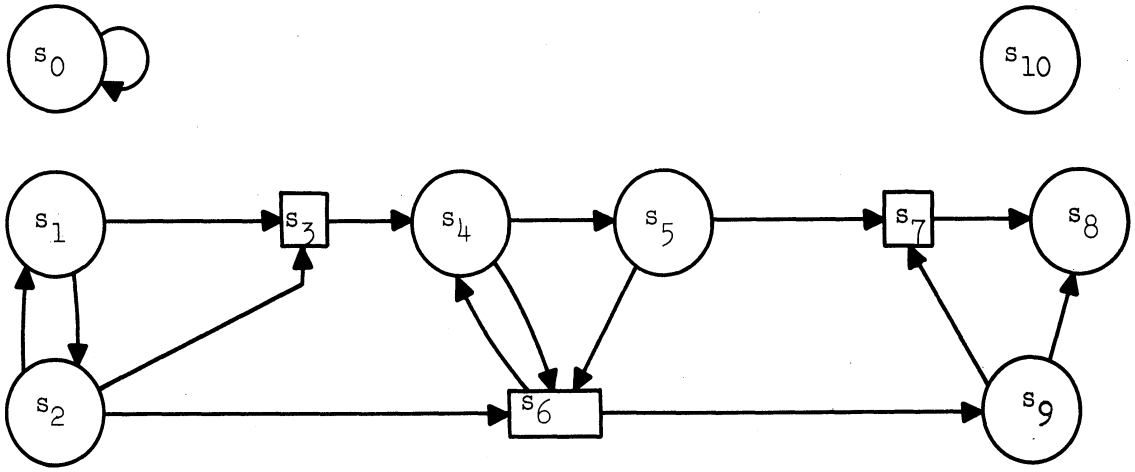


(a)

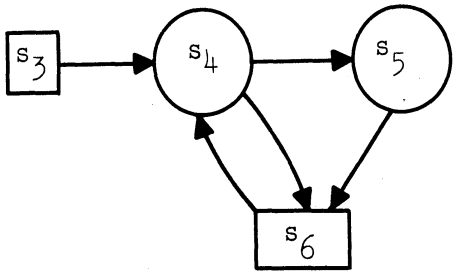


(b)

Fig. 2. (a) Ill-formed net with behavior $F(t) \equiv \sim E(t + 1)$; (b) Sequence generator $\Gamma = (S, G, R, I, \theta)$ associated with net (a).



(a)



(b)

Fig. 3. (a) Sequence generator $\Gamma = (S, G, R)$; (b) $\Gamma^{\dagger}$, the reduced form of Γ.

Some comments and explanations concerning the definition of sequence generator may be helpful. If $n = 0$, then $\Gamma = (S, G, R)$ is a sequence generator with no projections. Though our definition of sequence generator permits any number of projections, in this paper we will be mainly interested in sequence generators with zero, one or two projections. Furthermore, the set of complete states S of a sequence generator may be a null set; in this case the domain of definition of each function P^i will be empty. It is worth noting that essentially (but not quite) the same concept of sequence generator can be obtained without using the set S of complete states in the definition and

then defining S to be the union of G and the field of R.

We will use $[\alpha](j,k)$ (where j is a non-negative integer, k is a non-negative integer or $k = \omega$; $j \leq k$) to denote the sequence $< \alpha(j), \alpha(j+1), \ldots, \alpha(k) >$ when k is finite and the sequence $< \alpha(j), \alpha(j+1), \alpha(j+2), \ldots >$ when $k = \omega$. If P is a projection $P([\alpha](j,k))$ abbreviates the sequence $< P(\alpha(j)), P(\alpha(j+1)), \ldots, P(\alpha(k)) >$ when k is finite and the sequence $< P(\alpha(j)), P(\alpha(j+1)), P(\alpha(j+2)), \ldots >$ when $k = \omega$.

Definitions: Let $\Gamma = (S, G, R, P^1, \ldots, P^n)$ be a sequence generator and let k be a non-negative integer or ω. $[s](0,k)$ is a Γ-sequence if (1) $s(0) \in G$ and (2) for each j, $j < k$, $R(s(j), s(j+1))$. A complete state s is {Γ-accessible} [Γ-admissible] if s occurs in some {-----} [infinite] Γ-sequence.[1]

These concepts may be illustrated by reference to the direct transition diagram of Fig. 3(a). The sequence $< s_7, s_8 >$ is a Γ-sequence, while the sequence $< s_3, s_4, s_5, s_6, s_4, s_6, s_4, s_6, s_4, s_6, \ldots >$ is an infinite Γ-sequence. Complete states s_7, s_8, and s_9 are Γ-accessible but not Γ-admissible; complete states s_3, s_4, s_5, and s_6 are Γ-accessible and Γ-admissible, while states s_0, s_1, s_2, and s_{10} are inaccessible (and hence inadmissable).

Definitions: Let ρ be a binary relation and α a set; we define

$$\rho(\alpha) = \{y \mid (\exists x)\rho(x,y) \ \& \ x \in \alpha\}.$$

A complete state s of $\Gamma = (S, G, R, P^1, \ldots, P^n)$ is a terminal state of Γ if $R(\{s\})$ is null.

A terminal state of Γ is a complete state for which there is no successor by the direct transition relation R. Complete states s_8 and s_{10} are the terminal states of Fig. 3(a). Note that if Γ has no terminal states, every Γ-accessible state is Γ-admissible and vice versa.

[1] In Burks and Wright, 1953, p. 1364, we defined the concept of an admissible state of a net. When a net is converted into a sequence generator (see Section 1.2 below), these states will be accessible rather than admissible in the senses of these terms defined above.

We will sometimes need to combine several projections to make a composite projection of them. For this we will use the notation

$$P^1 \times P^2 \times \ldots \times P^n$$

which is defined by

$$[P^1 \times P^2 \times \ldots \times P^n](s) = \langle P^1(s), P^2(s), \ldots, P^n(s) \rangle.$$

1.2. SPECIAL CASES OF SEQUENCE GENERATORS

Many concepts in the theory of information processing turn out to be special cases of the concept of sequence generator or are closely related to this concept. We will discuss a number of these in the present subsection. Since digital computers (automata) and logical nets are of special interest to us, we will show in detail how the concept of sequence generator applies to them. In later sections we will derive both new and old results about automata and nets from our new theory of sequence generators.

We will begin with well-formed nets, review the method of deriving a finite automaton from a well-formed net, and then show how to derive a sequence generator from a finite automaton. We will use the definition of well-formed net of Burks and Wright, 1953, p. 1361, modified to allow arbitrary switching elements and delay elements whose initial output states are one, as well as delays whose initial output states are zero.[2] In net diagrams, certain nodes (junctions) are designated as net outputs and are distinguished by stars [see Fig. 1(a)].

A well-formed net (w.f.n.) may be analyzed in terms of its input states, delay output states, and net output states. A digital computer represented

[2] Sequence generators may also be derived from automata containing delays whose initial output states are unspecified; these are called "abstract delays" in Burks and Wang, 1953, p. 201, and Burks, 1959, Section 3. But we will not complicate the present discussion by considering automata with such delay elements.

by a w.f.n. operates as follows. The "state" of a net at a given time is determined by its input state i and its delay output state d at that time; these pairs $< i,d >$ are called the complete states of the net. For each time t ($t = 0,1,2,\ldots$) the complete state $< i,d >$ determines the net output state e at the same time (t) in accordance with an output function λ, i.e., $e = \lambda(i,d)$. At time 0 the delay output state d_0 is uniquely determined by the initial delay output states of the delay elements. For each time t the complete state $< i,d >$ determines the delay output state d_1 at the next moment of time (t + 1) in accordance with a direct transition function τ, i.e., $d_1 = \tau(i,d)$. The net of Fig. 1(a) is a well-formed net which represents a binary counter. A is the input node, the starred node C is its net output node, and B its delay output node (the initial state of B is zero). The state of C at t indicates the binary count, i.e., the number modulo 2 of 1's which have appeared (during the interval of time $0,\ldots,t$) on the input node A. The state analysis is given by the following table, where 0 is the initial delay output state.

i(t)	d(t)	d(t + 1) = τ(i,d)	e(t) = λ(i,d)
A	B	B	C
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Definition: A finite automaton is a sextuple $< \{i\}, \{d\}, \{e\}, d_0, \tau, \lambda >$ where $\{i\}$, $\{d\}$, $\{e\}$ are finite non-empty sets (whose elements are called input states, internal states, and output states, respectively), $d_0 \in \{d\}$ (d_0 is called the initial internal state), τ is a function from the Cartesian product $\{i\} \times \{d\}$ into $\{d\}$ (called the direct transition function), and λ is a function from the Cartesian product $\{i\} \times \{d\}$ onto $\{e\}$ (called the output function).

(This is essentially the definition of Burks and Wang, 1956, p. 203; see also Moore, 1956, p. 133.) The procedure for analyzing a well-formed net which is described in the preceding paragraph clearly converts a well-formed net into a finite automaton. This procedure is reversible; that is, given a finite automaton, one can construct a well-formed net which realizes it. Thus the concepts of well-formed net and finite automaton are basically equivalent and either can be taken as a formal definition of the concept "finite digital computer." (See Church, 1955, Kleene, 1956, p. 5, and Burks, 1959, Section 3, for other definitions of these concepts.)

A three-projection sequence generator $\Gamma = (S, G, R, I, \Theta, D)$ may be associated with a finite automaton as follows. The elements of S are the complete states $< i,d >$ and the elements of G are the complete states $< i,d_0 >$. The direct transition relation is defined by $R(< i_1,d_1 >, < i_2,d_2 >) \equiv [d_2 = \tau(i_1,d_1)]$ and the input, output, and internal state projections by $I(< i,d >) = i$, $\Theta(< i,d >) = \lambda(i,d)$, and $D(< i,d >) = d$, respectively. The sequence generator associated with the binary counter of Fig. 1(a) is represented by Fig. 1(b). As before, the rectangles represent elements of G. The subscripts on the complete states correspond to the nodes of the counter in the order A, B. $< s_{00},s_{10},s_{11},s_{00},s_{10},s_{01} >$ is an example of a finite Γ-sequence; in it the input sequence i_0,i_1,i_1,i_0,i_1,i_0 produces the output sequence e_0,e_1,e_0,e_0,e_1,e_1 (and thus three "ones" on the input leave the counter recording "one"). Note that, though the internal states d_0,d_1 are represented in Fig. 1(b), the nodes of the graph correspond to complete states and not to internal states, as is the case with the usual state diagrams used to represent automata.

We have shown how to transform a well-formed net into a finite automaton and vice-versa. We have also shown how to derive a sequence generator from a finite automaton. The latter process is not in general reversible. Only

certain sequence generators (those which are deterministic) may be realized by finite automata (see Section 2.1).

Our next application of sequence generators is to arbitrary "nets," including nets that are not well-formed. We will use the concept of Burks and Wright, 1953, p. 1353, modified to allow arbitrary switching elements and both kinds of concrete delays. Each switch element translates into a switch equivalence which gives the state of the switch output as a truth function of the switch input, and each delay element translates into two delay equivalences, called the "initial delay equivalence" and the "recursive delay equivalence." The initial delay equivalence gives the initial state of the delay output and the recursive delay equivalence equates the delay output at any time other than 0 to the delay input at the previous time. Hence, each net translates into a conjunction of equivalences. If the net is not well-formed, this conjunction will not directly correspond to (will not give the structure of) a digital computer, but it may specify a computation or behavior condition on a digital computer (see Section 4), and on this account is of interest. Figure 2(a) shows a net with input node E and output node F. The non-input switch element driving node A represents the contradictory or "always false" truth function. The initial state of the delay AB is "true," which for coding reasons we represent by "1"; the initial state of the delay $\mathfrak{C}$C is 0. The switch equivalences for this net are $A(t) \equiv 0$, $F(t) \equiv F(t)$, and $C(t) \equiv [E(t) \;\&\; \overline{B(t)}]$. $B(0) \equiv 1$ and $C(0) \equiv 0$ are the initial delay equivalences, while $B(t + 1) \equiv A(t)$ and $C(t + 1) \equiv F(t)$ are the recursive delay equivalences.

A two-projection sequence generator $\Gamma = (S, G, R, I, \Theta)$ may be associated with an arbitrary net in the following way:[3] A complete state s is an assign-

[3] If the net is well-formed either the procedure about to be described or the procedure described earlier may be used. The resultant sequence generator will, of course, be different in the two cases.

ment of a truth value to each node of the net which makes the switch equivalences of the net true. An element s of S is a generator (element of G) if s assigns to the delay output nodes truth values which make the initial delay equivalences true. $R(s_1,s_2)$, where s_1, $s_2 \in S$, if and only if the truth values which s_1 assigns to the delay input nodes and the truth values which s_2 assigns to the delay output nodes satisfy the recursive delay equivalences. For each complete state s, {I(s)} [Θ(s)] is s cut down to the {input} [output] nodes [i.e., {I(s)}[Θ(s)] is the net {input} [output] state contained in s]; the input projection will not exist if there are no input nodes.

The sequence generator $\Gamma = (S, G, R, I, \Theta)$ associated with Fig. 2(a) is represented by Fig. 2(b). Though there are 6 nodes in the net, there are only 8 complete states. The subscripts on the state symbols s_1, s_{37}, etc., are the decimal codings of the binary representations of the states of the nodes taken in the order E, A, B, C, F, G; e.g., the subscript on s_9 decodes into 001001, showing that in this state nodes B and G are active while the remaining nodes are inactive. The subscript of the input state i is the state of node E and the subscript on the output state θ is the state of node F. $< s_{10}, s_{37}, s_2, s_{38}, s_{37} >$ is a Γ-sequence which has a derived input sequence $< i_0, i_1, i_0, i_1, i_1 >$ and a derived output sequence $< \theta_0, \theta_1, \theta_0, \theta_0, \theta_1 >$. It can be proved that $F(t) \equiv \sim E(t + 1)$; such behavior would not, of course, be possible in a well-formed net.

Our process for associating a sequence generator with an arbitrary net is different from our process for associating a sequence generator with a well-formed net in the following basic respect. In the latter case we first defined input states, delay output states (internal states), and output states for the net, and then compounded complete states from input states and internal states. On the other hand, in associating a sequence generator with an arbitrary net, we first defined states (complete states) over every node,

and then derived input and output states by means of projections. It turns out that in general not every assignment of truth values to the input nodes of an arbitrary net is an input state. In fact, we know of no way of defining states for parts of a net (input, internal, and output states) without presupposing states for the whole net (complete states). Indeed, it was our work with arbitrary nets which led us to consider sequence generators (in which complete states are basic; input, internal, and output states derivative).

This completes our discussion of the method of transforming an arbitrary net into a two-projection sequence generator. This process may be reversed; that is, given any sequence generator with two projections, one can find a corresponding net. It is not difficult to construct this procedure (for going from a sequence generator to a net) from the information to be given in Section 4.2, so we will not describe it here.

There are other entities besides nets and well-formed nets (digital computers) which are either sequence generators or are closely related to sequence generators. The concept of a non-deterministic automaton of Rabin and Scott, 1959, Definition 9, is quite similar to our concept of a sequence generator. Sequence generators are in a certain sense equivalent to formulas constructed from truth functional connectives, monadic predicates, one individual variable "t" (which ranges over discrete times), the successor function, and zero (see Sections 4.2 and 4.3). The following are special cases of sequence generators: a finite state grammar (Chomsky and Miller, 1958, p. 95); sequential circuits representable in combinatory logic (Fitch, 1958, p. 263); incompletely specified automata, i.e., automata in which certain sequences of input states are proscribed (Aufenkamp and Hohn, 1957, Section IV); automata with terminal states (ibid., Section VII); and the flow diagrams used in programming a digital computer. A sequence generator may be used to characterize a class of finite sequences defined by a regular expression (see Section 4.4). Finite graphs may

be used to analyze certain games (König, 1936; and McKinsey, 1952, Chapter 6). There is an obvious relation between finite graphs and sequence generators, and hence some problems concerning games may be studied by means of sequence generators; we will give an example in the next subsection. Though he makes no reference to automata theory, Putnam, 1957, pp. 44-49, uses sequence generators to establish some results about satisfiability; he uses the concept of admissibility in connection with "Γ-sequences" which are infinite in both directions. We remark finally that Harary and Paper, 1957, in applying relational logic to linguistics use ideas closely related to the concept of sequence generator.

Though we have noted a number of applications of the concept of sequence generator, we wish to make it clear that we are not attempting in the present paper to solve all the problems that have been considered for these applications. In the next subsection we will establish some results concerning infinite Γ-sequences for sequence generators without projections. In Section 2 we will treat some concepts in which a single projection plays an essential role, and in Section 3 we will work with concepts in which two projections play a special role. In Section 4 we will present some generalizations and further applications of sequence generators.

1.3. REDUCED FORM ALGORITHM

Algorithms play a fundamental role in this paper, so we will make a few informal comments about them. An algorithm presupposes a well-defined set of entities, called "the domain of the algorithm." An algorithm is a finite system of rules which may be mechanically applied to any entity of its domain. An algorithm which terminates in a finite number of steps when applied to any entity of its domain is called a "terminating algorithm." The Reduced Form Algorithm to be described soon is a terminating algorithm, since, when it is

applied to any sequence generator, it will eventually terminate in a sequence generator. An algorithm with a domain D is called a decision procedure for a class A which is a subset of D, if for every element of D which belongs to A, the algorithm terminates in "yes," and for every element of D which does not belong to A, the algorithm terminates in "no." The truth table procedure is a decision procedure for the class of tautologies of the propositional calculus.

Before formulating the Reduced Form Algorithm, we will describe informally what it does. Let us call a state s of a sequence generator $\Gamma = (S, G, R, P^1,\ldots,P^n)$ "extendable" if there is an infinite sequence of complete states $< s(0), s(1), s(2),\ldots >$ such that s(0) is s and $R[s(i), s(i + 1)]$ for $i = 0, 1, 2,\ldots$. (Note that s is not necessarily a generator, and so the infinite sequence of complete states is not necessarily a Γ-sequence.) In Fig. 3 states s_0 and s_1 are extendable, while states s_7 and s_{10} are not. The Reduced Form Algorithm may be applied to any sequence generator Γ. In part 1 of the algorithm the operation of deleting terminal states is iterated until we arrive at a sequence generator $\ddot{\Gamma}$ with no terminal states. Since a sequence generator has non-extendable states if and only if it has terminal states, $\ddot{\Gamma}$ is essentially the result of deleting all non-extendable states from Γ. In part 2 of the algorithm, one begins with the generators of $\ddot{\Gamma}$ (and hence of Γ), and by a succession of steps obtains the accessible states of $\ddot{\Gamma}$. A new sequence generator $\Gamma^\dagger$, called the reduced form of Γ, is defined on the basis of the states so obtained. Since a state is admissible if and only if it is both extendable and accessible, $\Gamma^\dagger$ is just Γ cut down to its admissible states (see Theorem 1.3-1).

Algorithm (Reduced Form Algorithm): Consider any sequence generator $\Gamma = (S,G,R,P^1,\ldots,P^n)$.

(1) Form a new sequence generator by deleting all the terminal states of Γ. Iterate this process until you arrive at a sequence generator with no terminal

states. Call this final sequence generator $\ddot{\Gamma} = (\ddot{S},\ddot{G},\ddot{R},\ddot{P}^1,\ldots,\ddot{P}^n)$.

(2) Define A_i inductively by

$$A_0 = \ddot{G}$$

$$A_{i+1} = \ddot{R}(A_i)$$

Form the sequence $A_0,A_1,A_2\ldots$, stopping when $A_{m+1} \subset U_{i=0}^{m} A_i$. (Note: "$\alpha \subset \beta$" means that α is either included in β or equals β.) Let $\dot{S} = U_{i=0}^{m} A_i$, $\dot{G} = \ddot{G}$, and let $\{\dot{R}\}[P^i]$ be the {relation R} [projection P^i] cut down to $\dot{S}$. Define $\Gamma^\dagger = \dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P}^1,\ldots,\dot{P}^n)$.

We will illustrate the Reduced Form Algorithm. Let $\Gamma = (S,G,R)$ be the sequence generator represented by Fig. 3(a), with those complete states which belong to G being designated by rectangles. In part 1 of the algorithm we delete states s_8 and s_{10}, then state s_7, and then state s_9. $\ddot{S}$ consists of the remaining complete states, $\ddot{G}$ contains s_3 and s_6, and $\ddot{R}$ is R cut down to $\ddot{S}$. We have at the end of part 1 the sequence generator $\ddot{\Gamma}$ represented by the result of deleting everything to the right of state s_5 in Fig. 3(a). In part 2 of the algorithm we form the sequence $\{s_3,s_6\}[=A_0]$, $\{s_4\}[=A_1]$, $\{s_5,s_6\}[=A_2]$, $\{s_4,s_6\}[=A_3]$. Simultaneously we form the sequence $\{s_3,s_6\}[=U_{i=0}^{0} A_i]$, $\{s_3,s_4,s_6\}[=U_{i=0}^{1} A_i]$, $\{s_3,s_4,s_5,s_6\}[=U_{i=0}^{2} A_i]$, stopping at this point since $\{s_4,s_6\} \subset \{s_3,s_4,s_5,s_6\}$, i.e., $A_3 \subset U_{i=0}^{2} A_i$. Hence $S = \{s_3,s_4,s_5,s_6\}$ and $\Gamma^\dagger$, the reduced form of Γ, is represented by Fig. 3(b).

Theorem 1.3-1: The Reduced Form Algorithm, when applied to any sequence generator Γ, always terminates in a sequence generator $\Gamma^\dagger$. The set of complete states of $\Gamma^\dagger$ equals the set of Γ-admissible complete states.

As a step toward proving this theorem, we first establish

Lemma 1.3-2: Let ρ be a binary relation and δ_0 a set. Define δ_i for $i = 1,2\ldots$ inductively by $\delta_{i+1} = \rho(\delta_i)$ and let $\alpha_\ell = U_{i=0}^{\ell} \delta_i$ for $\ell = 0,1,2,\ldots$. If, for some j, $\alpha_j = \alpha_{j+1}$ then for all ℓ, $\alpha_\ell \subset \alpha_j$.

Proof: We note first that since the operator ρ may be distributed over the union, $\alpha_{\ell+1} = \alpha_\ell \cup \rho(\alpha_\ell)$. We now assume $\alpha_j = \alpha_{j+1}$ and prove that $\alpha_\ell \subset \alpha_j$, proving first by induction that for all $\ell \geq j$, $\alpha_\ell = \alpha_j$. The initial step is covered by the hypothesis that $\alpha_{j+1} = \alpha_j$. For the general step assume $\alpha_k = \alpha_j$, where $k > j$. By the fact noted above, $\alpha_{k+1} = \alpha_k \cup \rho(\alpha_k)$ and $\alpha_{j+1} = \alpha_j \cup \rho(\alpha_j)$. Combining the four preceding equalities, we get $\alpha_{k+1} = \alpha_j$. To conclude the proof, we note that it follows directly from the definition of α_ℓ that for $\ell < j$, $\alpha_\ell \subset \alpha_j$.

We turn now to the proof of Theorem 1.3-1. We will use freely the notation of the algorithm. (I) We prove first that the Reduced Form Algorithm, when applied to any sequence generator Γ, always terminates in a sequence generator $\Gamma\dagger$. Since S is a finite set, the first part of the algorithm terminates in a sequence generator $\ddot{\Gamma}$. The criterion for stopping in part 2 of the algorithm is based on a monotonically increasing sequence of subsets, of $\ddot{S}$, which is a finite set, so the second part of the algorithm will always terminate. Finally, it is clear that a sequence generator $\Gamma\dagger$ is defined in part 2 of the algorithm.

(II) We prove next that the set of complete states of $\Gamma\dagger$ equals the set of Γ-admissible complete states. (IIA) We consider a Γ-admissible complete state s_1 and show that $s_1 \in \dot{S}$. Since s_1 is Γ-admissible, there is an infinite sequence $[s](0,\omega)$ of Γ-admissible states such that for some k, $[s](k) = s_1$. A complete state of Γ is deleted by part 1 of the algorithm only if it cannot belong to an infinite Γ-sequence, and so $[s](0,k)$ is a $\ddot{\Gamma}$-sequence. Hence by the definition of A_k in the algorithm, $s_1 \in A_k$ and $s_1 \in U_{i=0}^{k} A_i$. We now apply Lemma 1.3-2, letting $\rho = \ddot{R}$ and $\delta_0 = \ddot{G}$. By (I) above, part 2 of the algorithm terminates; in the notation of the algorithm $A_{m+1} \subset U_{i=0}^{m} A_i$. The result of Lemma 1.3-2, put in this notation, is that for all ℓ, $U_{i=0}^{\ell} A_i \subset U_{i=0}^{m} A_i$. We have already shown that $s_1 \in U_{i=0}^{k} A_i$, and

so $s_1 \in U_{i=0}^{m} A_i$. But in the algorithm $\dot{S}$ is defined to be $U_{i=0}^{m} A_i$ and so $s_1 \in \dot{S}$.

(IIB) We next consider a complete state $s_1 \in \dot{S}$ and show that s_1 is Γ-admissible. In the notation of the algorithm $\dot{S} = U_{i=0}^{m} A_i$ and so $s_1 \in U_{i=0}^{m} A_i$. Hence s_1 is $\ddot{\Gamma}$-accessible. Part 1 of the algorithm terminates in a sequence generator $\ddot{\Gamma}$ with no terminal states. As remarked in Section 1.1, every accessible state of a sequence generator with no terminal states is an admissible state. Consequently, there exists an infinite $\ddot{\Gamma}$-sequence $[s](0,\omega)$ such that for some k, $s_1 = [s](k)$. By the nature of part 1 of the algorithm, $[s](0,\omega)$ is also an infinite Γ-sequence, and so s_1 is Γ-admissible.

Corollary 1.3-3: (a) Every complete state of $\Gamma\dagger$ is $\Gamma\dagger$-admissible.

(b) The set of infinite Γ-sequences equals the set of infinite $\Gamma\dagger$-sequences.

(c) Every finite $\Gamma\dagger$-sequence is an initial segment of an infinite Γ-sequence.

We will next discuss the Reduced Form Algorithm and some alternatives to it. Applied to an arbitrary sequence generator Γ, part 1 of the Reduced Form Algorithm produces the set of extendable states of Γ. Applied to an arbitrary sequence generator Γ, part 2 of the algorithm produces the set of Γ-accessible states. Since a complete state is admissible if and only if it is both extendable and accessible, the two parts of the Reduced Form Algorithm applied to a sequence generator Γ in either order produce the same sequence generator $\Gamma\dagger$. A sequence generator Γ derived from a well-formed net in the way indicated in Section 1.2 has no terminal states; consequently, when part 2 of the Reduced Form Algorithm is applied to Γ, it produces $\Gamma\dagger$.

There is an alternative procedure for finding the Γ-admissible complete states of a sequence generator. Let x be the number of complete states of Γ. Form all Γ-sequences of length x + 1; it can be proved that a state is Γ-accessible if and only if it occurs in one of these sequences. To find the Γ-admissible states, we operate on each sequence as follows: proceeding through the

sequence $< s(0), s(1),\ldots,s(x) >$ check an occurrence of a state whenever that state has occurred earlier in the same sequence; then delete all states which follow the last checked state. It can be shown that a state is Γ-admissible if and only if it occurs in one of the resultant sequences. This method of finding the Γ-admissible states can be made the essence of an alternative reduced form algorithm which is simpler to formulate and easier to prove adequate than our Reduced Form Algorithm. It is less efficient, however: in the example given earlier, $m = 2$ while $x = 11$. These differences seem to result from the following fundamental difference between these two algorithms. In the Reduced Form Algorithm the length of the computation is not specified in advance; rather, parts 1 and 2 each contain an internal "stop criterion": one proceeds until he is stopped by these criteria. In contrast, the alternative algorithm first establishes the length of the computation on the basis of a general property of the sequence generator (the number of complete states); since this length is established a priori, it is of course determined by the worst case, even though in most cases far fewer steps would have sufficed. This is analogous to the contrast between asynchronous circuits, in which completion of an operation is sensed and the next operation begun immediately, and synchronous circuits, in which the same amount of time is allowed for a given operation in every case, and this is, of course the time required for the worst case (plus a "safety factor"!). We have presented the more efficient of these two algorithms, although it is more difficult to formulate and prove adequate, because finding the reduced form is basic to so many automata algorithms; see, for example, Sections 2.3 and 3.4. But though in many later cases we know of more efficient algorithms (see, for example, the alternative to the h-univalence Decision Procedure in Section 3.4), we will not present them because we feel that perspicuity of theory and simplicity of exposition are more important there.

We mentioned in Section 1.2 that certain puzzles give rise to sequence generators. The so-called "15 puzzle" is a good example since it may be solved by means of our Reduced Form Algorithm. The puzzle consists of a 4 x 4 array of 15 movable blocks (numbered 1 through 15) and one empty position. A "move" consists in changing a pattern into any one of the (at most) four patterns obtained by shifting a block into the (neighboring) empty space. The problem is to achieve a stipulated pattern by a succession of moves starting from a given pattern. A sequence generator $\Gamma = (S,G,R,P)$ corresponding to the puzzle may be defined as follows. The 4 x 4 matrices whose entries are the numbers 0 through 15 are the complete states of Γ; there are 16! of these. The starting pattern is the sole generator of Γ. Two states s_1 and s_2 stand in the direct transition relation R if there is a move taking the pattern corresponding to s_1 into the pattern corresponding to s_2. The projection P has the value 1 on the single pattern stipulated to be the goal and 0 otherwise. The problem is solved by constructing a finite Γ-sequence $< s(0), s(1), s(2),\ldots,s(t) >$ such that $P[s(t)] = 1$, if such a sequence exists. Clearly this sequence exists if and only if the complete state with a projection of 1 is Γ-accessible. Whether or not this is the case can be determined by applying part 2 of the Reduced Form Algorithm to Γ: if such a sequence exists, it will be found in the course of carrying out the algorithm.[4] It turns out that exactly half of the complete states of Γ are Γ-accessible and that each of these Γ-accessible states is also Γ-admissible.

[4]There is a much simpler algorithm for finding the Γ-accessible states of this particular sequence generator. See W. W. R. Ball, Mathematical Recreations and Essays, Macmillan, 1940, pp. 299-303.

2. SEQUENCE GENERATORS WITH ONE PROJECTION

2.1. DEFINITIONS

In the last sub-section we made no particular use of the projections of a sequence generator. In this section we shall define some concepts which apply primarily to sequence generators with one projection and will prove some theorems about these concepts. In most applications this single projection is an input projection, an output projection, or a combined input-output projection. In the next section we will work with sequence generators having two projections. These two projections will usually be an input and an output projection.

Definition: The behavior of $\Gamma = (S, G, R, P^1, P^2, \ldots, P^n)$, where $n > 0$, is the set $P([s](0,k))$, where $P = P^1 x\, P^2 x \ldots P^n$ and $[s](0,k)$ is a Γ-sequence (finite or infinite). "$\mathcal{B}(\Gamma)$" denotes the behavior of Γ. The infinite behavior of Γ, denoted by "$\mathcal{B}^{\omega}(\Gamma)$", is the set of infinite sequences in $\mathcal{B}(\Gamma)$. Corollary 1.3-3b can now be reformulated as follows.

Corollary 2.1-1: $$\mathcal{B}^{\omega}(\Gamma) = \mathcal{B}^{\omega}(\Gamma^{\dagger}).$$

It is worth noting that in general it is not true that for a sequence generator $\Gamma = (S, G, R, P)$ there exists a sequence generator $\dot{\Gamma}$ such that the set of $\dot{\Gamma}$-sequences equals the behavior of Γ. This may be shown by a simple example. Let $S = \{s_0, s_1, s_2\}$, $G = \{s_0\}$, $R = \{< s_0, s_1 >, < s_1, s_2 >, < s_2, s_0 >\}$, and $P(s_0) = P(s_1) = p_0$, $P(s_2) = p_1$. There is one infinite Γ-sequence $< s_0, s_1, s_2, s_0, s_1, s_2, s_0, s_1, s_2 \ldots >$ and the behavior of Γ consists of the sequence $< p_0, p_0, p_1, p_0, p_0, p_1, \ldots >$ and all its initial segments, and does not include the infinite sequence $< p_0, p_0, p_0, \ldots >$. Consider a sequence generator $\dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R})$ such that $\{p_0, p_1\} \subset \dot{S}$ and such that $< p_0, p_0, p_1, p_0, p_0, p_1, \ldots >$ is an infinite $\dot{\Gamma}$-sequence. It follows from the existence of this sequence

that $\dot{R}(p_o,p_o)$ and $p_o \in \dot{G}$, and hence that $< p_o,p_o,p_o,\ldots >$ is an infinite $\dot{\Gamma}$-sequence.

Some remarks about the application of the concept of behavior to nets will be appropriate. By the methods of Section 1.2, we can associate with every net (well-formed or not) a sequence generator $\Gamma = (S, G, R, I, \Theta)$, where I is the input projection and Θ is the output projection. The behavior of a digital computer (w.f.n.) consists of the relationship between its inputs and its outputs, and similarly for an arbitrary net. The behavior of a net may be regarded as the set of sequences (finite and infinite) of pairs $< i(0), \theta(0) >$, $< i(1), \theta(1) >$, $< i(2), \theta(2) >$,... for which there is a Γ-sequence $[s](0,k)$ such that $i(t) = I\{s(t)\}$ and $\theta(t) = \Theta\{s(t)\}$ for every t. This is clearly $\beta(\Gamma)$, the behavior of the sequence generator $\Gamma = (S, G, R, I, \Theta)$. In Section 2.3 we present a Behavior Inclusion Procedure to be applied to a pair $< \Gamma, \dot{\Gamma} >$ to decide whether the behavior of Γ is included in the behavior of $\dot{\Gamma}$; when applied to the pair $< \dot{\Gamma}, \Gamma >$ as well as to the pair $< \Gamma, \dot{\Gamma} >$, this tells us whether the behaviors of Γ and $\dot{\Gamma}$ are equal. Thus, through these considerations, the Behavior Inclusion Procedure can be used to decide whether the behavior of an arbitrary net N is included in or equal to the behavior of a net $\dot{N}$. In the case of well-formed nets, however, a much more efficient algorithm for deciding equality of behaviors is known (Burks, Wang, 1956, Section 2.2); a basic part of this algorithm consists essentially of finding the reduced form (Section 1.3) of a sequence generator associated with the combined nets. Actually this algorithm applies to any deterministic sequence generator (this concept is defined below); moreover, if Γ and $\dot{\Gamma}$ are both deterministic and $\beta(\Gamma) \subset \beta(\dot{\Gamma})$, then $\beta(\dot{\Gamma}) \subset \beta(\Gamma)$, so this algorithm also answers the question as to whether $\beta(\Gamma) \subset \beta(\dot{\Gamma})$ for the case of deterministic sequence generators.

The following lemma will be needed in subsequent proofs. It is a classical

interpretation of Brouwer's Fan theorem (Heyting, 1956, pp. 42-43) and is closely related to König's Infinity Lemma concerning infinite graphs (König, 1936, p. 81). Our lemma, however, is stronger than König's Infinity Lemma in that it does not require that the α's be pairwise disjoint; because of this difference, we present a proof of it here.

Lemma 2.1-2: Let $< \alpha_0,\alpha_1,\alpha_2,\ldots >$ be an ω-sequence of finite non-empty sets and let ρ be a binary relation. If for every $x \in \alpha_{i+1}$ there is a $y \in \alpha_i$ such that $\rho(y,x)$, then there is an infinite sequence $< z_0,z_1,z_2,\ldots >$ such that for each i, $z_i \in \alpha_i$ and $\rho(z_i,z_{i+1})$.

Proof: Let β_i consist of all finite sequences $< x_i,x_{i+1},\ldots,x_{i+k} >$ where $k = 0,1,2,\ldots,x_j \in \alpha_j$ for $i \leq j \leq i + k$, and $\rho(x_j, x_{j+1})$ for $i \leq j < i + k$. It follows from the requirement on ρ in the hypothesis of the lemma that for each i,k, and element y_{i+k} of α_{i+k} there is an element of β_i $< y_i, y_{i+1},\ldots, y_{i+k} >$. Since this is so for any k, each β_i is infinite. We will now define by induction the desired sequence $< z_0,z_1,z_2,\ldots >$.

Initial step: Since β_0 is infinite while α_0 is finite there will be some element z_0 of α_0 such that an infinite number of elements of β_0 begin with z_0. Let δ_0 be the subset of β_0 all of whose elements begin with z_0.

General step: Assume given a sequence $< z_0,z_1,\ldots, z_i >$ (where $i = 0,1,2,\ldots$) which belongs to β_0 and satisfies the condition that the set δ_i of elements of β_i which begin with z_i is infinite. The result δ'_{i+1} of deleting the first element of each member of β_i is an infinite subset of β_{i+1}. Since α_{i+1} is finite, there will be some element z_{i+1} of α_{i+1} such that $\rho(z_i, z_{i+1})$ and an infinite number of elements of δ'_{i+1} begin with z_{i+1}. Let δ_{i+1} be the subset of δ'_{i+1}, all of whose elements begin with z_{i+1}; δ'_{i+1} is a subset of β_{i+1} and hence δ_{i+1} is also. Hence $< z_0,z_1,\ldots,z_i,z_{i+1} >$ belongs to β_0 and satisfies the condition that the set δ_{i+1} of elements of β_{i+1} which begin with z_{i+1} is infinite. Thus the inductive hypothesis has been established for the

sequence $< z_0, z_1, \ldots, z_i, z_{i+1} >$. This completes the proof of Lemma 2.1-2.

It may be shown by means of this lemma that a sequence of P-states is an element of the behavior of a sequence generator if and only if every initial segment is.

Theorem 2.1-3 (Infinity Theorem): Let $\Gamma = (S,G,R,P)$ be a sequence generator with behavior $\mathfrak{B}(\Gamma)$ and let $[p](0,k)$, $k = 0,1,2,\ldots,\omega$, be a sequence of P-states. $[p](0,k) \in \mathfrak{B}(\Gamma)$ if and only if for every finite $i \leq k$, $[p](0,i) \in \mathfrak{B}(\Gamma)$.

Proof: The proof of the theorem for finite k is obvious. It is also obvious that $[p](0,\omega) \in \mathfrak{B}(\Gamma)$ implies that for every finite i, $[p](0,i) \in \mathfrak{B}(\Gamma)$. It remains to be proved that if $[p](0,i) \in \mathfrak{B}(\Gamma)$ for every finite i, then $[p](0,\omega) \in \mathfrak{B}(\Gamma)$. We define α_i by $s_1 \in \alpha_i$ if there exists a Γ-sequence $[s](0,i)$ such that $[p](0,i) = P\{[s](0,i)\}$ and $s_1 = s(i)$. It is clear that each α_i is finite and non-empty. We let $\rho = R$ and show that the hypothesis of Lemma 2.1-2 is satisfied. Suppose $s_2 \in \alpha_{i+1}$. By definition of α_{i+1} there exists a Γ-sequence $[s_3](0, i + 1)$ such that $[p](0,i + 1) = P\{[s_3](0, i + 1)\}$ and $s_2 = [s_3](i + 1)$. Let $s_4 = s_3(i)$. By the definition of α_i, $s_4 \in \alpha_i$ and by the definition of a Γ-sequence, $R(s_1,s_2)$. By Lemma 2.1-2 there is an infinite Γ-sequence $[s_5](0,\omega)$ and by the definition of α_i we have $P\{[s_5](0,\omega)\} = [p](0,\omega)$. Hence $[p](0,\omega) \in \mathfrak{B}(\Gamma)$.

The following is a corollary of the Infinity Theorem. Let $\Gamma = (S, G, R, P)$ and $\dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{P})$ be two sequence generators and suppose that for every finite k, if $[p](0,k) \in \mathfrak{B}(\Gamma)$ then $[p](0,k) \in \mathfrak{B}(\dot{\Gamma})$; then $\mathfrak{B}(\Gamma) \subset \mathfrak{B}(\dot{\Gamma})$. For consider any infinite sequence $[p](0,\omega) \in \mathfrak{B}(\Gamma)$. By the Infinity Theorem, for each k $[p](0,k) \in \mathfrak{B}(\Gamma)$. Then by hypothesis, for each k $[p](0,k) \in \mathfrak{B}(\dot{\Gamma})$. Finally, by the Infinity Theorem $[p](0,\omega) \in \mathfrak{B}(\dot{\Gamma})$. This result holds for Γ and $\dot{\Gamma}$ interchanged, of course, so we have: if for every finite k $[p](0,k) \in \mathfrak{B}(\Gamma) \equiv [p](0,k) \in \mathfrak{B}(\dot{\Gamma})$, then $\mathfrak{B}(\Gamma) = \mathfrak{B}(\dot{\Gamma})$. Thus the Infinity Theorem shows that the "finite" behavior of a sequence generator determines its (complete) behavior.

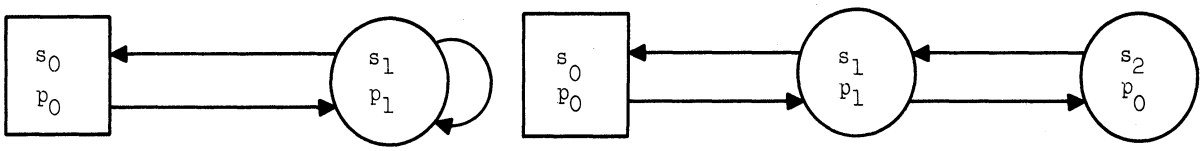
Definitions: Let $\Gamma = (S, G, R, P)$ be a sequence generator. Γ is solvable if every infinite sequence of P-states belongs to its behavior. Γ is {semi-deterministic} [deterministic] if it satisfies the conditions:

(1) For any P-state p, there is {at most one} [exactly one] complete state s of Γ such that $s \in G$ and $P(s) = p$.

(2) For any complete state s_1 and any P-state p of Γ, there is {at most one} [exactly one] complete state s_2 such that $R(s_1, s_2)$ and $P(s_2) = p$.

It is obvious from the definition of {semi-determinism} [determinism] that there is a decision procedure for the class of {semi-deterministic} [deterministic] sequence generators. The problem of solvability is not so simple, but we will later develop a decision procedure for solvability (see Theorem 2.3-2).

Let us illustrate these concepts. The sequence generator of Fig. 1(b) less its last two projections, is clearly deterministic. The sequence generator of Fig. 4(a) is semi-deterministic but not solvable, while the sequence generator of Fig. 4(b) is neither semi-deterministic nor solvable.



(a) (b)

Fig. 4. (a) Semi-deterministic non-solvable sequence generator; (b) Sequence generator neither solvable nor semi-deterministic.

By simple inspection it can be ascertained that the sequence generator (S, G, R, P) of Fig. 10(a) is neither semi-deterministic nor deterministic. It is, however, solvable, as the following considerations show. Given any sequence of P-states, divide it into a sequence (finite or infinite) of

subsequences (finite or infinite), where each subsequence is either an iteration of p_0 or an iteration of p_1 and the two types of subsequences alternate. Now a Γ-sequence $s_0, s_0, \ldots, s_0, s_1$ produces a P-state sequence $p_0, p_0, \ldots, p_0, p_0$ followed by at least one occurrence of p_1, while a Γ-sequence $s_3, s_3, \ldots, s_3, s_2$ produces a P-state sequence $p_1, p_1, \ldots, p_1, p_1$ followed by at least one occurrence of p_0. Hence for any sequence of P-states [p](0,k) one can construct a Γ-sequence [s](0,k) such that [p](0,k) = P([s](0,k)), and so (S, G, R, P) is solvable. Consider next (S, G, R, I) of Fig. 2(b). (S, G, R, I) is not semi-deterministic, since the input sequence $< i_0, i_0, i_0 >$ is the projection of both $< s_9, s_1, s_1 >$ and $< s_9, s_1, s_2 >$. But (S, G, R, I) is solvable, as may be shown by an analysis like that just given for Fig. 10(a); indeed, except for labeling, the behavior of Fig. 2(b) is the same as the behavior of Fig. 10(a).

The following lemma may be established by simple mathematical inductions with reference to the appropriate definitions.

Lemma 2.1-4: Let Γ = (S, G, R, P).

(a) If Γ is {semi-deterministic} [deterministic] (solvable), then $\Gamma\dagger$ is {semi-deterministic} [deterministic] (solvable).

(b) If every complete state of Γ is Γ-accessible, then Γ is {semi-deterministic} [deterministic] if and only if for every finite sequence of P-states [p](0,t) there exists {at most one} [exactly one] Γ-sequence [s](0,t) such that P{[s](0,t)} = [p](0,t).

(c) If Γ is deterministic, then Γ is solvable.

Other senses of semi-determinism and of determinism may be obtained by replacing every occurrence of "complete state" in the above definition of semi-determinism and determinism either by "admissible complete state" or by "accessible complete state." We will call the concepts obtained by making the latter substitution "semi-determinism$_1$" and "determinism$_1$." It may be

shown that these two concepts are equivalent to the conditions stated in the consequent of part (b) of Lemma 2.1-4. In the case of arbitrary nets determinism$_1$ becomes the determinism of Burks and Wright, 1953, p. 1359.

The process described in Section 1.2 associates with a finite automaton a sequence generator $\Gamma = (S, G, R, I, \Theta, D)$ such that (S, G, R, I) is deterministic. Conversely, given a sequence generator $\Gamma = (S, G, R, I, \Theta)$, where (S, G, R, I) is deterministic, we can define a corresponding finite automaton. Let the set of input states $\{i\}$ and the set of output states $\{e\}$ be the ranges of the projections I and Θ, respectively. The set of internal states $\{d\}$ of the automaton is a set of sets of complete states of Γ defined as follows:

$$\alpha \in \{d\} \ .\equiv. \ \alpha = G \ .\vee. \ (\exists s)(\alpha = R(s)),$$

where α ranges over non-null subsets of S. Let the initial internal state $d_0 = G$. The direct transition function τ is given by

$$\tau(i,d) = R(\iota s \mid s \in d \ \& \ I(s) = i),$$

where "$\iota s \mid \ldots$" means "the complete state s satisfying the condition "...". Finally, the output function λ of the net is defined by

$$\lambda(i,d) = \Theta(\iota s \mid s \in d \ \& \ I(s) = i).$$

We will give an example. Figure 5(a) is a deterministic sequence generator. The set of input states for the associated automaton is $\{i_0,i_1\}$ and the set of output states is $\{e_0,e_1,e_2,e_3,e_4\}$. The set of internal states consists of the sets $\{s_0,s_1\}$, $\{s_2,s_3\}$, and $\{s_3,s_4\}$, which we will call d_0,d_1, and d_2, respectively. d_0 is the initial internal state since $\{s_0,s_1\} = G$. The direct transition and output functions are given by the table below.

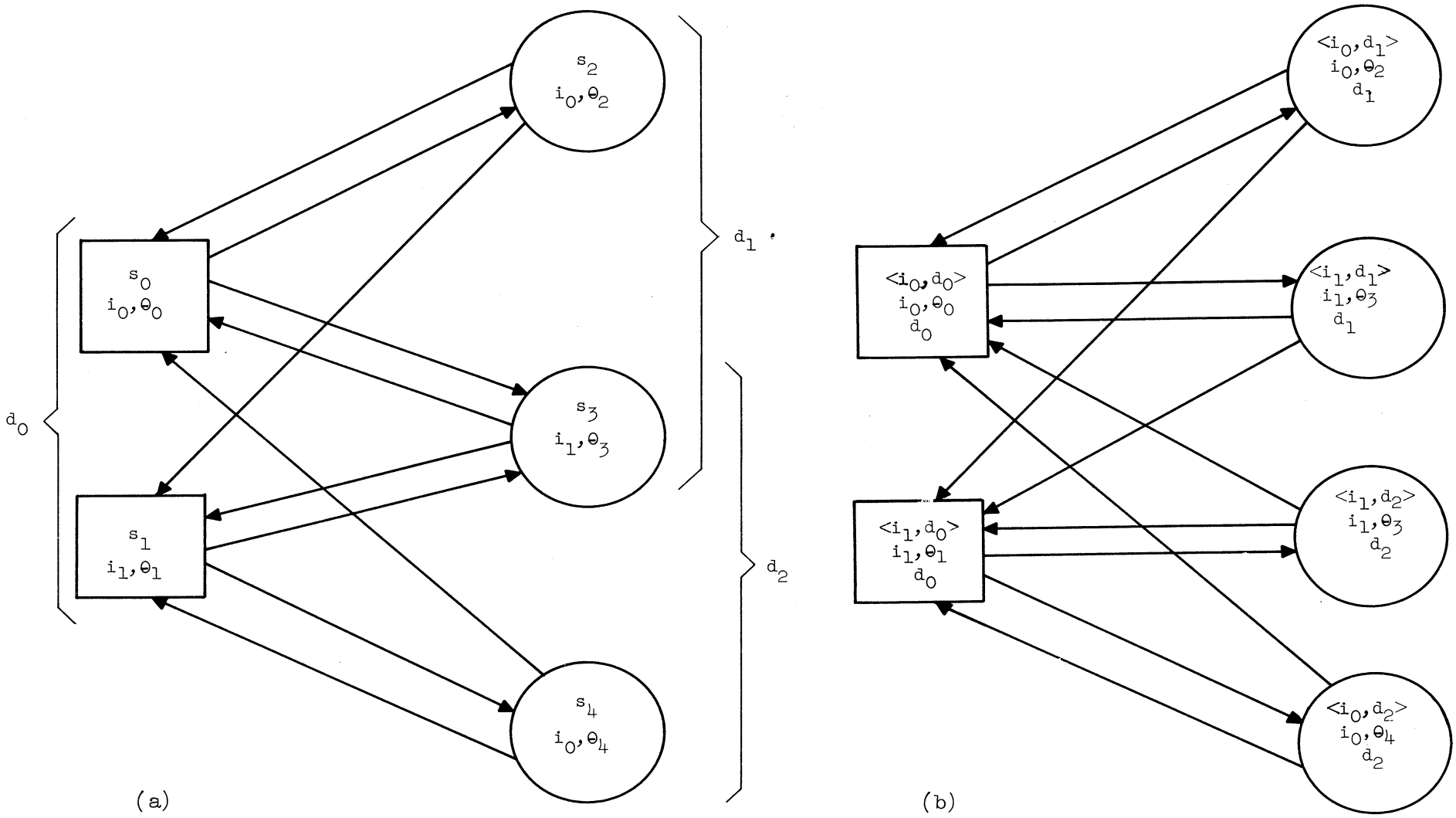


Fig. 5. Deterministic sequence generator $\Gamma = (S, G, R, I, \theta)$; (b) Internal state sequence generator $\dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{I}, \dot{\theta}, \dot{D})$ corresponding to (a).

$i(t)$	$d(t)$	$d(t+1) = \tau(i,d)$	$e(t) = \lambda(i,d)$
i_0	d_0	d_1	e_0
i_1	d_0	d_2	e_1
i_0	d_1	d_0	e_2
i_1	d_1	d_0	e_3
i_0	d_2	d_0	e_4
i_1	d_2	d_0	e_3

In Section 1.2 we gave a process for converting a finite automaton into a three-projection sequence generator. When this process is applied to the finite automaton just described, the result is the sequence generator $\dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{I}, \dot{\Theta}, \dot{D})$ of Fig. 5(b). It should be noted that $\beta(\Gamma) = \beta(\dot{S}, \dot{G}, \dot{R}, \dot{I}, \dot{\Theta})$. Hence when the two procedures just described are applied successively to a sequence generator $\Gamma = (S, G, R, I, \Theta)$, where (S, G, R, I) is deterministic, the result is a sequence generator $\dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{I}, \dot{\Theta}, \dot{D})$ with an internal state projection $\dot{D}$ and such that the behavior of $(\dot{S}, \dot{G}, \dot{R}, \dot{I}, \dot{\Theta})$ is the same as the behavior of Γ.

This is an opportune time to compare the state diagrams that we have been using, which may be called "complete state graphs," with those ordinarily used in discussing automata, which may be called "internal state graphs." The nodes of a complete state graph represent complete states and the lines represent transitions between complete states; these lines are unlabeled since the input states, output states, etc., are derived from the complete states by means of the projections. The nodes of an internal state graph represent internal states; the labeled lines represent transitions between internal states, with the labels indicating the inputs that cause the transitions and the outputs which are produced by the transitions. Since the definition of sequence generator (Section 1.1) is in terms of complete states, complete state graphs give a more direct representation of sequence generators than do internal

state graphs. We have considered definitions of sequence generators in terms of internal states, but none of these is both as general and as simple to formulate and work with as the definition we have given. However, certain kinds of sequence generators can best be analyzed in terms of internal states and are more simply represented by internal state graphs than by complete state graphs. For example, deterministic sequence generators can be analyzed in terms of internal states in the way we have just shown; moreover, the resulting internal state diagram is always simpler than the corresponding complete state diagram. Whenever the practicality of a technique of analysis is of interest and the internal state diagram is simpler than the complete state diagram, the former should, of course, be used.

Any property of a one-projection sequence generator and any operation applicable to a one-projection sequence generator can be extended to a sequence generator $\Gamma = (S, G, R)$ without projections by adjoining to it a constant projection P (i.e., a projection with only one P-state); Γ is solvable, deterministic, etc., if (S, G, R, P) is solvable, deterministic, etc. For example, a well-formed net without input nodes has associated with it (by either of the techniques of Section 1.2) a sequence generator (S, G, R) with one infinite (periodic) Γ-sequence. For constant P, (S, G, R, P) is solvable and semi-deterministic, and hence deterministic, and so is (S, G, R); see, for example, Fig. 6(a).

2.2. SUBSET SEQUENCE GENERATOR OPERATION

We will next define an operation, denoted by "*," called "the subset sequence generator operation." This operation may be applied to any sequence generator Γ to obtain its subset sequence generator Γ^*. The complete states of Γ^* are sets of complete states of Γ. The generators, the direct transition relation, and the projections of Γ^* are defined in terms of Γ in such a way

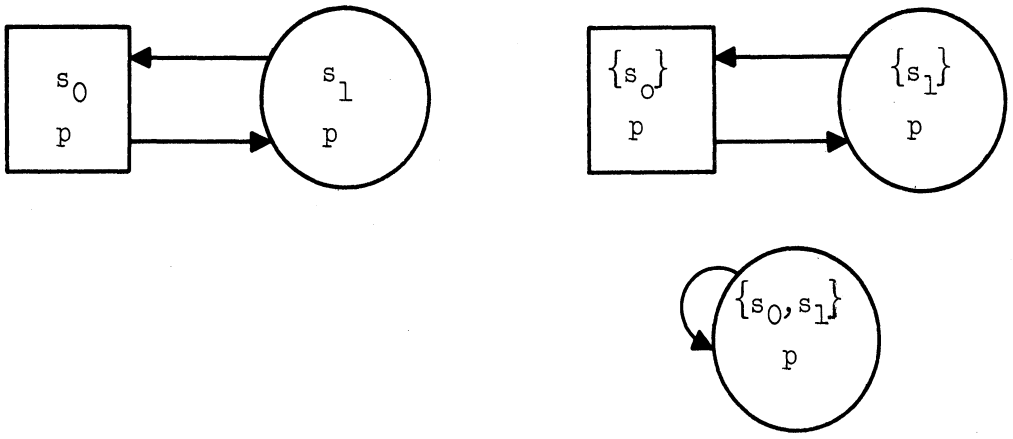


Fig. 6. The construction of a subset sequence generator. (a) $\Gamma = (S, G, R, P)$; (b) Γ^*, the subset sequence generator of Γ.

that Γ^* has the same behavior as Γ (Theorem 2.2-3 below) and Γ^* is always semi-deterministic, even though Γ may not be (Lemma 2.2-1 below).

Definition: The subset sequence generator operation, denoted by "*," applies to any sequence generator $\Gamma = (S, G, R, P^1, P^2, \ldots, P^n)$, where $n > 0$, and produces a sequence generator $\Gamma^* = \dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{P}^1, \dot{P}^2, \ldots, \dot{P}^n)$.

Let $P = P^1 \times P^2 \times \ldots \times P^n$.

(1) A subset x of S is an element of $\dot{S}$ if and only if x is non-null and P has the same value for all elements of x. This definition can be expressed symbolically as follows, where Λ is the null set, and the variable x ranges over subsets of S:

$$x \in \dot{S} \;:\equiv:\; x \neq \Lambda \;\&\; (s_1, s_2)\; \{[(s_1 \in S) \;\&\; (s_2 \in S) \;\&\; (s_1 \in x) \;\&\; (s_2 \in x) \supset [P(s_1) = P(s_2)]\}$$

(2) The elements of $\dot{G}$ are maximal subsets of G which are elements of $\dot{S}$. Formally, $\dot{s} \in \dot{G} \;:\equiv:\; \dot{s} \in \dot{S} \;\&\; \dot{s} \subset G \;\&\; (\dot{s}_1)\; \{[\dot{s}_1 \in \dot{S} \;\&\; (\dot{s} \subset \dot{s}_1 \subset G)] \supset (\dot{s} = \dot{s}_1)\}$

(3) Two complete states $\dot{s}_1$ and $\dot{s}_2$ of $\dot{S}$ stand in the direct transition relation $\dot{R}$ if and only if $\dot{s}_2$ is a maximal set of direct successors (by R) of elements of $\dot{s}_1$. Formally,

$$\dot{R}(\dot{s}_1,\dot{s}_2) \quad :\equiv: \quad \dot{s}_1 \in \dot{S} \;\&\; s_2 \subset R(\dot{s}_1) \;\&\; (\dot{s}_3)\ \{[\dot{s}_3 \in \dot{S} \;\&\; (\dot{s}_2 \subset \dot{s}_3 \subset R(\dot{s}_1))] \supset (\dot{s}_2 = \dot{s}_3)\}$$

(4) All the elements of a state $\dot{s}$ of $\dot{S}$ have the same P^i state (for $i = 1,2,\ldots,n$), and we take this common-value to be the $\dot{P}^i$-state of $\dot{s}$. Formally, $\dot{P}^i(\dot{s}) = P^i(s)$ where $s \in \dot{s}$, $\dot{s} \in \dot{S}$.

(Γ^* is called "the subset sequence generator" of Γ. Our concept of a subset sequence generator is similar to concepts used by Myhill, 1957, p. 122, Medvedev, 1958, p. 13, and Rabin and Scott, 1959, Definition 11.)

It should be noted that the concepts of behavior (Section 2.1) and subset sequence generator are essentially one projection concepts in the sense that when many projections $P^1,P^2,\ldots,P^n$ are given, the composite projection $P^1 \times P^2 \times \ldots \times P^n$ is used in the definitions of the concepts. In subsequent theorems and algorithms we will, for the sake of simplicity, usually state our results for sequence generators with one projection, since it is obvious how to extend them to the many-projection case.

The construction of subset sequence generators is illustrated in Figs. 6 and 7. Note that the generators and complete states of the subset sequence generator Γ^* are determined without reference to the direct transition relation of Γ. Figure 6 shows that even though Γ is in reduced form, Γ^* may not be, though in fact, if Γ is in reduced form, then Γ^* has no terminal states and so $\beta(\Gamma^*) = \beta(\Gamma^*\dagger)$. In Fig. 7 we begin with a semi-deterministic sequence generator, add to it in various ways to obtain three sequence generators, $\dot{\Gamma}$, $\ddot{\Gamma}$, $\dddot{\Gamma}$ which are not semi-deterministic, and then derive the subset sequence generator of each of these. All the subset sequence generators Γ^*, $\dot{\Gamma}^*$, $\ddot{\Gamma}^*$, $\dddot{\Gamma}^*$ are semi-deterministic, as they must be by the next lemma. None of the sequence

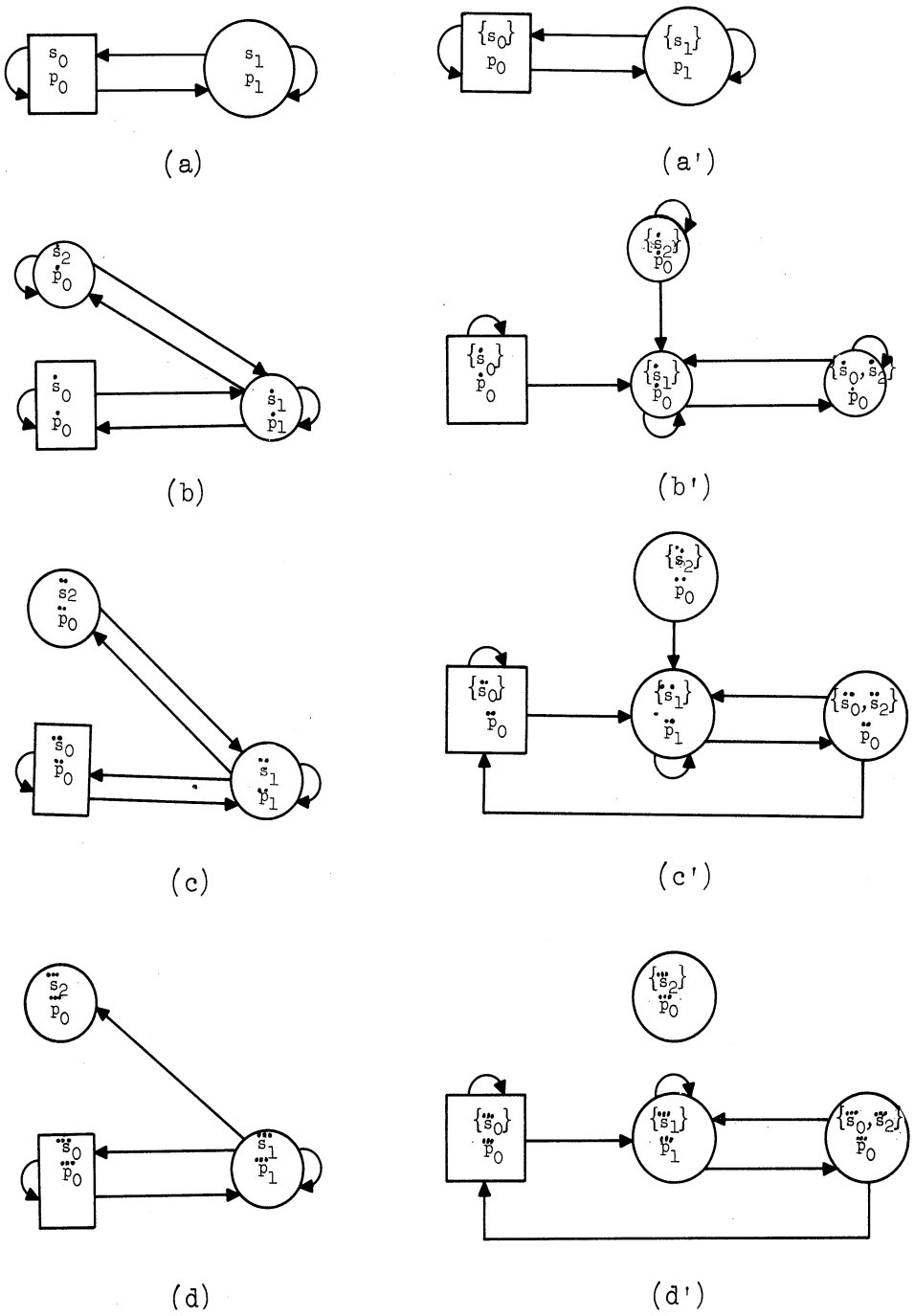


Fig. 7. Examples which illustrate Lemma 2.2-1: For any sequence generator $\Gamma = (S, G, R, P)$, Γ^* is semi-deterministic. (a) Sequence generator $\Gamma = (S, G, R, P)$. Γ is semi-deterministic. (a') Γ^*, the subset sequence generator of Γ. Γ^* is semi-deterministic. (b) Sequence generator $\dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{P})$. $\dot{\Gamma}$ is not semi-deterministic. (b') $\dot{\Gamma}^*$, the subset sequence generator of $\dot{\Gamma}$. $\dot{\Gamma}^*$ is semi-deterministic. (c) $\ddot{\Gamma} = (\ddot{S}, \ddot{G}, \ddot{R}, \ddot{P})$. $\ddot{\Gamma}$ is not semi-deterministic. (c') $\ddot{\Gamma}^*$, the subset sequence generator of $\ddot{\Gamma}$. $\ddot{\Gamma}^*$ is semi-deterministic. (d) $\dddot{\Gamma} = (\dddot{S}, \dddot{G}, \dddot{R}, \dddot{P})$. $\dddot{\Gamma}$ is not semi-deterministic. (d') $\dddot{\Gamma}^*$, the subset sequence generator of $\dddot{\Gamma}$. $\dddot{\Gamma}^*$ is semi-deterministic.

generators Γ, $\dot{\Gamma}$, $\ddot{\Gamma}$, $\dddot{\Gamma}$ is solvable; Γ^*, $\dot{\Gamma}^*$, $\ddot{\Gamma}^*$, and $\dddot{\Gamma}^*$ are not solvable either (cf. corollary 2.2-4).

Lemma 2.2-1: For any sequence generator $\Gamma = (S, G, R, P)$, Γ^* is semi-deterministic.

Proof: Let $\Gamma^* = \dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{P})$. It follows from the construction of $\dot{G}$ that for any $\dot{s}_1, \dot{s}_2$, if $\dot{s}_1 \in \dot{G}$, $\dot{s}_2 \in \dot{G}$, and $\dot{P}(\dot{s}_1) = \dot{P}(\dot{s}_2)$, then $\dot{s}_1 = \dot{s}_2$, and it follows from the definition of $\dot{R}$ that for any $\dot{s}, \dot{s}_1, \dot{s}_2$, if $\dot{R}(\dot{s}, \dot{s}_1)$, $\dot{R}(\dot{s}, \dot{s}_2)$, and $\dot{P}(\dot{s}_1) = \dot{P}(\dot{s}_2)$, then $\dot{s}_1 = \dot{s}_2$.

Given a sequence generator $\Gamma = (S, G, R, P)$, by the above lemma its subset sequence generator $\Gamma^* = \dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{P})$ is semi-deterministic. Hence for any given finite sequence of P-states $[p](0,t)$ there is at most one complete state $\dot{s}_1$ satisfying the condition that there exists a $\dot{\Gamma}$-sequence $[\dot{s}](0,t-1), \dot{s}_1$ such that $P\{[\dot{s}](0,t-1), \dot{s}_1\} = [p](0,t)$. Moreover, this state $\dot{s}_1$ is a set of states of Γ. We will use the locution "the state $\dot{s}_1 \in \dot{S}$ corresponding to $[p](0,t)$" to refer to this set of states $\dot{s}_1$ if it exists, otherwise to the null set.

Lemma 2.2-2: Let $\Gamma^* = \dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{P})$ be the subset sequence generator of $\Gamma = (S, G, R, P)$, let $[p](0,t)$ be any finite sequence of P-states, and let α be the set of states s_1 for which there exists a Γ-sequence $[s](0,t-1), s_1$ such that $P\{[s](0,t-1), s_1\} = [p](0,t)$. Then α is the state $\dot{s}_1 \in \dot{S}$ corresponding to $[p](0,t)$.

Proof: (I) We first prove by an induction on t that for any finite Γ-sequence $[s](0,t)$ there is a $\dot{\Gamma}$-sequence $[\dot{s}](0,t)$ satisfying the conditions

$$(a) \quad P\{[s](0,t)\} = \dot{P}\{[\dot{s}](0,t)\}$$

$$(b) \quad s(t) \in \dot{s}(t).$$

Initial step: given the Γ-sequence $s(0)$, it follows by the definition of $\dot{G}$ that there exists a complete state $\dot{s}(0)$ such that $s(0) \in \dot{s}(0)$ and $\dot{s}(0) \in \dot{G}$.

General step: The inductive hypothesis is that for every Γ-sequence $[s](0,k)$ there is a $\dot{\Gamma}$-sequence $[\dot{s}](0,k)$ satisfying the conditions (a) and (b). Consider any Γ-sequence $[s](0,k+1)$. By the inductive hypothesis there exists a $\dot{\Gamma}$-sequence $[\dot{s}](0,k)$ such that $[s](0,k)$ and $[\dot{s}](0,k)$ satisfy (a) and (b). Since $R(s(k), s(k+1))$ it follows by the definition of $\mathring{R}$ that there exists a complete state $\dot{s}_1$ such that $s(k+1) \in \dot{s}_1$ and $\mathring{R}(\dot{s}(k), \dot{s}_1)$. Hence $P\{s(k+1)\} = \mathring{P}(\dot{s}_1)$ and $[\dot{s}](0,k), \dot{s}_1$ is a Γ-sequence, and so $[s](0,k+1)$ and $[\dot{s}](0,k), \dot{s}_1$ satisfy conditions (a) and (b).

(II) We next prove by an induction on t that for any finite $\dot{\Gamma}$-sequence $[\dot{s}](0,t)$ and complete state $s_1 \in \dot{s}(t)$ there is a Γ-sequence $[s](0,t)$ satisfying the conditions

$$(c) \quad P\{[s](0,t)\} = \dot{P}\{[\dot{s}](0,t)\}$$

$$(d) \quad s(t) = s_1.$$

Initial step: given a $\dot{\Gamma}$-sequence $\dot{s}(0)$ and a state $s_1 \in \dot{s}(0)$, it follows by the definition of $\dot{G}$ that s_1 is the desired Γ-sequence. General step: the inductive hypothesis is that for every $\dot{\Gamma}$-sequence $[\dot{s}](0,k)$ and state $s_1 \in \dot{s}(k)$ there is a Γ-sequence $[s](0,k)$ satisfying conditions (c) and (d). Consider any $\dot{\Gamma}$-sequence $[\dot{s}](0,k+1)$ and a state $s_2 \in \dot{s}(k+1)$. By the definition $\mathring{R}$ there exists a state s_1 satisfying the conditions $s_1 \in \dot{s}(k)$ and $R(s_1,s_2)$. By the inductive hypothesis there is a Γ-sequence $[s](0,k)$ such that $P\{[s](0,k)\} = \mathring{P}\{[\dot{s}](0,k)\}$ and $s(k) = s_1$. $[s](0,k), s_2$ is the desired Γ-sequence. This completes the proof of Lemma 2.2-2.

Theorem 2.2-3: For any sequence generator $\Gamma = (S, G, R, R)$ with behavior $\beta(\Gamma)$, $\beta(\Gamma) = \beta(\Gamma^*)$.

Proof: By the preceding Lemma 2.2-2, for every finite t, $[p](0,t) \in \beta(\Gamma)$ if and only if $[p](0,t) \in \beta(\Gamma^*)$. The theorem to be proved now follows by the Infinity Theorem (2.1-3). It should be noted in this connection that the

proof of the Infinity Theorem makes implicit use of Γ^*, the subset sequence generator of Γ. In fact, the sequence $< \alpha_0, \alpha_1, \alpha_2, \ldots >$ employed in the proof is an infinite Γ^*-sequence.

Corollary 2.2-4: For any sequence generator $\Gamma = (S, G, R, P)$, Γ is solvable if and only if Γ^* is solvable.

2.3. DECISION PROCEDURES

Behavior Inclusion Procedure: Consider two sequence generators $\Gamma = (S, G, R, P)$ and $\dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{P})$, and let {a} [$\dot{a}$] be the number of states in {S} [$\dot{S}$]. Form all {Γ-sequences} [$\dot{\Gamma}$-sequences] of length $1 + a2^{\dot{a}}$ or less and form the set {α} [$\dot{\alpha}$] of their {P-projections} [$\dot{P}$-projections]. Write "yes" or "no" as $\alpha \subset \dot{\alpha}$ or not.

Theorem 2.3-1: Let A be the class of pairs of one-projection sequence generators $< \Gamma, \dot{\Gamma} >$ such that $\beta(\Gamma) \subset \beta(\dot{\Gamma})$, i.e., such that the behavior of Γ is included in that of $\dot{\Gamma}$. The Behavior Inclusion Procedure is a decision procedure for A.

Proof: (I) It is obvious that if $\beta(\Gamma) \subset \beta(\dot{\Gamma})$ then $\alpha \subset \dot{\alpha}$, i.e., that the algorithm yields "yes."

(II) We assume that $\alpha \subset \dot{\alpha}$, i.e., that the algorithm yields "yes," and prove that $\beta(\Gamma) \subset \beta(\dot{\Gamma})$. Let $\ddot{\Gamma} = (\ddot{S}, \ddot{G}, \ddot{R}, \ddot{P}) = \dot{\Gamma}^*$ and let $\ddot{\alpha}$ be the set of $\ddot{P}$-projections of $\ddot{\Gamma}$-sequences of length $1 + a2^{\dot{a}}$ or less. By Theorem 2.2-3 $\dot{\alpha} = \ddot{\alpha}$ and $\beta(\dot{\Gamma}) = \beta(\ddot{\Gamma})$, so we will assume that $\alpha \subset \ddot{\alpha}$ and prove that $\beta(\Gamma) \subset \beta(\ddot{\Gamma})$.

(IIA) Let $\{\alpha_t\}$ [$\ddot{\alpha}_t$] be the subset of sequences of $\{\beta(\Gamma)\}$ [$\beta(\ddot{\Gamma})$] of length t + 1 or less. We will now establish by induction that for every t, $\alpha_t \subset \ddot{\alpha}_t$.

Initial step: since by assumption $\alpha \subset \ddot{\alpha}$ and by definition $\alpha = \alpha_{a \cdot 2^{\dot{a}}}$ and $\ddot{\alpha} = \ddot{\alpha}_{a \cdot 2^{\dot{a}}}$, so $\alpha \subset \ddot{\alpha}$ for $k = a \cdot 2^{\dot{a}}$.

General step: we assume that $\alpha_k \subset \ddot{\alpha}_k$ for $k \geq a \cdot 2^{\dot{a}}$ and prove that $\alpha_{k+1} \subset \ddot{\alpha}_{k+1}$. Consider an arbitrary Γ-sequence $[s](0,k+1)$ for $k \geq a \cdot 2^{\dot{a}}$. Let

$$[p](0,k+1) = P\{[s](0,k+1)\}.$$

By the inductive hypothesis there exists a $\ddot{\Gamma}$-sequence $[\ddot{s}](0,k)$ such that

$$\ddot{P}\{[\ddot{s}](0,k)\} = [p](0,k).$$

We will show that there exists a complete state $\ddot{s}_0$ satisfying the conditions

$$(1) \quad \ddot{R}\{\ddot{s}(k), \ddot{s}_0\}$$
$$(2) \quad \ddot{P}(\ddot{s}_0) = p(k+1),$$

i.e., that there exists a $\ddot{\Gamma}$-sequence $< [\ddot{s}](0,k), \ddot{s}_0 >$ such that $\ddot{P}\{[\ddot{s}](0,k), \ddot{s}_0\} = [p](0,k+1)$. It follows from the nature of the subset sequence generator construction that $\ddot{\Gamma}$ has no more than $2^{\dot{a}}$ complete states, and hence the number of pairs of complete states $< s, \ddot{s} >$ is no more than $a \cdot 2^{\dot{a}}$. Since $k \geq a \cdot 2^{\dot{a}}$ there exist t_1, t_2 such that $0 \leq t_1 < t_2 \leq a2^{\dot{a}} \leq k$, $s(t_1) = s(t_2)$ and $\ddot{s}(t_1) = \ddot{s}(t_2)$. Let

$$(3) \quad [s_1](0,\ell+1) = \{[s](0,t_1), [s](t_2+1,k+1)\}$$
$$(4) \quad [\ddot{s}_1](0,\ell) = \{[\ddot{s}](0,t_1), [\ddot{s}](t_2+1,k)\}$$
$$(5) \quad [p_1](0,\ell+1) = \{[p](0,t_1), [p](t_2+1, k+1)\}$$

where $\ell = k - (t_2 - t_1)$. Note that

$$(6) \quad P\{[s_1](0,\ell+1)\} = [p_1](0,\ell+1)$$
$$(7) \quad P\{[\ddot{s}_1](0,\ell)\} = [p_1](0,\ell)$$

Because $\{s(t_1) = s(t_2)\}$ $[\ddot{s}(t_1) = \ddot{s}(t_2)]$ we have that $\{[s_1](0,\ell+1)$ is a Γ-sequence$\}$ $\left[[\ddot{s}_1](0,\ell) \text{ is a } \ddot{\Gamma}\text{-sequence}\right]$. And since $\ell + 1 \leq k$ there exists by the inductive hypothesis a $\ddot{\Gamma}$-sequence $[\ddot{s}_2](0,\ell+1)$

such that

$$(8)\quad \ddot{P}\{[\ddot{s}_2](0,\ell+1)\} = [p_1](0,\ell+1).$$

It follows from (7) and (8) that

$$(9)\quad \ddot{P}\{[\ddot{s}_2](0,\ell)\} = \ddot{P}\{[\ddot{s}_1](0,\ell)\} = [p_1](0,\ell)$$

and since $\ddot{\Gamma}$ is semi-deterministic (Lemmas 2.1-4 and 2.2-1) we have that

$$(10)\quad [\ddot{s}_2](0,\ell) = [\ddot{s}_1](0,\ell).$$

It follows from (4) that

$$(11)\quad \ddot{s}_1(\ell) = \ddot{s}(k)$$

and hence by (10)

$$(12)\quad \ddot{s}_2(\ell) = \ddot{s}(k)$$

Since $[\ddot{s}_2](0,\ell+1)$ is a $\ddot{\Gamma}$-sequence we have that $\ddot{R}\{\ddot{s}_2(\ell), \ddot{s}_2(\ell+1)\}$ and hence by (12) that

$$(13)\quad \ddot{R}\{\ddot{s}(k), \ddot{s}_2(\ell+1)\}.$$

Now by (8) and (5)

$$(14)\quad \ddot{P}\{\ddot{s}_2(\ell+1)\} = p(k+1).$$

Conditions (13) and (14) show that $\ddot{s}_2(\ell+1)$ satisfies conditions (1) and (2) and hence that $\ddot{s}_2(\ell+1)$ is the desired state $\ddot{s}_o$.

(IIB) We have shown that if $\alpha \subset \ddot{\alpha}$ then for every t, $\alpha_t \subset \ddot{\alpha}_t$. It follows by the Infinity Theorem (Theorem 2.1-3) that if $\alpha \subset \ddot{\alpha}$, then $\beta(\Gamma) \subset \beta(\ddot{\Gamma})$. As remarked earlier, this is equivalent to: if $\alpha \subset \dot{\alpha}$ then $\beta(\Gamma) \subset \beta(\dot{\Gamma})$. This completes the proof of Theorem 2.3-1.

In formulating the Behavior Inclusion Procedure we have not attempted to minimize the computation required. Many simplifications will occur to anyone who uses this algorithm. For example, since any two elements of a complete state $\dot{\dot{s}} \in \dot{\dot{S}}$ must have the same projection, the bound $1 + a \cdot 2^{\dot{a}}$ may be greatly reduced. Note also that if $\dot{\Gamma}$ is already semi-deterministic, it is not necessary to make use of $\dot{\Gamma}^*$ in the proof, and the bound $1 + a \cdot 2^{\dot{a}}$ may be replaced by $1 + a\dot{a}$.

The Behavior Inclusion Procedure may be used as the basis of a decision procedure for solvability. Let $\Gamma = (S, G, R, P)$ be given. By definition Γ is solvable if every infinite sequence of P-states belongs to its behavior (Section 2.1). $\dot{\Gamma} = (S, S, \dot{R}, P)$, where $\dot{R}(s_1, s_2)$ for all $s_1, s_2 \in S$, has as its behavior the set of all sequences of P-states. Hence the behavior of $\dot{\Gamma}$ includes all infinite sequences of P-states, so Γ is solvable if and only if $\mathcal{B}(\dot{\Gamma}) \subseteq \mathcal{B}(\Gamma)$. By Theorem 2.3-1 the Behavior Inclusion Procedure is a decision procedure for behavior inclusion, so we have proved the following theorem.

Theorem 2.3-2: Let $\Gamma = (S, G, R, P)$ be a sequence generator and let $\dot{R}(s_1, s_2)$ for all $s_1, s_2 \in S$. The Behavior Inclusion Procedure applied to the pair $< (S, S, \dot{R}, P), \Gamma >$ is a decision procedure for the solvability of Γ.

When the Behavior Inclusion Procedure is applied first to the pair $< \Gamma, \dot{\Gamma} >$ and then to the pair $< \dot{\Gamma}, \Gamma >$ the result is "yes" in both cases if and only if $\mathcal{B}(\Gamma) = \mathcal{B}(\dot{\Gamma})$. This "behavior equivalence procedure" may be used to reduce the number of complete states of a sequence generator so as to obtain a behaviorally equivalent sequence generator with fewer states. Consider $\Gamma = (S, G, R, P)$. We will say that two complete states s_1 and s_2 are "behaviorally equivalent" if

$$\mathcal{B}[(S, \{s_1\}, R, P)] = \mathcal{B}[(S, \{s_2\}, R, P)].$$

Let $\dot{\Gamma}$ be the result of identifying all behaviorally equivalent states of Γ.

$\dot{\Gamma}$ will in general have fewer states than Γ and yet $\mathcal{B}(\dot{\Gamma}) = \mathcal{B}(\Gamma)$. Moore's concept of two sequential machines being indistinguishable by any experiment is a special case of our concept of behavioral equivalence, and the above process of identifying behaviorally equivalent states is analogous to the "reduction procedure" of Moore, 1956, and Mealy, 1955. We have examples to show that the procedure we described does not always lead to a behaviorally equivalent sequence generator with a minimal number of complete states and that the procedure of Moore and Mealy does not always lead to a behaviorally equivalent sequential machine with a minimum number of internal states.*

---

* Moore and Mealy showed that identifying behaviorally equivalent states of a sequential machine does lead to a minimum number of internal states if either the sequential machine is "strongly connected" (Moore) or it has exactly one generator (Mealy). The counter-examples mentioned above are not strongly connected and have more than one generator.

3. SEQUENCE GENERATORS WITH TWO PROJECTIONS

3.1. DEFINITIONS

The results of the last section concern primarily one projection of a sequence generator. In the present section we will work mainly with two-projection sequence generators.

Definition: $\Gamma = (S, G, R, P, Q)$ is h-univalent ($h = 0, 1, 2, \ldots ; \omega$) if for every two infinite Γ-sequences $[s_1](0,\omega)$, $[s_2](0,\omega)$ and any time t, if $P([s_1](0,t + h)) = P([s_2](0,t + h))$ then $Q(s_1(t)) = Q(s_2(t))$. (By definition, $t + \omega = \omega$.)

Note that h-univalence is essentially a property of a set of infinite sequences of pairs $< p,q >$, and hence a property of $\mathcal{B}^{\omega}(\Gamma)$, the infinite behavior of Γ. As a consequence the following lemma holds.

Lemma 3.1-1: Let $\Gamma = (S, G, R, P, Q)$ and $\dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{P}, \dot{Q})$. If $\mathcal{B}^{\omega}(\Gamma) = \mathcal{B}^{\omega}(\dot{\Gamma})$, then Γ is h-univalent if and only if $\dot{\Gamma}$ is h-univalent. This lemma, together with Corollary 2.1-1 and Theorem 2.2-3, immediately yields

Lemma 3.1-2: Let $\Gamma = (S, G, R, P, Q)$. The following three conditions are equivalent: (1) Γ is h-univalent, (2) Γ^* is h-univalent, and (3) $\Gamma^{\dagger}$ is h-univalent.

There is a close connection between zero-univalence and semi-determinism which is brought out by the following lemma.

Lemma 3.1-3: (a) Let $\Gamma = (S,G,R,P,Q)$ and $\Gamma = \Gamma\dagger$. Γ is 0-univalent if and only if for any two finite Γ-sequences $[s_1](0,t)$ and $[s_2](0,t)$, if $P([s_1](0,t)) = P([s_2](0,t))$ then $Q([s_1](0,t)) = Q([s_2](0,t))$. (b) Let $\Gamma = (S,G,R,P)$ and $\Gamma = \Gamma\dagger$. Γ is semi-deterministic if and only if for any two finite Γ-sequences, $[s_1](0,t)$ and $[s_2](0,t)$, if $P([s_1](0,t)) = P([s_2](0,t))$ then $[s_1](0,t) = [s_2](0,t)$.

Note that the sequence generator of part (a) of the lemma has two projections, while that of part (b) has one projection. Part (a) may be established by using Corollary 1.3-3 and the definition of univalence; part (b) follows from Corollary 1.3-3 and Lemma 2.1-4b. It follows from Lemma 3.1-3 that for any projection Q, if (S,G,R,P) is semi-deterministic then (S,G,R,P,Q) is 0-univalent. The converse is not in general true, but the following lemma asserts a connection between the 0-univalence of a sequence generator and the semi-determinism of a related sequence generator.

Lemma 3.1-4: Let $\Gamma = (S,G,R,P,Q)$ and let $\Gamma^{*\dagger} = \dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{P}, \dot{Q})$. Γ is zero-univalent if and only if $(\dot{S},\dot{G},\dot{R},\dot{P})$ is semi-deterministic.

It might seem that since $\dot{\Gamma}$ is the reduced form of the subset sequence generator of Γ, it would follow immediately by Lemma 2.2-1 that if Γ is 0-univalent then $(\dot{S},\dot{G},\dot{R},\dot{P})$ is semi-deterministic. This is by no means the case. It can be shown from Lemma 2.2-1 by means of the definition of the subset sequence generator operation that $(\dot{S},\dot{G},\dot{R},\dot{P}x\dot{Q})$ is semi-deterministic, while the conclusion of Lemma 3.1-4 is that $(\dot{S},\dot{G},\dot{R},\dot{P})$, which is a different sequence generator, is semi-deterministic. For any projection $\dot{Q}$, if a sequence generator (S,G,R,P) is semi-deterministic, then $(\dot{S},\dot{G},\dot{R},\dot{P}x\dot{Q})$ is semi-deterministic, but the converse is not in general true.

Proof of Lemma 3.1-4 ("Only if" part): We assume that Γ is 0-univalent and prove that $(\dot{S},\dot{G},\dot{R},\dot{P})$ is semi-deterministic.

(I) We will use three sequence generators in the proof besides Γ. These are $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$, $\ddot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P}x\dot{Q})$, and $\dddot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P})$. We will first establish some results that will enable us to use Lemma 3.1-3a on $\dot{\Gamma}$ and Lemma 3.1-3b on $\ddot{\Gamma}$ and $\dddot{\Gamma}$. (A) By construction $\dot{\Gamma} = \dot{\Gamma}\dagger$ and by Lemma 3.1-2 $\dot{\Gamma}$ is 0-univalent.

(B) By construction $\ddot{\Gamma} = \ddot{\Gamma}\dagger$ and by Lemma 2.1-1 $\ddot{\Gamma}$ is semi-deterministic.

(C) By construction $\dddot{\Gamma} = \dddot{\Gamma}\dagger$. Our task is to prove that $\dddot{\Gamma}$ is semi-deterministic.

(II) Since $\dot{\Gamma}$, $\ddot{\Gamma}$, and $\dddot{\Gamma}$ have $\dot{S}$, $\dot{G}$, $\dot{R}$ in common, the sets of $\dot{\Gamma}$-sequences,

$\ddot{\Gamma}$-sequences, and $\dddot{\Gamma}$-sequences are identical with one another. Consider now any two finite $\dot{\Gamma}$-sequences $[\dot{s}_1](0,t)$ and $[\dot{s}_2](0,t)$; these are also arbitrary $\ddot{\Gamma}$-sequences and arbitrary $\dddot{\Gamma}$-sequences. Using (IA) and applying Lemma 3.1-3a to $\dot{\Gamma}$, we obtain

(1) If $\dot{P}([\dot{s}_1](0,t)) = \dot{P}([\dot{s}_2](0,t))$, then $\dot{Q}([\dot{s}_1](0,t)) = \dot{Q}([\dot{s}_2](0,t))$.

Using (IB) and applying Lemma 3.1-3b to $\ddot{\Gamma}$, we obtain

(2) If $\dot{P}([\dot{s}_1](0,t)) = \dot{P}([\dot{s}_2](0,t))$ and $\dot{Q}([\dot{s}_1](0,t)) = \dot{Q}([\dot{s}_2](0,t))$, then $[\dot{s}_1](0,t) = [\dot{s}_2](0,t)$.

Combining (1) and (2) and noting that $[\dot{s}_1](0,t)$ and $[\dot{s}_2](0,t)$ are arbitrary $\dddot{\Gamma}$-sequences, we get

(3) For any two finite $\dddot{\Gamma}$-sequences $[\dot{s}_1](0,t)$ and $[\dot{s}_2](0,t)$, if $\dot{P}([\dot{s}_1](0,t)) = \dot{P}([\dot{s}_2](0,t))$, then $[\dot{s}_1](0,t) = [\dot{s}_2](0,t)$.

Using (3) and (IC) and applying Lemma 3.1-3b to $\ddddot{\Gamma}$, we obtain

(4) $\ddddot{\Gamma}$ is semi-deterministic,

which completes the proof of the "only if" part of the Lemma.

("If" part): We assume that $(\dot{S},\dot{G},\dot{R},\dot{P})$ is semi-deterministic and prove that Γ is 0-univalent. Since every complete state of $\dot{\Gamma}$ is $\dot{\Gamma}$-accessible, by Lemma 2.1-4b we have that for every finite sequence of $\dot{P}$-states $[\dot{p}](0,t)$ there exists at most one $\dot{\Gamma}$-sequence $[\dot{s}](0,t)$ such that $\dot{P}\{[\dot{s}](0,t)\} = [\dot{p}](0,t)$. Hence for any two $\dot{\Gamma}$-sequences $[\dot{s}_1](0,\omega)$, $[\dot{s}_2](0,\omega)$ and any time t, if $\dot{P}([\dot{s}_1](0,t)) = \dot{P}([\dot{s}_2](0,t))$, then $\dot{s}_1(t) = \dot{s}_2(t)$. Since a projection is a (single-valued) function, we have that if $\dot{P}([\dot{s}_1](0, t + 0)) = \dot{P}([\dot{s}_2](0,t + 0))$ then $\dot{Q}(\dot{s}_1(t)) = \dot{Q}(\dot{s}_2(t))$, so $\dot{\Gamma}$ is 0-univalent. $\dot{\Gamma} = \Gamma^{*\dagger}$, and by Lemma 3.1-2 Γ is 0-univalent. This completes the proof of Lemma 3.1-4.

We next apply this lemma to an example. Consider $\Gamma = (S,G,R,P,Q)$ of Fig. 8(a). Note that the complete states s_2 and s_4 have the same projections (p_o and q_o) and stand in the same relation to state s_o. Thus the two Γ-sequences

$$s_o, s_4, s_o$$

$$s_o, s_2, s_o$$

have the same sequence of P-projections

$$p_o, p_o, p_o$$

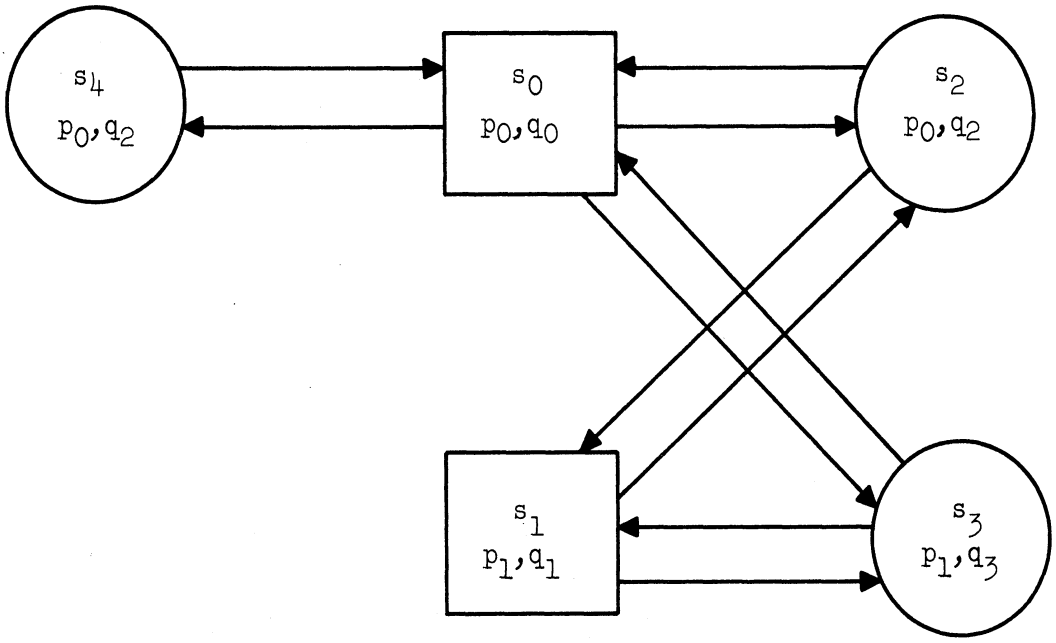
and hence (S,G,R,P) is not semi-deterministic. These two Γ-sequences do have the same sequence of Q-Projections

$$q_o, q_2, q_o$$

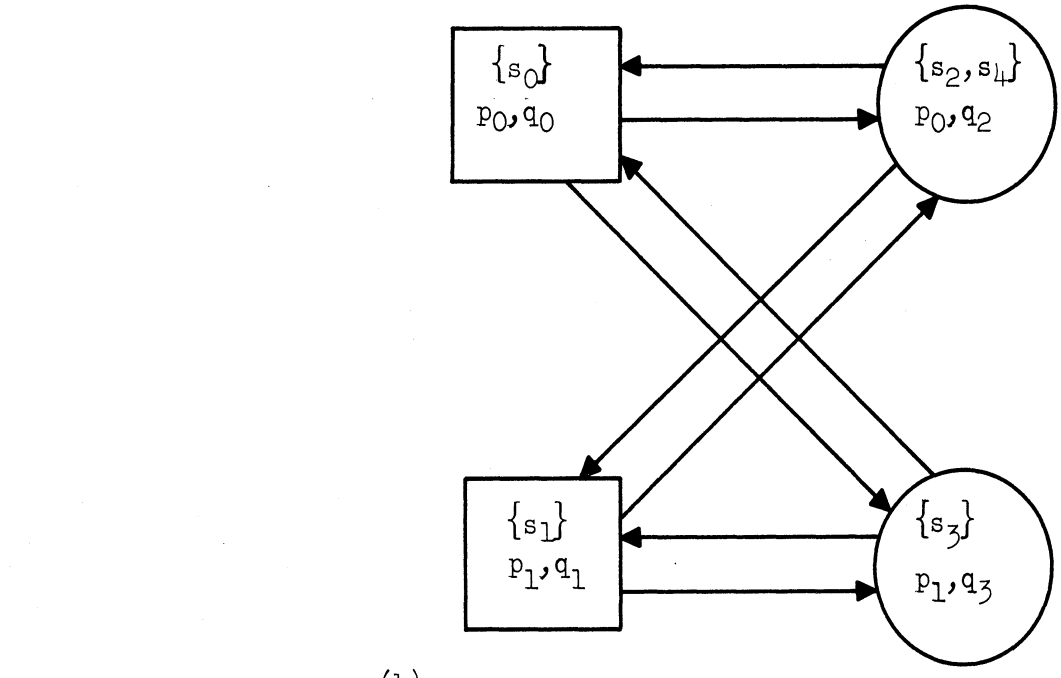
and in fact Γ is 0-univalent. By Lemma 3.1-4 $(\dot{S}, \dot{G}, \dot{R}, \dot{P})$ of Fig. 8(b) must be semi-deterministic. An examination of the states $(\dot{S}, \dot{G}, \dot{R}, \dot{P})$ shows that it is deterministic, so <u>a fortiori</u> it is semi-deterministic. (The determinism of $(\dot{S}, \dot{G}, \dot{R}, \dot{P})$ will be discussed after Lemma 3.2-3 below.) Note that the main difference between Γ and $\Gamma^{*\dagger}$ in Fig. 8 is that the two states s_2, s_4 of Γ have become a single state $\{s_2, s_4\}$ of $\Gamma^{*\dagger}$.

<u>Definition</u>: $\Gamma = (S, G, R, P, Q)$ is <u>uniquely solvable</u> if (1) (S,G, R, P) is solvable and (2) (S, G, R, P, Q) is ω-univalent. We remarked earlier that h-univalence is essentially a property of the infinite behavior of a sequence generator, and this remark applies to unique solvability as well. Thus Γ is uniquely solvable if and only if for any infinite sequence of P-states $[p](0,\omega)$ there is exactly one sequence of Q-states $[q](0,\omega)$ such that the sequence

$$< p(0), q(0) >, < p(1), q(1) >, \ldots$$



(a)



(b)

Fig. 8. Illustration of Lemma 3.1-4: Γ is zero-univalent if and only if $(\dot{S}, \dot{G}, \dot{R}, \dot{P})$ is semi-deterministic. (a) $\Gamma = (S, G, R, P, Q)$. Γ is zero-univalent and uniquely solvable, but (S,G,R,P) is not semi-deterministic. (b) $\Gamma^{*\dagger} = \dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{P}, \dot{Q})$. $(\dot{S}, \dot{G}, \dot{R}, \dot{P})$ is deterministic, and a fortiori semi-deterministic.

belongs to $\beta(\Gamma)$. To put the point in another way: a sequence generator $\Gamma = (S, G, R, P, Q)$ is uniquely solvable if and only if its behavior defines a single-valued function (transformation) from the set of all infinite sequences of P-states into the set of all infinite sequences of Q-states. Various consequences follow from this fact. The result of replacing "h-univalence" by "uniquely solvable" in Lemma 3.1-1 is also a lemma. A similar remark holds for Lemma 3.1-2 except that $\Gamma\dagger$ may have fewer P-states (values of p) than Γ.

It was shown in Section 2.1 that well-formed nets and deterministic sequence generators are equivalent in a certain sense: for every w.f.n. there is a corresponding deterministic sequence generator and vice-versa. The w.f.n. gives the structure of an automaton while the associated deterministic sequence generator gives the corresponding complete state diagram. An analogous relation holds between the well-behaved nets of Burks and Wright, 1953, p. 1358, and uniquely solvable sequence generators. Consider any net and label all its non-input nodes as output nodes. The procedure of Section 1.2 will associate with this net a sequence generator which is uniquely solvable if and only if the original net is well-behaved.

3.2. THE DISPLACEMENT OPERATOR AND THE ℓ-SHIFT OPERATION

We will first define a displacement operator $\mathscr{D}^k$ which applies to sets composed of finite sequences of pairs and/or ω-sequences of pairs. Roughly speaking, $\mathscr{D}^k$ has the effect of leaving the first element of each pair where it is and displacing the second element of each pair k places to the right. Displacing the second element of the first pair k places to the right will leave k gaps, since the first pair is not preceded by any pair. It will be convenient always to fill these gaps with the same element; we will use the null set Λ for this purpose.

Definition: Let the universe of discourse V consist of all finite sequences of pairs and all ω-sequences of pairs and let Λ be the null set. The operator $\mathcal{D}$ (without superscript) is defined to apply to any sequence of V as follows:

$$\mathcal{D}(\langle x_0, y_0\rangle, \langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \langle x_3, y_3\rangle, \ldots) =$$
$$(\langle x_0, \Lambda\rangle, \langle x_1, y_0\rangle, \langle x_2, y_1\rangle, \langle x_3, y_2\rangle, \ldots)$$
$$\mathcal{D}(\langle x_0, y_0\rangle, \langle x_1, y_1\rangle, \ldots, \langle x_{n-1}, y_{n-1}\rangle, \langle x_n, y_n\rangle) =$$
$$(\langle x_0, \Lambda\rangle, \langle x_1, y_0\rangle, \ldots, \langle x_{n-1}, y_{n-2}\rangle, \langle x_n, y_{n-1}\rangle).$$

The operator $\mathcal{D}$ is extended to apply to an arbitrary set α of V by

$$\mathcal{D}(\alpha) = \{v \mid (\exists u)\ [u \in \alpha\ \&\ v = \mathcal{D}(u)]\},$$

where v and u range over elements of V. Finally, we define $\mathcal{D}^k$, k = 0,1,2..., to apply to an arbitrary set α of V by the induction

$$\mathcal{D}^0(\alpha) = \alpha$$
$$\mathcal{D}^{i+1}(\alpha) = \mathcal{D}(\mathcal{D}^i(\alpha))$$

$\mathcal{D}^\ell$ is called the displacement operator.

We next define a shifting operator which may be applied to an arbitrary sequence generator $\Gamma = (S, G, R, P, Q)$ to produce the ℓ-shifted sequence generator $\Gamma^\ell = \dot{\Gamma} = (\dot{S}, \dot{G}, \dot{R}, \dot{P}, \dot{Q})$. The effect of this operation is to displace the behavior of Γ, so that the behavior of Γ^ℓ, i.e., $\beta(\Gamma^\ell)$, equals the displaced behavior of Γ, i.e., $\mathcal{D}^\ell[\beta(\Gamma)]$, as is shown in Lemma 3.2-1 below. To help make clear the definition of Γ^ℓ, we will make some remarks about Γ^1. Extend Q to apply to Λ, so that $Q(\Lambda) = \Lambda$. The generators of Γ^1 are the pairs $\langle \Lambda, s\rangle$, where s belongs to G. Suppose

$$s_0, s_1, s_2, s_3, s_4$$

is a Γ-sequence with the resulting behavior element

$$< p_0, q_0 >, < p_1, q_1 >, < p_2, q_2 >, < p_3, q_3 >, < p_4, q_4 >.$$

Then

$$< \Lambda, s_0 >, < s_0, s_1 >, < s_1, s_2 >, < s_2, s_3 >, < s_3, s_4 >$$

is the corresponding Γ^1-sequence with the resulting behavior element

$$< p_0, \Lambda >, < p_1, q_0 >, < p_2, q_1 >, < p_3, q_3 >, < p_4, q_3 >.$$

Γ^ℓ may be obtained by shifting Γ ℓ times in this way.

Definition: The unit-shift operation, denoted by "$\diamond$," applies to any sequence generator $\Gamma = (S,G,R,P,Q)$ and produces a sequence generator $\Gamma^\diamond = \dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$ defined as follows. (We wish to assume throughout that Λ is not an element of S; if it is, S should first be redefined so it is not.) The elements of $\dot{G}$ are all the pairs $< \Lambda, s >$ where s belongs to G:

$$< \Lambda, s > \epsilon\, \dot{G} \equiv s \epsilon G.$$

(2) The elements of $\dot{S}$ are all the pairs $< s_1, s_2 >$ which either belong to $\dot{G}$ or are connected by the direct transition relation R:

$$\dot{S} = \{< s_1, s_2 > | < s_1, s_2 > \epsilon\, \dot{G} \vee R(s_1, s_2)\}$$

(3) Two complete states $< s_1, s_2 >$ and $< s_3, s_4 >$ of $\dot{S}$ stand in the direct transition relation $\dot{R}$ if and only if $s_2 = s_3$:

$$\dot{R}(< s_1, s_2 >, < s_3, s_4 >) \equiv [< s_1, s_2 >, < s_3, s_4 > \epsilon\, \dot{S} \;\&\; s_2 = s_3].$$

(4) The $\dot{P}$-projection of a complete state $< s_1, s_2 >$ of $\dot{S}$ is the P-projection of its second element s_2:

$$\dot{P}(< s_1, s_2 >) = P(s_2), \text{ where } < s_1, s_2 > \epsilon\, \dot{S}.$$

(5) Extend Q to apply to Λ, stipulating that $Q(\Lambda) = \Lambda$. The $\dot{Q}$-projection of the complete state $< s_1, s_2 >$ of $\dot{S}$ is the Q-projection of its <u>first</u> element:

$$\dot{Q}(< s_1, s_2 >) = Q(s_1), \text{ where } < s_1, s_2 > \epsilon \dot{S}.$$

The <u>ℓ-shift operation</u>, denoted by "ℓ," applies to any sequence generator $\Gamma = (S,G,R,P,Q)$ and produces a sequence generator Γ^{ℓ}; it is defined in terms of the unit-shift operation by means of an induction:

$$\Gamma^{0} = \Gamma$$
$$\Gamma^{\ell+1} = (\Gamma^{\ell})^{\diamond} .$$

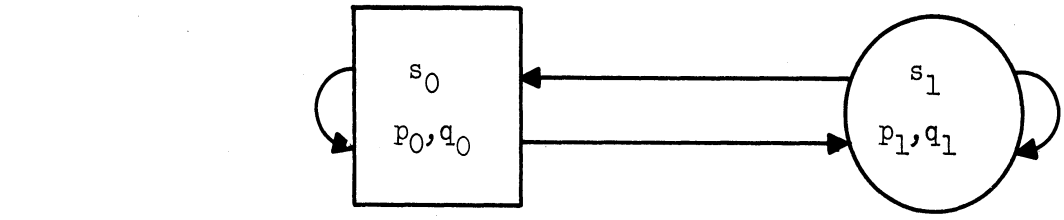
The ℓ-shift construction is illustrated in Fig. 9. Part (a) shows a sequence generator Γ with two projections, while part (b) shows Γ^1, the result of shifting Γ one unit of time. Note that both Γ and Γ^1 are in reduced form; this is a special case of the general fact that if Γ is in reduced form then Γ^h is in reduced form. Note next that the generator $< \Lambda, s_0 >$ can only occur as the first state of a Γ^1-sequence. Since Γ^{ℓ} is defined by induction on $\Gamma^{\diamond}$, it follows by induction that if a Γ^{ℓ}-admissible complete state occurs in some Γ^{ℓ}-sequence at a time $\tau < \ell$ then all occurrences of this state are at time τ. Thus the Γ^{ℓ}-admissible complete states are partitioned into two sets, those that occur only before time ℓ, and those that occur only at time ℓ or later.

The ℓ-shift operation is of interest because it shifts behavior in the same way the displacement operator $\mathcal{D}^{\ell}$ does. Compare the behaviors of Γ and Γ^1. The behavior element [an element of $\mathcal{B}(\Gamma)$]

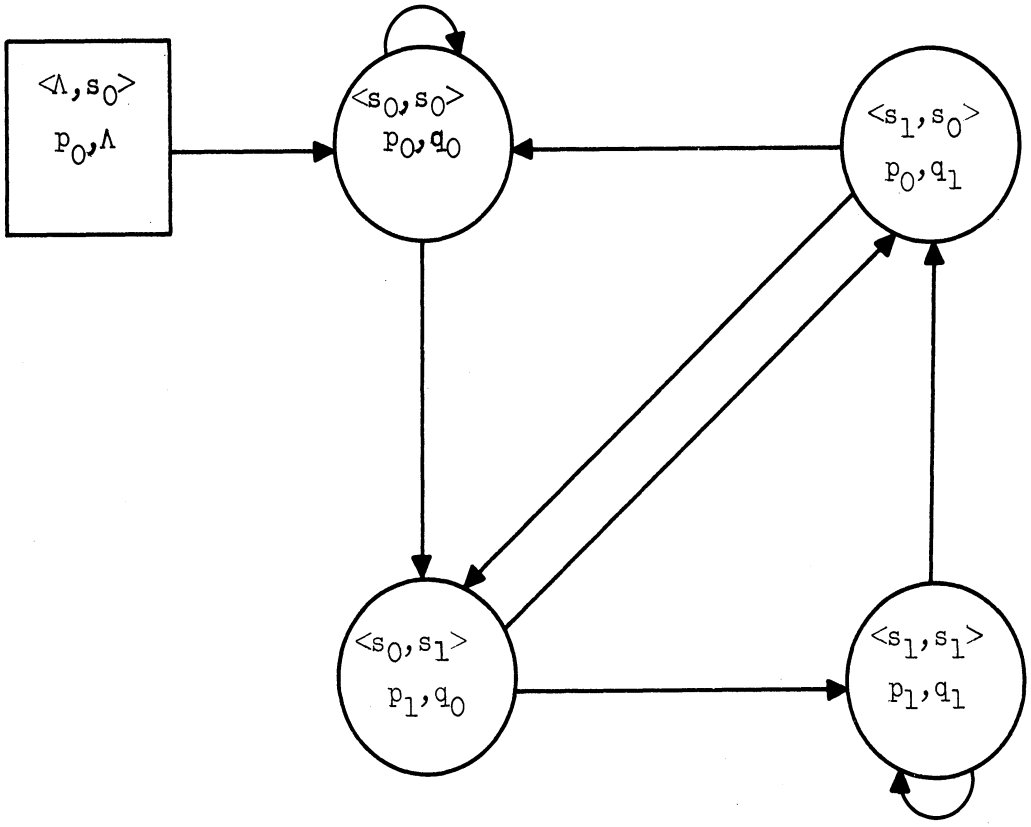
(1) $< p_0, q_0 >, < p_0, q_0 >, < p_1, q_1 >, < p_0, q_0 >$

is derived from the Γ-sequence

$$s_0, s_0, s_1, s_1, s_0.$$



(a)



(b)

Fig. 9. Illustration of the ℓ-shift construction, for $\ell = 1$. In accordance with Lemma 3.2-1, $\mathcal{D}^1[\mathcal{B}(\Gamma)] = \mathcal{B}(\Gamma^1)$. (a) $\Gamma = (S, G, R, P, Q)$; (b) Γ^1 (unit-shifted sequence generator of Γ).

The corresponding Γ^1-sequence is

$$< \Lambda, s_0 >, < s_0, s_0 >, < s_0, s_1 >, < s_1, s_1 >, < s_1, s_0 >,$$

which gives rise to the behavior element [an element of $\mathcal{B}(\Gamma^1)$]

$$(2) \quad < p_0, \Lambda >, < p_0, q_0 >, < p_1, q_0 >, < p_1, q_1 >, < p_0, q_1 >.$$

Note that this last sequence (2) is the result of displacing sequence (1) by one unit. This is an example of the general fact that $\mathcal{B}(\Gamma^1) = \mathcal{D}^1[\mathcal{B}(\Gamma)]$, which is a special case of the following lemma.

Lemma 3.2-1: Let $\Gamma = (S,G,R,P,Q)$. Then

$$(a) \quad \mathcal{D}^\ell[\mathcal{B}(\Gamma)] = \mathcal{B}(\Gamma^\ell)$$
$$(b) \quad \mathcal{D}^\ell[\mathcal{B}^\omega(\Gamma)] = \mathcal{B}^\omega(\Gamma^\ell).$$

Proof: (IA) We prove first that $\mathcal{D}[\mathcal{B}^\omega(\Gamma)] = \mathcal{B}^\omega(\Gamma^{\diamond})$. Let $\Gamma^{\diamond} = \dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$. It follows from the definition of the unit shift operator that there is a one-one correspondence between the set of infinite Γ-sequences and the set of infinite $\dot{\Gamma}$-sequences with corresponding sequences being of the form

$$(1) \quad s_0, s_1, s_2, s_3, \ldots .$$

$$(2) \quad < \Lambda, s_0 >, < s_0, s_1 >, < s_1, s_2 >, < s_2, s_3 >, \ldots .$$

When $P \times Q$ is applied to (1), we get

$$(3) \quad < p_0, q_0 >, < p_1, q_1 >, < p_2, q_2 >, < p_3, q_3 >, \ldots$$

as an element of $\mathcal{B}^\omega(\Gamma)$, and when $\dot{P} \times \dot{Q}$ is applied to (2), we get

$$(4) \quad < p_0, \Lambda >, < p_1, q_0 >, < p_2, q_1 >, < p_3, q_2 >, \ldots$$

as an element of $\mathcal{B}^\omega(\Gamma)$. By definition of $\mathcal{D}$,

$$\mathcal{D}[(3)] = (4).$$

Hence

$$\mathcal{D}[\mathcal{B}^{\omega}(\Gamma)] = \mathcal{B}^{\omega}(\Gamma^{\diamond}).$$

(IB) Applying mathematical induction to result (IA), we get

$$\mathcal{D}^{\ell}[\mathcal{B}^{\omega}(\Gamma)] = \mathcal{B}^{\omega}(\Gamma^{\ell})].$$

(IIA) An argument similar to that of (IA) may be given for finite Γ-sequences and finite $\Gamma^{\diamond}$-sequences. When the result is combined with result (IA), we get

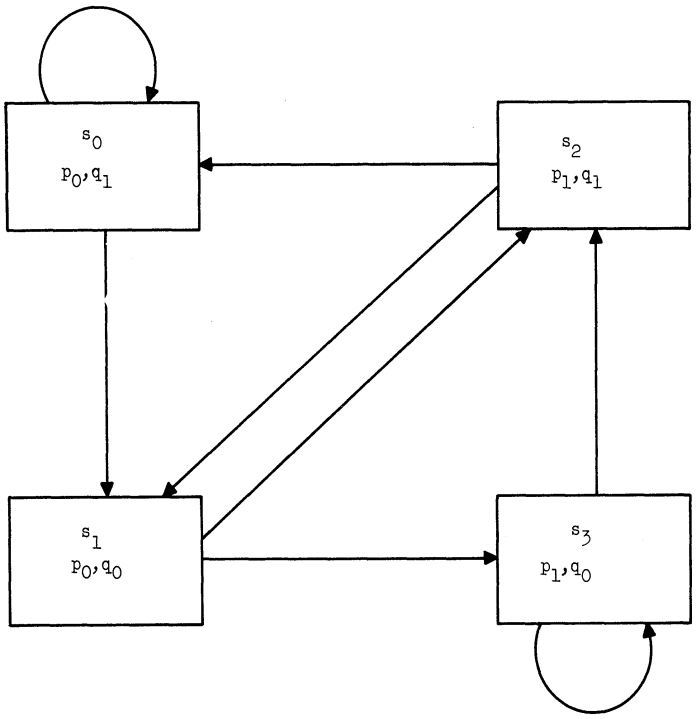
$$\mathcal{D}[\mathcal{B}(\Gamma)] = \mathcal{B}(\Gamma^{\diamond}).$$

(IIB) Applying mathematical induction to (IIA), we get

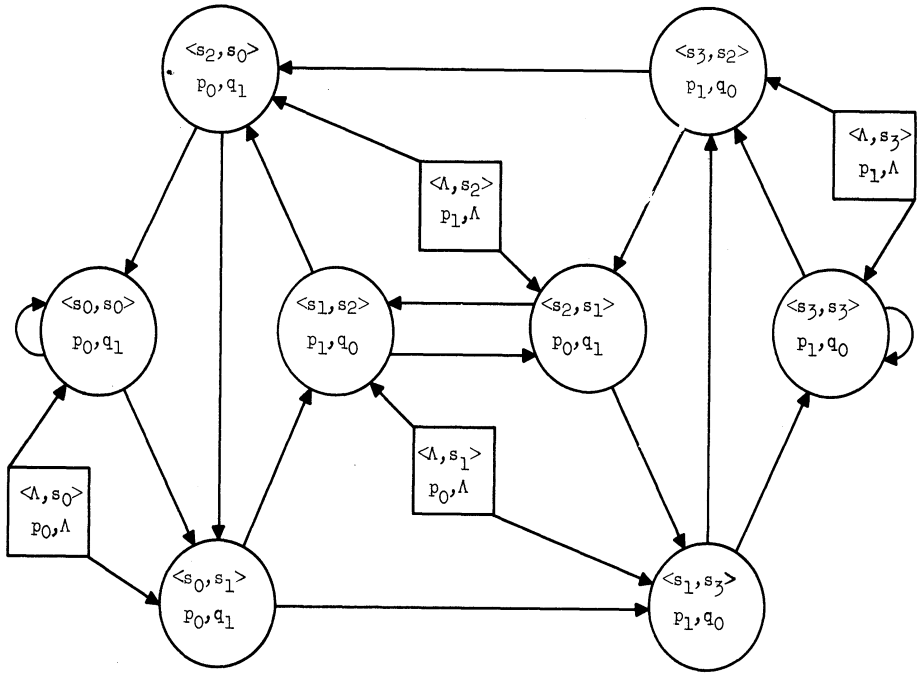
$$\mathcal{D}^{\ell}[\mathcal{B}(\Gamma)] = \mathcal{B}(\Gamma^{\ell}).$$

Corollary 3.2-2: Let $\Gamma = (S,G,R,P,Q)$ and $\Gamma^{\ell} = \dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$.
[Γ is $(\ell + h)$-univalent] {(S,G,R,P) is solvable} (Γ is uniquely solvable) if and only if [Γ^{ℓ} is h-univalent] {$(\dot{S},\dot{G},\dot{R},\dot{P})$ is solvable} (Γ^{ℓ} is uniquely solvable).

This corollary is illustrated by Fig. 10. Consider Γ of this figure. It was shown in Section 2.1 (in the paragraph preceding Lemma 2.1-4) that (S, G, R, P) is solvable. Also, Γ is unit-univalent. To see this, observe that every immediate successor (by the direct transition relation R) of a given complete state s has the same P-projection; e.g., $R(s_0) = \{s_0,s_1\}$ and $P(s_0) = P(s_1) = p_0$. Since (S, G, R, P) is solvable and Γ is unit-univalent, Γ is uniquely solvable. Turn now to Γ^1, the unit-shifted sequence generator of Γ. Since (S, G, R, P) is solvable and Γ is unit-univalent and uniquely solvable, by Corollary 3.2-2 Γ^1 (less its last projection) must be solvable, and Γ^1 must be zero-univalent and also uniquely solvable.



(a)



(b)

Fig. 10. Illustration of Corollary 3.2-2. [Γ is 1-univalent] {Γ, less its last projection, is solvable} (Γ is uniquely solvable) if and only if [Γ^1 is 0-univalent] {Γ^1, less its last projection, is solvable} (Γ^1 is uniquely solvable). (a) $\Gamma = (S, G, R, P, Q)$. (S, G, R, P) is solvable but not deterministic. Γ is unit-univalent and uniquely solvable. (b) $\overline{\Gamma^1}$, the unit-shifted sequence generator of Γ. Γ^1, less its last projection, is solvable, but not deterministic. Γ^1, is zero-univalent and uniquely solvable.

Lemma 3.2-3: Let $\Gamma = (S,G,R,P,Q)$ and $\Gamma^{h*\dagger} = \dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$. Then (a) if Γ is h-univalent for some finite h and (S,G,R,P) is solvable, then $(\dot{S},\dot{G},\dot{R},\dot{P})$ is deterministic (b) $\mathscr{D}^h[\mathcal{B}^\omega(\Gamma)] = \mathcal{B}^\omega(\dot{\Gamma})$.

Proof: (IA) We will prove first that $(\dot{S},\dot{G},\dot{R},\dot{P})$ is solvable. It is given that (S,G,R,P) is solvable. By Corollary 3.2-2, Lemma 2.2-4, and Lemma 2.1-4a, $(\dot{S},\dot{G},\dot{R},\dot{P})$ is solvable. (IB) We prove next that $(\dot{S},\dot{G},\dot{R},\dot{P})$ is semi-deterministic. It is given that Γ is h-univalent. By Corollary 3.2-2, Γ^h is 0-univalent. By Lemma 3.1-4, $(\dot{S},\dot{G},\dot{R},\dot{P})$ is semi-deterministic. (IC) It follows from Lemma 2.1-4b and the definition of solvable that if every complete state of any sequence generator $\Gamma = (S,G,R,P)$ is Γ-accessible, then Γ is deterministic if and only if Γ is both semi-deterministic and solvable. Applying this principle to $(\dot{S},\dot{G},\dot{R},\dot{P})$ and using (IA) and (IB), we conclude that $(\dot{S},\dot{G},\dot{R},\dot{P})$ is deterministic. This proves part (a) of the lemma.

(II) By Lemma 3.2-1b

(1) $\mathscr{D}^h[\mathcal{B}^\omega(\Gamma)] = \mathcal{B}^\omega(\Gamma^h)$.

By Theorem 2.2-3 $\mathcal{B}(\Gamma^h) = \mathcal{B}(\Gamma^{h*})$ and hence

(2) $\mathcal{B}^\omega(\Gamma^h) = \mathcal{B}^\omega(\Gamma^{h*})$.

But by Corollary 2.1-1

(3) $\mathcal{B}^\omega(\Gamma^{h*}) = \mathcal{B}^\omega(\Gamma^{h*\dagger})$.

Combining (1), (2), and (3) gives part (b) of Lemma 3.2-3 and completes the proof of the present lemma.

We may apply this lemma to Fig. 8. As noted in Section 3.1, (S,G,R,P) is not semi-deterministic but (S,G,R,P,Q) is 0-univalent. It is easy to see that (S,G,R,P) is solvable. Applying Lemma 3.2-3 with h = 0, we conclude that $(\dot{S},\dot{G},\dot{R},\dot{P})$ is deterministic and that $\mathcal{B}^\omega(\Gamma) = \mathcal{B}^\omega(\dot{\Gamma})$. These two facts may

be confirmed by inspection of Fig. 8(b).

Actually $\mathcal{B}(\Gamma) = \mathcal{B}(\dot{\Gamma})$ in Fig. 8, i.e., the finite behaviors of Γ and Γ are equal as well as the infinite behaviors. There is a variant of Lemma 3.2-3 which covers this point. Since our main interest in the present section is in infinite behavior, we will merely state this result without proof. Let $\Gamma = (S,G,R,P,Q)$ be h-univalent and (S,G,R,P) solvable; then Γ^{h*} less its Q-projection is deterministic, $\mathcal{D}^h[\mathcal{B}(\Gamma)] = \mathcal{B}(\Gamma^{h*})$, and if Γ is in reduced form then $\mathcal{B}(\Gamma^{h*}) = \mathcal{B}(\Gamma^{h*t})$.

Figure 11 also illustrates Lemma 3.2-3. We begin with $\Gamma = (S, G, R, P, Q)$,

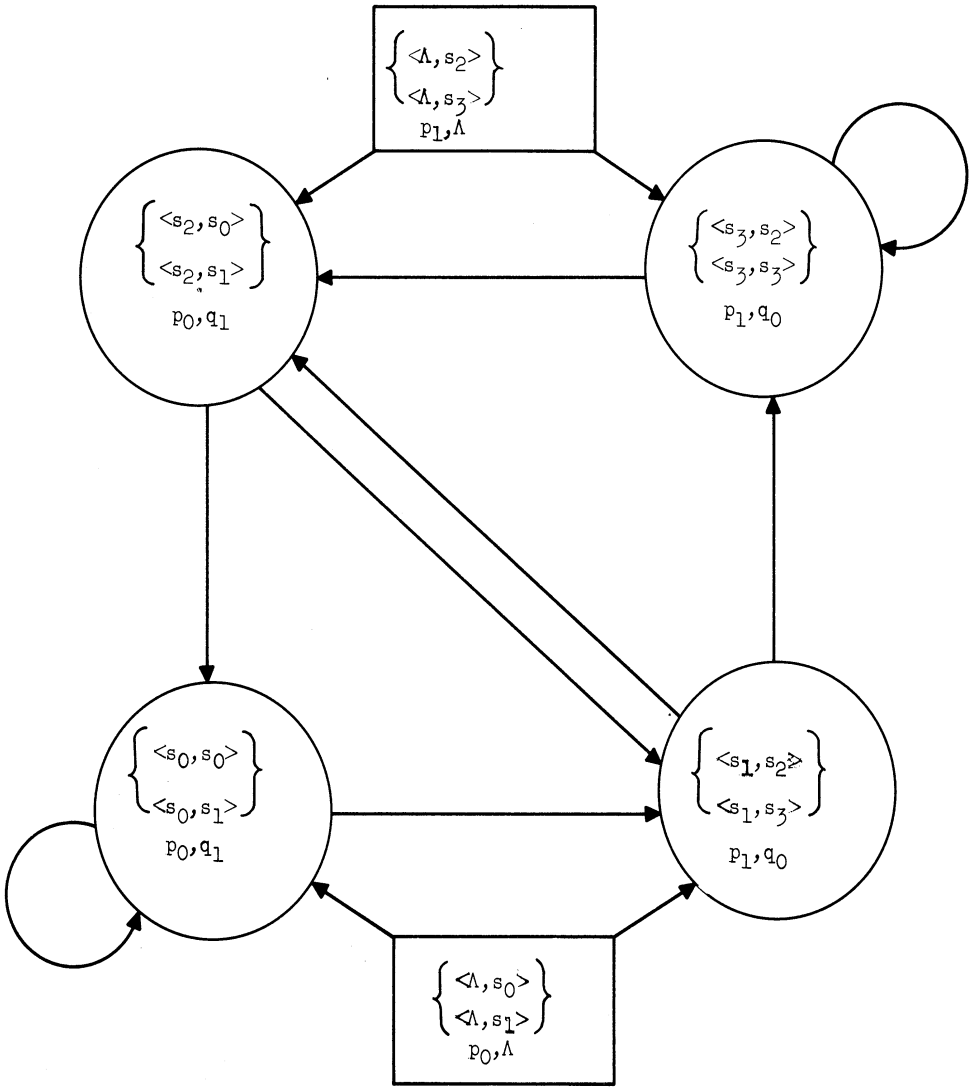


Fig. 11. Γ^{1*t}, where Γ is Figure 10(a). Γ and Γ^1 (less their last projections) are not deterministic, but Γ^{1*t} (less its last projection) is deterministic. $\mathcal{D}^1[\mathcal{B}^\omega(\Gamma)] = \mathcal{B}^\omega(\Gamma^{1*t})$. This illustrates Lemma 3.2-3.

where Γ is unit-univalent and (S, G, R, P) is solvable. Lemma 3.2-3 tells us that $\Gamma^{1*\dagger}$ (less its last projection) is deterministic, and also that $\mathscr{D}^1[\beta^\omega(\Gamma)] = \beta^\omega(\Gamma^{1*\dagger})$. [$\Gamma^1$ is shown in Fig. 10(b); it has 12 complete states. Γ^{1*} has 28 states, but only 6 of these are Γ^{1*}-admissible, so $\Gamma^{1*\dagger}$ has only 6 states.]

3.3. TIME-SHIFT THEOREM

We will prove now a lemma which is used in proving one of our main theorems (the Time-shift Theorem) and in validating a procedure for h-univalence.

Lemma 3.3-1 (Fixed Bound Lemma): Let $\Gamma = (S,G,R,P,Q)$ be a sequence generator with k Γ-admissible complete states. Then Γ is ω-univalent if and only if it is k^2-univalent.

Proof: The proof in one direction is obvious. To prove that if Γ is ω-univalent it is k^2-univalent, we consider any two Γ-sequences $[s_1](0,\omega)$, $[s_2](0,\omega)$ and any time t such that $P([s_1](0,t + k^2)) = P([s_2](0,t + k^2))$. Since there are k^2 distinct pairs of complete states, there are two times t_1, t_2 such that $t \leq t_1 < t_2 \leq t + k^2$, $s_1(t_1) = s_1(t_2)$, and $s_2(t_1) = s_2(t_2)$. Form the sequences

$$[s_3](0,\omega) = [s_1](0,t_2 - 1), [s_1](t_1,t_2 - 1), [s_1](t_1,t_2 - 1), \ldots$$
$$[s_4](0,\omega) = [s_2](0,t_2 - 1), [s_2](t_1,t_2 - 1), [s_2](t_1,t_2 - 1), \ldots$$

These are both Γ-sequences since they are composed of segments of Γ-sequences linked by the direct transition relation. Since $P([s_1](0,t + k^2)) = P([s_2](0,t + k^2))$ we have by construction $P([s_3](0,\omega)) = P([s_4](0,\omega))$. Because Γ is ω-univalent, $Q([s_3](0,\omega)) = Q([s_4](0,\omega))$. Then by construction $Q(s_1(t)) = Q(s_3(t))$ and $Q(s_2(t)) = Q(s_4(t))$, and so $Q(s_1(t)) = Q(s_2(t))$. Hence Γ is k^2-univalent.

Consider a sequence generator $\Gamma = (S,G,R,P,Q)$. If (S,G,R,P) is deterministic

then (S,G,R,P,Q) is uniquely solvable, but the converse does not in general hold [see Fig. 8(a)]. We noted earlier (Section 3.1) that unique solvability is essentially a property of the infinite behavior of a sequence generator. This suggests the question: what is the relation of the behaviors of uniquely solvable sequence generators to the behaviors of deterministic ones? This question is answered by the following theorem, which shows that for every uniquely solvable sequence generator there is a deterministic sequence generator whose infinite behavior is a displacement of the infinite behavior of the given sequence generator. In Section 4 we will introduce a concept of "computation." Using this concept, the result may be expressed: the behavior of every uniquely solvable sequence generator can be computed by a finite automaton.

Theorem 3.3-2 (Time-shift Theorem): Let $\Gamma = (S,G,R,P,Q)$ have k Γ-admissible complete states and let $\Gamma^{k^2*\dagger} = \dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$. Then

(a) if Γ is uniquely solvable, then $(\dot{S},\dot{G},\dot{R},\dot{P})$ is deterministic

(b) $\mathscr{D}^{k^2}[\mathcal{B}(\Gamma)] = \mathcal{B}^{\omega}(\dot{\Gamma})$.

Proof: This follows immediately from the definition of uniquely solvable (Section 3.1) Lemma 3.2-3, and the Fixed Bound Lemma (3.3-1).

Consider the Time-shift Theorem in relation to Γ of Fig. 10(a) and the derived $\Gamma^{1*\dagger}$ of Fig. 11. Γ is uniquely solvable and has 4 Γ-admissible complete states. Then the Time-shift Theorem tells us that $\Gamma^{16*\dagger}$, less its last projection, is deterministic. This is clearly so, for $\Gamma^{1*\dagger}$, less its last projection, is deterministic, and further applications of the ℓ-shift operation will obviously not destroy this property.

We pause to note an analogue of the Time-shift Theorem in which the shifting takes place in the opposite direction. The displacement operator $\mathscr{D}^{\ell}$ was defined to produce a right-shift of the Q-projections of a Γ-sequence; that is, it shifts the Q-projections ℓ steps later in time, leaving the

P-projections as they were. One could easily extend this operator to cover shifts in the opposite direction (i.e., with the Q-projections moved earlier in time); this could be symbolized by using the same operator $\mathcal{D}^{\ell}$, allowing negative as well as positive integer values for ℓ. Similarly, the ℓ-shift operator can be extended to produce shifts of the Q-projections to the left; again, we can use the same symbolism Γ^{ℓ} and signify left-shifts by negative values of ℓ. We then get the following partial analogue to the Time-shift Theorem. Let $\Gamma = (S,G,R,P,Q)$ be a sequence generator, with (S,G,R,P) deterministic. Let $\dot{\Gamma} = \Gamma^{\ell}$, where ℓ is negative. Then $\dot{\Gamma}$ is uniquely solvable, and $\mathcal{D}^{\ell}[\mathcal{B}^{\omega}(\Gamma)] = \mathcal{B}^{\omega}(\dot{\Gamma})$. Combining this with the Time-shift Theorem, we obtain the following result: the set of infinite behaviors of uniquely solvable sequence generators is exactly the set of displaced infinite behaviors of deterministic sequence generators.

It is not obvious from the definition that the class of h-univalent sequence generators is decidable. However, this is in fact the case, as we will now show.

h-univalence Procedure (where h is any non-negative integer of ω):
Let $\Gamma = (S,G,R,P,Q)$ be the given sequence generator. Find k, the number of admissible complete states, by the Reduced Form Algorithm. Let $\ell = \min(h,k^2)$. Form $\Gamma^{\ell *\dagger} = \dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$. Answer "yes" or "no" as $(\dot{S},\dot{G},\dot{R},\dot{P})$ is semi-deterministic or not.

Theorem 3.3-3: The h-univalence procedure is a decision procedure for the class of h-univalent sequence generators.

Proof: We will use the notation of the algorithm. By the Fixed Bound Lemma Γ is h-univalent if and only if Γ is ℓ-univalent. By Corollary 3.2-2 Γ is ℓ-univalent if and only if Γ^{ℓ} is 0-univalent. By Lemma 3.1-4 Γ^{ℓ} is 0-univalent if and only if $(\dot{S},\dot{G},\dot{R},\dot{P})$ is semi-deterministic. As noted in Section 2.1, it is obvious from the definition of semi-determinism that there is a decision

procedure for the class of semi-deterministic sequence generators. This completes the proof of the theorem.

It can be shown that the following is a characterization of h-univalence. Let $\Gamma = (S,G,R,P,Q)$, k the number of Γ-admissible complete states, and $\ell = \min(h,k^2)$. Then Γ is h-univalent, if and only if, for any two Γ-sequences $[s_1](0,\omega)$ and $[s_2](0,\omega)$ and any time $t \leq k^2$, if $P([s_1](0,t + \ell)) = P([s_2](0,t + \ell))$, then $Q(s_1(t)) = Q(s_2(t))$. This characterization can be made the basis of a decision procedure for h-univalence which is more efficient than the one we have given.

Since unique solvability is defined in terms of solvability and ω-univalence (Section 3.1), by combining the ω-univalence Procedure with the decision procedure for solvability of Theorem 2.3-2, we obtain a decision procedure for unique solvability.

4. GENERALIZATIONS AND APPLICATIONS

4.1. COMPUTATION

We will next define a concept of "computation" which corresponds more closely to the way a digital computer is used than does the concept of behavior; essentially the same concept is defined in Burks, 1960. Computers are employed to produce answers to questions; the questions go in as inputs, the answers appear as outputs. Generally speaking, the answer is not produced immediately, but only after a time delay. Moreover, except in real-time computation, the answer will not appear at the same rate as the input information is received. In contrast, all outputs of a computer are part of its behavior, whether they contribute to the answer or not. For these reasons the concept of behavior does not fit the question-answer mode of using a computer as closely as the concept of computation to be defined. Computation differs from behavior in that in the case of computation not all "output states" are interpreted as part of the answer, but only those selected as the "computed output states" by the sequence generator itself. The concept of computation reflects the fact that in our theory the internal operations of a sequence generator are strictly correlated to the basic time scale, while the computed outputs are not.

In the definition of computation we will need the μ-operator (selection operator). Suppose ϕx expresses some condition on the natural numbers. "$(\mu x)\phi x$" designates the smallest number satisfying the condition ϕx if there is one; if no number satisfies ϕx, then "$(\mu x)\phi x$" is undefined. For example, $(\mu x)(x > 3^2) = 10$, while "$(\mu x)(x^2 = 2)$" is undefined.

Definition: Let $\Gamma = (S,G,R,P,Q,C)$ be a sequence generator with the projection C having only the two values 0, 1. Let $[s](0,\omega)$ be an arbitrary

infinite Γ-sequence and define

$$\gamma(t) = Q\{s(\mu x)\{\sum_{y=0}^{x} C\{s(y)\} = t + 1\}\};$$

note that if $\sum_{y=0}^{\infty} C\{s(y)\}$ is finite then $\gamma(t)$ is undefined for any $t \geq \sum_{y=0}^{\infty} C\{s(y)\}$. If $\sum_{y=0}^{\infty} C\{s(y)\}$ is unbounded, then the sequence

$$< P\{s(0)\}, \gamma(0) >, < P\{s(1)\}, \gamma(1) >, \ldots$$

is an <u>infinite computation element</u> of Γ. If $\sum_{y=0}^{\infty} C\{s(y)\} = k$ (k being finite), then the sequence

$$< P\{s(0)\}, \gamma(0) >, \ldots, < P\{s(k-1)\}, \gamma(k-1) >, <P\{s(k)\} >, < P\{s(k+1)\}>, \ldots$$

is a <u>finite computation element</u> of Γ. The <u>computation</u> of Γ, denoted by $\mathcal{C}(\Gamma)$, is the set composed of all infinite computation elements and all finite computation elements of Γ.

An example will help make the concept of computation clear. Consider a sequence generator Γ which has only two infinite Γ-sequences. These are shown below, together with the projections of each and the derived sequences $[\gamma](0,k)$.

$$[s_1](0,\omega) = s_0, s_1, s_2, s_3, s_3, s_3, \ldots$$
$$P([s_1](0,\omega)) = p_0, p_1, p_1, p_3, p_3, p_3, \ldots$$
$$Q([s_1](0,\omega)) = q_0, q_0, q_1, q_1, q_1, q_1, \ldots$$
$$C([s_1](0,\omega)) = 0, 1, 1, 0, 0, 0, \ldots$$
$$[\gamma_1](0,1) = q_0, q_1$$

$$[s_2](0,\omega) = s_0, s_1, s_2, s_4, s_5, s_4, s_5, s_4, s_5, \ldots$$
$$P([s_2](0,\omega)) = p_0, p_1, p_1, p_3, p_5, p_3, p_5, p_3, p_5, \ldots$$
$$Q([s_2](0,\omega)) = q_0, q_0, q_1, q_2, q_3, q_2, q_3, q_2, q_3, \ldots$$
$$C([s_1](0,\omega)) = 0, 1, 1, 0, 1, 0, 1, 0, 1, \ldots$$
$$[\gamma_2](0,\omega) = q_0, q_1, q_3, q_3, q_3, \ldots$$

Hence the computation $\mathcal{C}(\Gamma)$ is the set consisting of the two sequences

$$< p_0, p_0 >, < p_1, q_1 >, < p_1 >, < p_3 >, < p_3 >, < p_3 >, \ldots$$

$$< p_0, q_0 >, < p_1, q_1 >, < p_1, q_3 >, < p_3, q_3 >, < p_5, q_3 >, \ldots$$

Note that the first of these is a finite computation element of Γ while the second is an infinite computation element of Γ.

We will now show that the concept of computation is a bona fide generalization of the concept of behavior. Let α be the class of all two-projection sequence generators. Let β be the class of all three-projection sequence generators $\Gamma = (S,G,R,P,Q,C)$ such that C has the values 0, 1 and every element of $\mathcal{C}(\Gamma)$ is infinite. Then the class $\{\mathcal{B}^{\omega}(\Gamma) \,|\, \Gamma \epsilon \alpha\}$ of infinite behaviors of elements of α is a proper subclass of the class $\{\mathcal{C}(\Gamma) \,|\, \Gamma \epsilon \beta\}$ of the computations of elements of β. To see that

$$\{\mathcal{B}^{\omega}(\Gamma) \,|\, \Gamma \epsilon \alpha\} \subsetneq \{\mathcal{C}(\Gamma) \,|\, \Gamma \epsilon \beta\},$$

note that for each element $\Gamma = (S,G,R,P,Q)$ of α there is an element $\dot{\Gamma} = (S,G,R,P,Q,C)$ of β in which $C(s) = 1$ for every $s \epsilon S$, so that $\mathcal{B}^{\omega}(\Gamma) = \mathcal{C}(\dot{\Gamma})$. That the inclusion of $\{\mathcal{B}^{\omega}(\Gamma) \,|\, \Gamma \epsilon \alpha\}$ in $\{\mathcal{C}(\Gamma) \,|\, \Gamma \epsilon \beta\}$ is a proper one is shown by Fig. 12. In Fig. 12, $\dot{P}(\dot{s}_0) = \dot{P}(\dot{s}_2) = 0$, $\dot{P}(\dot{s}_1) = \dot{P}(\dot{s}_3) = 1$, $\dot{P}(\dot{s}) = \dot{Q}(\dot{s})$ for every $\dot{s} \epsilon \dot{S}$, $C(\dot{s}_0) = C(\dot{s}_1) = 1$, and $C(\dot{s}_2) = C(\dot{s}_3) = 0$. The sequence generator $\dot{\Gamma}$ of this figure has the property that

$$\mathcal{C}(\dot{\Gamma}) \notin \{\mathcal{B}^{\omega}(\Gamma) \,|\, \Gamma \epsilon \alpha\}.$$

To see this, note that since $\dot{P}(\dot{s}) = \dot{Q}(\dot{s})$ for every $\dot{s} \epsilon \dot{S}$, the computation $\mathcal{C}(\dot{\Gamma})$ consists of all infinite sequences of the form

$$< \dot{P}(\dot{s}(0)), \dot{P}(\dot{s}(0)) >, < \dot{P}(\dot{s}(1), \dot{P}(\dot{s}(2)) >, \ldots, < \dot{P}(\dot{s}(t)), \dot{P}(\dot{s}(2t)), \ldots,$$

where $\dot{P}(\dot{s})$ is either zero or one. By the Fixed Bound Lemma (3.3-1) no uniquely

solvable sequence generator can have $\mathcal{C}(\dot{\Gamma})$ as its behavior, and since, as we remarked in Section 3.1, unique solvability is essentially a property of the infinite behavior of a sequence generator, no sequence generator can have $\mathcal{C}(\dot{\Gamma})$ as its infinite behavior. Since ω-univalence and unique solvability apply to sequences of pairs (Section 3.1), these concepts may be extended to cover the computation of a sequence generator; the computation $\mathcal{C}(\dot{\Gamma})$ of the sequence generator of Fig. 12 is uniquely solvable.

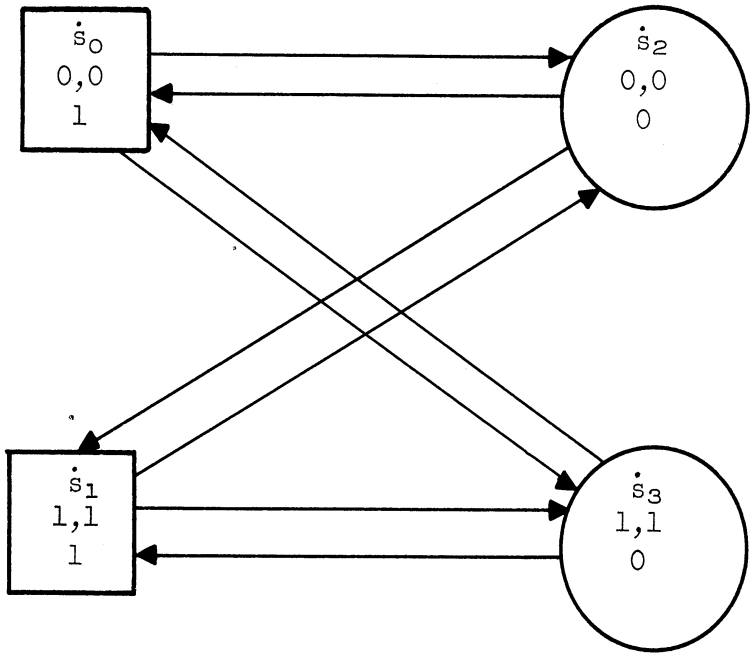


Fig. 12. $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q},\dot{C})$. No sequence generator can have $\mathcal{C}(\dot{\Gamma})$ as its infinite behavior.

Since computation is a bona fide generalization of behavior, the existence of certain algorithms for behavior (Section 2.3) does not guarantee the existence of corresponding algorithms for computation. It is not known whether either of these two algorithms exist: an algorithm to decide whether the computation of one sequence generator is included in the computation of another, a decision procedure for the computational equivalence of two sequence generators. We do have an algorithm to decide of any pair of sequence generators $\Gamma = (S,G,R,P,Q,C)$ and $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q},\dot{C})$ such that $\mathcal{C}(\dot{\Gamma})$ is ω-univalent, whether the computation $\mathcal{C}(\Gamma)$ is included in the computation $\mathcal{C}(\dot{\Gamma})$ or not, but this

algorithm and its justification are too long and involved to be included here.

There is a procedure for deciding of a sequence generator $\Gamma = (S,G,R,P,Q,C)$, where C has only the values 0, 1, whether all, some but not all, or none of the elements of $\mathcal{C}(\Gamma)$ are infinite. One first finds the reduced form $\Gamma\dagger$. Note that $\mathcal{C}(\Gamma\dagger) = \mathcal{C}(\Gamma)$ for any Γ. Then each non-repetitive cycle of complete states of $\Gamma\dagger$ is examined to determine whether $C(s_i) = 1$ for any state s_i of this cycle. If $C(s_i) = 1$ for some state s_i of this cycle, there will be an infinite $\Gamma\dagger$-sequence made of iterations of the cycle, which $\Gamma\dagger$-sequence will give rise to an infinite computation element of $\mathcal{C}(\Gamma\dagger)$ and hence of $\mathcal{C}(\Gamma)$. On the other hand, if for every state s_i of a cycle $C(s_i) = 0$, then there is an infinite $\Gamma\dagger$-sequence made of repetitions of this cycle which will give rise to a finite computation element of $\mathcal{C}(\Gamma)$.

Let us now return to the Time-shift Theorem (3.3-2) and extend it to cover the case of computation. Let $\Gamma = (S,G,R,P,Q,C)$ have k Γ-admissible complete states and let C have only the values 0, 1. The ℓ-shift operator was defined for two-projection sequence generators, but it may easily be extended to cover sequence generators with a computation projection C in the following way. Form (S,G,R,P,QxC); apply the k^2-shift operator to it with the modification that $< \Lambda, 0 >$ is the QxC projection of those admissible complete states that occur before time k^2 (cf. Section 3.2). Call the result $(\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q}x\dot{C})$. Then form $(\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q},\dot{C})^{*\dagger} = \ddot{\Gamma} = (\ddot{S},\ddot{G},\ddot{R},\ddot{P},\ddot{Q},\ddot{C})$. Now apply Lemma 3.2-3 and 3.3-1, noting that a displacement of the infinite behavior of a sequence generator with a computation projection does not alter its computation, since the behavior includes the computed output. The net result of all this is as follows. Given a sequence generator $\Gamma = (S,G,R,P,Q,C)$ with C having the values 0, 1, there is an effective procedure for constructing $\ddot{\Gamma} = (\ddot{S},\ddot{G},\ddot{R},\ddot{P},\ddot{Q},\ddot{C})$ such that:

(a) If (S,G,R,P,QxC) is uniquely solvable, then $(\ddot{S},\ddot{G},\ddot{R},\ddot{P})$ is deterministic,

(b) $\mathcal{C}(\Gamma) = \mathcal{C}(\ddot{\Gamma})$.

When the foregoing statement is applied to the case where C is always one, it implies that the infinite behavior of any uniquely solvable sequence generator is the computation of a deterministic sequence generator. This justifies the statement made prior to the Time-shift Theorem (3.3-2) that the infinite behavior of any uniquely solvable sequence generator can be computed by a finite automaton. We will illustrate this with an example (Fig. 13). We begin with Γ of Fig. 10(a), which is uniquely solvable (but not deterministic). In this case a unit-shift, followed by an application of the * and †, suffices to produce a deterministic sequence generator $\Gamma^{1*\dagger}$, shown in Fig. 11. $\dot{\Gamma}$ of Fig. 13(a) is a simplification of this. To $\dot{\Gamma}$ we now add a projection C which is 0 for the generators, 1 otherwise, obtaining $\ddot{\Gamma}$ of Fig. 13(b). $\ddot{\Gamma}$ is a deterministic sequence generator which computes the infinite behavior of the original Γ, i.e., $\mathcal{C}(\ddot{\Gamma}) = \mathcal{B}^{\omega}(\Gamma)$. By a slight generalization of the process described in Section 2.1, we can pass from the deterministic sequence generator $\ddot{\Gamma}$ to a finite automaton and to a w.f.n. The w.f.n. which produces the computation $\mathcal{C}(\ddot{\Gamma})$ is shown in Fig. 13(c). It is perhaps worth noting that the computation of this net [Fig. 13(c)] is the infinite behavior of the net of Fig. 2(a).

The concept of unique solvability may be generalized to cover computation. Let $\Gamma = (S,G,R,P,Q,C)$ be a sequence generator, C having the values 0, 1. The computation $\mathcal{C}(\Gamma)$ is "uniquely solvable" if for each infinite sequence of P-states $[p](0,\omega)$ there is exactly one element of $\mathcal{C}(\Gamma)$ which contains $[p](0,\omega)$ (i.e., which is composed of this sequence of P-states together with zero or more Q-states). The motivation behind the concept of a computation being uniquely solvable is this. We may think of a sequence generator with an input projection P, an output projection Q, and a control projection C as specifying a computational relation between infinite input sequences and computed

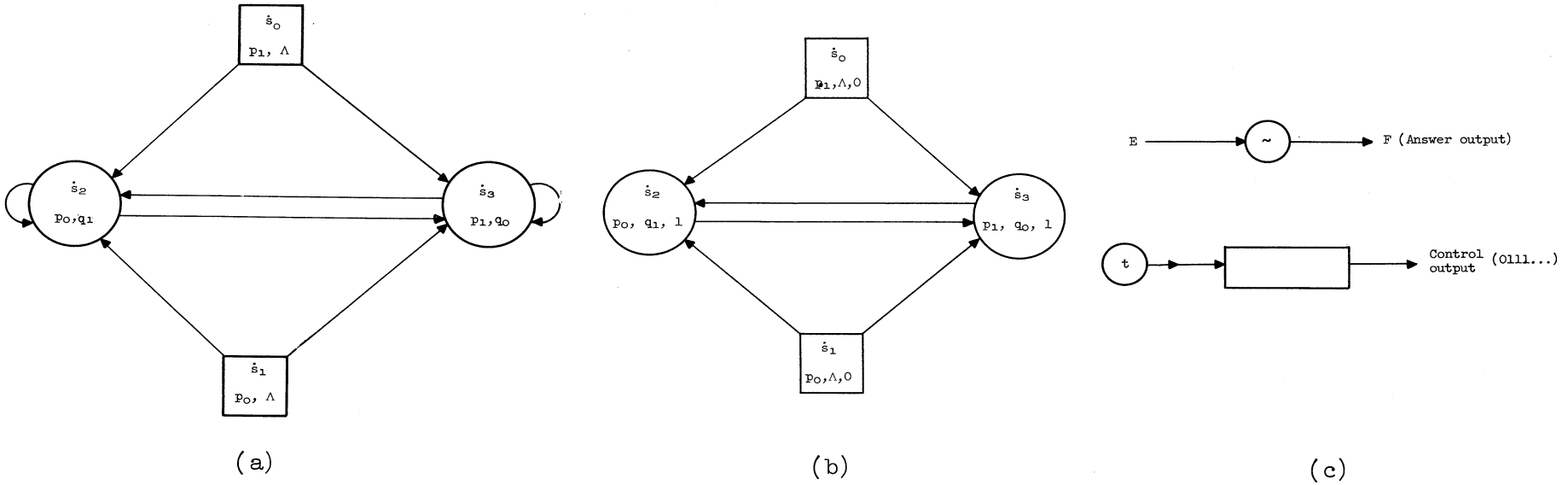


Fig. 13. (a) $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$, which results from $\Gamma^{1*\dagger}$ of Fig. 11 by identifying and re-naming behaviorally equivalent states. $\dot{\Gamma}$, less its last projection, is deterministic. $\mathcal{B}(\dot{\Gamma}) = \mathcal{B}(\Gamma^{1*\dagger})$. (b) $\ddot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q},C)$, which results from $\dot{\Gamma}$ by adding a control projection $C(\dot{s}_0) = C(\dot{s}_1) = 0$, $C(\dot{s}_2) = C(\dot{s}_3) = 1$. $\mathcal{C}(\Gamma) = \mathcal{B}^{\omega}(\Gamma)$, where Γ is Fig. 10(a). (c) A net which has the computation $\mathcal{C}(\ddot{\Gamma})$.

output sequences, and the sequence generator does it uniquely if each input sequence determines exactly one computed output sequence. It is easy to construct examples of sequence generators $\Gamma = (S,G,R,P,Q,C)$ such that (S,G,R,P,Q) is not uniquely solvable, though the computation $\mathcal{C}(\Gamma)$ is. Thus we are led naturally to the question: Is there a decision procedure which will tell whether the computation $\mathcal{C}(\Gamma)$ of a sequence generator $\Gamma = (S,G,R,P,Q,C)$ is uniquely solvable? It is also of interest to know whether the analogue of the Time-shift Theorem (3.3-2) holds for uniquely solvable computations, i.e., whether or not the following is true of a sequence generator $\Gamma = (S,G,R,P,Q,C)$, with C having the values 0, 1: if $\mathcal{C}(\Gamma)$ is uniquely solvable, then there is a sequence generator $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q},\dot{C})$ such that $(\dot{S},\dot{G},\dot{R},\dot{P})$ is deterministic and $\mathcal{C}(\Gamma) = \mathcal{C}(\dot{\Gamma})$. We do not know the answer to either of these questions.

4.2. FORMULAS AND SEQUENCE GENERATORS

We will now discuss the relation of some formulas of symbolic logic to sequence generators. Because of limitations of space we will not give a detailed or rigorous treatment of the subject but will rely heavily on examples and will present theorems without proofs.

The language L being considered here is a first-order monadic predicate calculus with a successor function and zero. The symbols of L are: an infinite list of monadic predicate variables A, B, C, . . . ; an infinite list of individual variables t, t_1, t_2, . . . ; the successor function ' ; the individual constant 0; all truth-functional connectives; and parentheses. The individual variables range over natural numbers 0, 0', 0", ... The predicate variables range over predicates of natural numbers, i.e., over sets of natural numbers.

Consider an arbitrary w.f.f. of L; the result of universally quantifying

all the individual variables of L is called an A-formula (arbitrary-formula). $B(t_1) \equiv B(t_2''')$ is a w.f.f. of L, and so $(t_1)(t_2)[B(t_1) \equiv B(t_2''')]$ is an A-formula. $(t_1)(t_2)[B(t_1) \equiv B(t_2''')]$ means that for all times t_1, t_2, B is true at time t_1 if and only if B is true at time $t_2 + 3$. Note that since language L does not contain quantifiers, an A-formula is not a w.f.f. of L; this does not matter for our purposes.

The extension of an A-formula with predicate variables $B_1, B_2, \ldots, B_k$ is the set of all k-tuples of predicates which satisfy the formula, i.e., for which the formula is true of the natural numbers when B_i is interpreted as the i'th predicate of the k-tuple. We will regard a predicate, e.g., B_i, as an infinite binary sequence $[s](0,\omega)$ in which s(t) is 1 or 0 according to whether $B_i(t)$ is true or false. When predicates are viewed in this way, a k-tuple of predicates becomes an ω-sequence of k-tuples or column vectors, i.e., a k by ω binary matrix.

We will give some examples. $(t_1)(t_2)[B(t_1) \equiv B(t_2''')]$ is satisfied by 1111... but not by 1010... , for in the later case $B(0) \not\equiv B(0''')$. The extension of $(t)[B(0'') \ \& \ (B(t) \equiv B(t''))]$ consists of all sequences of the form

$$1, x_1, 1, x_2, 1, x_3, 1, x_4, \ldots .$$

The extension of $(t)\{\overline{B_2(0)} \ \& \ [B_1(t) \equiv B_2(t')]\}$ consists of all two by omega matrices of the form

$$\begin{matrix} x_0 & x_1 & x_2 & x_3 & \ldots \\ 0 & x_1 & x_2 & x_3 & \ldots . \end{matrix}$$

Sequence generators without projections may be translated into a particular kind of A-formula, called an M-formula. For example, the sequence generator of Fig. 14 may be translated into

$$(t)[(\overline{B_1(0)} \;\&\; \overline{B_2(0)}) \;v\; (B_1(0) \;\&\; B_2(0))] \;\&\; \{(\overline{B_1(t)} \;\&\; \overline{B_2(t)} \;\&\; \overline{B_1(t')} \;\&\; B_2(t')) \;v\; (\overline{B_1(t)} \;\&\; B_2(t) \;\&\; \overline{B_1(t')} \;\&\; \overline{B_2(t')})\}.$$

The bracketed conjunct of this A-formula contains no individual symbol other than the constant zero; it tells us what the generators are. The braced conjunct contains no individual symbols other than t and t'; it tells what direct transitions are possible. Clearly, the set of infinite Γ-sequences of Fig. 14

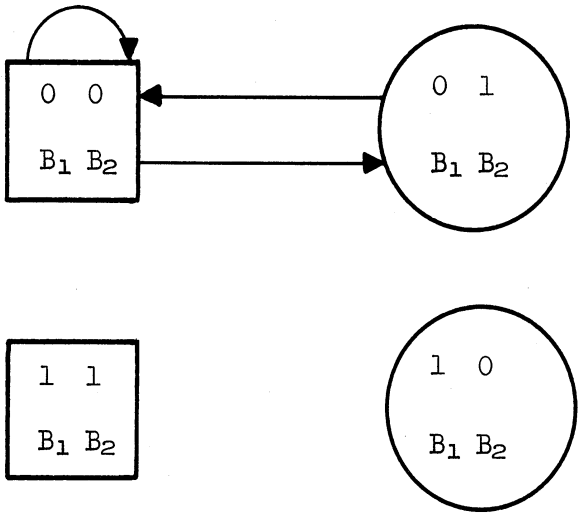


Fig. 14. $\Gamma = (S,G,R)$.

equals the extension of the formula given above. Any A-formula consisting of a universal individual quantifier operating on a conjunction, the first conjunct having only zero as argument, the second conjunct having only one variable with at most one prime as argument, is called an M-formula (minimal formula). Our previous example illustrates the fact that any sequence generator $\Gamma = (S,G,R)$ can be translated into an M-formula $\mathcal{F}$ with k predicates by coding its states into binary k-tuples, the extension of $\mathcal{F}$ being the set of Γ-sequences. Conversely, any M-formula $\mathcal{F}$ may be translated into a sequence generator $\Gamma = (S,G,R)$ such that the set of infinite Γ-sequences is the extension of $\mathcal{F}$.

We will consider next two types of A-formulas more general than M-formulas.

Any A-formula of the form $(t)\mathcal{F}_1(0, 0^1, \ldots, t^{j1}) \;\&\; \mathcal{F}_2(t, t^1, \ldots, t^{j2})$, where j_1, j_2, indicate the maximum number of primes of arguments of $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively, is a D-formula (decomposable-formula).

"$[B(0) \;\&\; B(1) \;\&\; \{(\overline{B(t)} \;\&\; \overline{B(t')}) \equiv B(t'')\}]$"is a D-formula whose extension is the single infinite sequence 001001... . In Section 2.1 (immediately after corollary 2.1-1) we proved that there is no sequence generator $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R})$ whose set of infinite Γ-sequences consists of the single sequence 001001... . It follows by the results described in the preceding paragraph that there is no M-formula whose extension consists of this sequence, i.e., there is no M-formula logically equivalent to the D-formula given above.

Any A-formula with at most one individual variable is an O-formula (one-variable formula). $(t)[B(0) \supset (B(t) \equiv B(t'))]$ is an O-formula whose extension consists of 111... together with all binary sequences beginning with zero. It can be proved that there is no D-formula with this extension, i.e., $(t)[B(0) \supset (B(t) \equiv B(t'))]$ cannot be decomposed into a D-formula $(t)(\mathcal{F}_1 \;\&\; \mathcal{F}_2)$ logically equivalent to it. Thus O-formulas are stronger than D-formulas, just as D-formulas are stronger than M-formulas. A-formulas are stronger than all these, for it can be proved that there is no O-formula with the extension of the A-formula $[\overline{B(t_1)} \vee \overline{B(t_1')}] \vee [B(t_2) \vee B(t_2')]$. This formula is equivalent to the condition that a binary sequence cannot have both consecutive ones and consecutive zeros.

There is thus a hierarchy of A-formulas, with sequence generators corresponding to the formulas of lowest level, i.e., the M-formulas. This suggests generalizing the concept of sequence generator so there is a kind of sequence generator corresponding to each level of the hierarchy. This could be done, as an example will make clear. The generalized sequence generator $\Gamma = (S, R_G^3)$ is defined as follows. S is the set $\{0,1\}$. R_G^3 is the triadic relation $\{< 100 >, < 111 >, < 000 >, < 001 >, < 010 >, < 011 >\}$. An infinite Γ-sequence

of this generalized sequence generator is any binary sequence $[s](0,\omega)$ such that all t, $R_G^3(s(0), s(t), s(t'))$. Note that generators and the direct transition relation are not defined separately in $\Gamma = (S,R_G^3)$. The set of Γ-sequences of (S, R_G^3) is equivalent to the extension of the O-formula $(t)[B(0) \supset (B(t) \equiv B(t'))]$. In this example the states of the sequence generator are zero and one, with the associated A-formula having one predicate (B). In general, of course, the states will be k-tuples of zeros and ones and the associated A-formula will have k predicates. Proceeding in a similar manner one can define generalized sequence generators corresponding to D-formulas and to A-formulas, with the result that for each formula $\mathcal{F}$ of a given type there is a corresponding generalized sequence generator Γ such that the extension of $\mathcal{F}$ equals the set of infinite Γ-sequences of Γ.

While these generalizations are of interest in showing the relations of formulas to sequence generators, they are not as easy to work with as the corresponding D-formulas, O-formulas, and A-formulas. In contrast, sequence generators are easier to work with than the corresponding M-formulas. Moreover, by employing projections it is possible to reduce any A-formula $\mathcal{F}$ to a sequence generator $\Gamma = (S,G,R,P)$ so that the infinite behavior $\beta^{\omega}(\Gamma)$ equals the extension of $\mathcal{F}$, and in this way to investigate A-formulas of all kinds by means of sequence generators with projections. After defining some terms we will explain this process in more detail.

A <u>ΣA-formula</u> is the result of prefixing a sequence of existential predicate quantifiers to an A-formula; "ΣO-formula," "ΣD-formula," and "ΣM-formula" are similarly defined. We will call the set of k-tuples of predicates which satisfy a ΣA-formula the <u>behavior</u> of the formula. For example, the binary sequence 00110011... is the behavior of the M-formula

$$(\Sigma C)(t)\{[\overline{B(0)} \ \& \ \overline{C(0)}] \ \& \ [C(t) \not\equiv C(t')] \ \& \ (B(t') \not\equiv (B(t) \equiv C(t)))]\}.$$

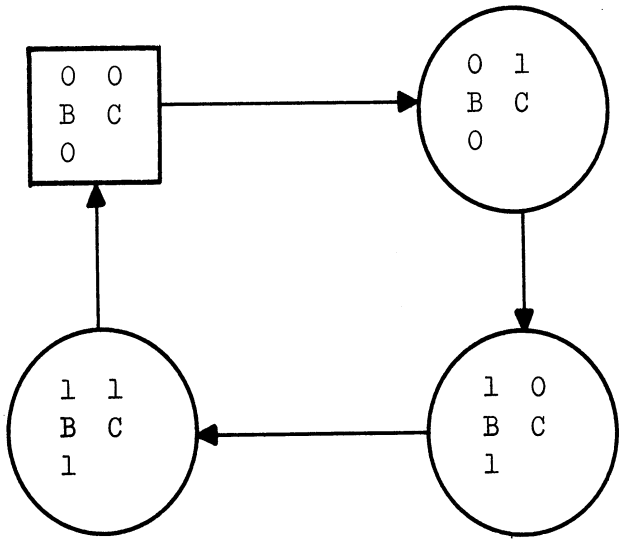


Fig. 15. $\Gamma = (S,G,R,P)$.

Our preceding examples show that the sets of extensions of M-formulas, D-formulas, O-formulas, and A-formulas get progressively larger. In contrast, the set of behaviors of ΣA-formulas equals the set of behaviors of ΣM-formulas. Given any ΣA-formula, one can, by a procedure of Church, 1960, p. 36 ff., construct a behaviorally equivalent ΣO-formula. There is, moreover, a process for reducing a ΣO-formula to a behaviorally equivalent ΣM-formula. A description of the process is too long to include here, but the essential steps are illustrated in the following example. Consider the O-formula

$$(t)[B(0') \supset \{B(t'') \equiv B(t)\}].$$

We introduce a predicate C_1 which is defined by the conditions $C_1(t') \equiv C_1(t)$ and $B(0') \equiv C_1(0)$, so that $C_1(t) \equiv B(0')$ for all t. Since these conditions imply that $C_1(t) \equiv B(0')$, we may substitute $C_1(t)$ for $B(0')$ in the original O-formula and conjoin the two conditions to obtain a ΣD-formula $(\Sigma C_1)(t)[B(0') \equiv C_1(0)] \;\&\; [\{C_1(t) \supset (B(t'') \equiv B(t))\} \;\&\; \{C_1(t') \equiv C_1(t)\}]$ which has the same behavior as the given O-formula. We next introduce a

predicate C_2, defined by the condition $B(t') \equiv C_2(t)$, which implies that $B(t'') \equiv C_2(t')$ and $B(0') \equiv C_2(0)$. Finally, we substitute $C_2(t')$ for $B(t'')$ and $C_2(0)$ for $B(0')$ in the ΣD-formula just obtained and conjoin the condition $B(t') \equiv C_2(t)$, thereby obtaining a ΣM-formula

$$(\Sigma C_2)(\Sigma C_1)(t)[C_2(0) \equiv C_1(0)] \,\&\, [\{C_1(t) \supset (C_2(t') \equiv B(t))\} \,\&\, \{C_1(t') \equiv C_1(t)\} \,\&\, \{B(t') \equiv C_2(t)\}]$$

which has the same behavior as the ΣD-formula and hence as the original 0-formula.

Each ΣM-formula $\mathcal{F}$ may be converted into a one-projection sequence generator $\Gamma = (S,G,R,P)$ such that $\mathfrak{B}^{\omega}(\Gamma)$ is the behavior of $\mathcal{F}$, and conversely. Again we will not state the algorithm for this conversion but will illustrate it. Consider the ΣM-formula

$$(\Sigma C)(t)\{[\overline{B(0)} \,\&\, \overline{C(0)}] \,\&\, [(C(t) \not\equiv C(t')) \,\&\, (B(t') \not\equiv (B(t) \equiv C(t)))]\}.$$

Drop the existential quantifier and convert the resultant M-formula into a sequence generator (S,G,R) in the way indicated before; see Fig. 15. Next, add a projection P defined so that $P(B(t) \,\&\, C(t)) = B(t)$, obtaining $\Gamma = (S,G,R,P)$. $\mathfrak{B}^{\omega}(\Gamma)$ is the behavior of the original M-formula. The reverse procedure of going from an arbitrary one-projection sequence generator to a ΣM-formula with the same behavior is only slightly more difficult.

To sum up: given a ΣA-formula, one can effectively construct a ΣM-formula which has the same behavior. Given a ΣM-formula $\mathcal{F}$, one can effectively construct $\Gamma = (S,G,R,P)$ such that $\mathfrak{B}^{\omega}(\Gamma)$ equals the behavior of $\mathcal{F}$. Conversely, given $\Gamma = (S,G,R,P)$ one can effectively find a ΣM-formula $\mathcal{F}$ whose behavior is $\mathfrak{B}^{\omega}(\Gamma)$. Hence the class of infinite behaviors of one-projection sequence generators equals the class of behaviors of ΣA-formulas, and so ΣA-formulas may be investigated by means of sequence generators with one projection.

One can go further than this by classifying the free predicate variables of a ΣA-formula so as to correspond to several projections. For example, one can divide the free predicate variables of a ΣA-formula into two categories, the first corresponding to a projection P, the second to a projection Q. When this is done, the two-projection concepts and theorems of Section 3 apply to ΣA-formulas. It is worth noting what the Time-shift Theorem (3.3-2) becomes when looked at in this way. A deterministic sequence generator corresponds to a ΣM-formula in which the Q-predicates (outputs) and bound predicates at time t + 1 are defined recursively in terms of themselves at time t and of the P-predicates (inputs) at times t and t + 1; this is essentially what Church, 1960, p. 11, calls a system of "restricted recursions." One may also consider recursive definitions in which the values of the Q-predicates and bound predicates at time t depends on the values of the P-predicates at later times; these recursions are essentially what Church, 1960, p. 12, calls "unrestricted singulary recursions," and can be expressed by a special type of ΣD-formula.

Put in these terms, the Time-shift Theorem becomes: for every uniquely solvable ΣA-formula there is a logically equivalent ΣD-formula which is composed of unrestricted singulary recursions. This ΣD-formula is of a special form which may be thought of as a normal form for uniquely solvable ΣA-formulas. A normal form ΣD-formula translates into a well-behaved net (see the last paragraph of Section 3.1) which consists of a well-formed part (corresponding to a deterministic sequence generator) together with a sequence of delays (which shifts the inputs earlier in time), and so the Time-shift Theorem can also be interpreted as saying that every well-behaved net has a normal form. Figure 16 gives the normal form for the well-behaved net of Fig. 2(a). The upper part produces the time-shift $H(t) \equiv E(t')$ while the lower part, which is well-formed, produces $F(t) \equiv \sim H(t)$; hence $F(t) \equiv \sim E(t')$, as in Fig. 2(a).

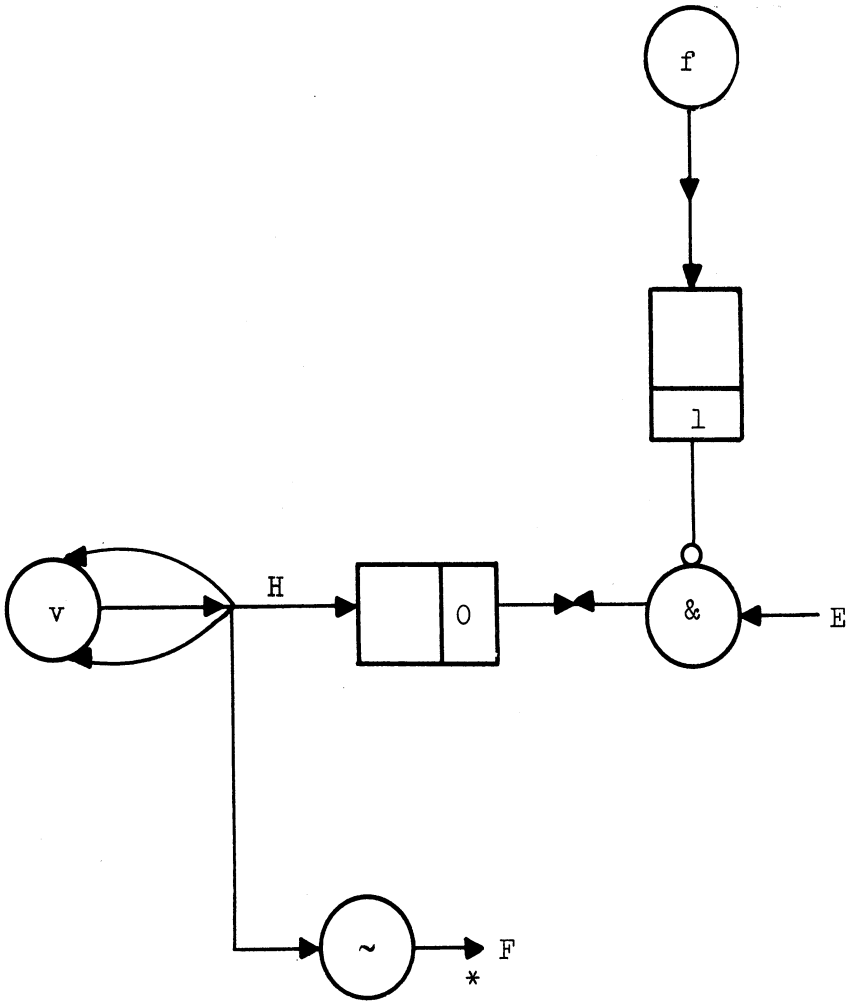


Fig. 16. Normal form of the well-behaved net of Fig. 2(a). $F(t) \equiv \sim E(t')$.

One can think of a uniquely solvable ΣA-formula whose free predicate variables are divided into P-predicates and Q-predicates as giving an implicit definition of the Q-predicates in terms of the P-predicates. Looked at in this way the Time-shift Theorem tells us that for every ΣA-formula which implicitly defines the Q-predicates in terms of the P-predicates, there is a ΣD-formula which recursively defines the Q-predicates in terms of the P-predicates.

4.3. SEQUENCE GENERATORS AND CONDITIONS

Suppose one wishes to design an automaton or other deterministic information-processing system. Using a formula, a diagram, or a set of tables, one may specify the output of the system as a deterministic function of the inputs of the system. Sometimes, however, the designer has in mind only a condition or requirement which he wishes the behavior of the device to satisfy, there being many different behaviors which satisfy this condition. The designer may wish to consider all systems whose behaviors satisfy the given condition and select from among these by some criterion, such as minimality of components. A well-known example of this is the use of "don't care" cases in formulating a switch requirement. Because the requirement imposes no restrictions on the switch behavior for the "don't care" inputs, it may be satisfied by different switching functions. The designer wishes to select from all switches which satisfy this requirement one with a minimal number of switching elements.

Many different languages may be used for expressing conditions on information-processing systems and for describing such systems. Consider first the language of ΣA-formulas (Section 4.2). Suppose $\mathcal{F}$ and $\dot{\mathcal{F}}$ are ΣA-formulas whose free predicate variables are divided into input variables and output variables. $\mathcal{F}$ may describe a computer system and $\dot{\mathcal{F}}$ may express some relation between inputs and outputs which a designer would like the digital system to satisfy. The idea of a system $\mathcal{F}$ satisfying condition $\dot{\mathcal{F}}$ can be formulated in logical terms by saying, first, that $\mathcal{F}$ and $\dot{\mathcal{F}}$ have the same input variables and the same output variables, and second, that $\mathcal{F} \supset \dot{\mathcal{F}}$ is valid. Whenever the pair $\mathcal{F}$, $\dot{\mathcal{F}}$ satisfies these two conditions we will speak of formula $\mathcal{F}$ being "a solution of" formula $\dot{\mathcal{F}}$.

Let us see next how to formulate these ideas in sequence-generator terms. In passing between formulas and sequence generators, one must do some coding

or decoding, since the predicates of the formulas are two-valued, whereas a sequence generator has, in general, more than two states. If this coding is handled properly, the following definition of "Γ is a solution of $\dot{\Gamma}$" is essentially the same as the definition of "$\mathcal{F}$ is a solution of $\dot{\mathcal{F}}$" just given. Let $\Gamma = (S,G,R,P,Q)$ describe some digital system and let $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$ express a condition. P and $\dot{P}$ are interpreted as input projections and Q and $\dot{Q}$ as output projections. Γ is a solution of $\dot{\Gamma}$ whenever, first, $\mathcal{B}^{\omega}(\Gamma) \subseteq \mathcal{B}^{\omega}(\dot{\Gamma})$, and second, every $\dot{P}$-state occurring in $\mathcal{B}^{\omega}(\dot{\Gamma})$ occurs in $\mathcal{B}^{\omega}(\Gamma)$.

There are several kinds of design algorithms which make use of conditions. Büchi, Elgot, and Wright, 1958, define three kinds of algorithms for sets of formulas whose free predicate variables are divided into input variables and output variables. We will define analogous kinds of algorithms for sequence generators. Let α and β be two classes of two-projection sequence generators. A solution algorithm for $< \alpha,\beta >$ is a decision procedure which applies to any pair $< \Gamma,\dot{\Gamma} >$ such that $\Gamma \epsilon \alpha$ and $\dot{\Gamma} \epsilon \beta$ and answers the question: is Γ a solution of $\dot{\Gamma}$? A solvability algorithm for $< \alpha,\beta >$ applies to any $\dot{\Gamma} \epsilon \beta$ and is a decision procedure for the question: does there exist a $\Gamma \epsilon \alpha$ such that Γ is a solution of $\dot{\Gamma}$? A synthesis algorithm for $< \alpha,\beta >$ applies to any $\dot{\Gamma} \epsilon \beta$ and produces a $\Gamma \epsilon \alpha$ such that Γ is a solution of $\dot{\Gamma}$, if there is one.

Note that as a synthesis algorithm is here defined it may be non-terminating, but that a synthesis algorithm combined with a solvability algorithm is terminating, producing a sequence generator of the desired kind if one exists, terminating in a "no" otherwise. Note also that, if α can be recursively enumerated, the existence of a synthesis algorithm for $< \alpha,\beta >$ follows from the existence of a solution algorithm for $< \alpha,\beta >$, since the elements of α can be enumerated and each compared with the given element of β by the solution algorithm.

Clearly the Behavior Inclusion Procedure of Section 2.3 constitutes the

basis of a solution algorithm for any classes of sequence generators α,β. Now there are decision procedures for the sets of deterministic and semi-deterministic (Section 2.2), solvable (Section 2.3), h-univalent, and uniquely solvable (Section 3.3) sequence generators. Consequently, if α is any one of these sets it can be recursively enumerated, and so there is a synthesis algorithm for α and any set of sequence generators β.

For our next result we need the concept of one sequence generator being "a part of" another. $\Gamma = (S,G,R,P^1, \ldots, P^n)$ is a part of $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P}^1, \ldots, \dot{P}^n)$ whenever $S \subseteq \dot{S}$, $G \subseteq \dot{G}$, $R \subseteq \dot{R}$ and each P^i is $\dot{P}^i$ cut down to S. As an example of this concept we mention that the reduced form $\Gamma\dagger$ is a part of Γ. Clearly if Γ is a part of $\dot{\Gamma}$, $\beta(\Gamma) \subseteq \beta(\dot{\Gamma})$. The following is a theorem: let $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$ be semi-deterministic with respect to $\dot{P}x\dot{Q}$; there is a P-deterministic $\Gamma = (S,G,R,P,Q)$ which is a solution of $\dot{\Gamma}$ if and only if there is a P-deterministic $\Gamma = (S,G,R,P,Q)$ which is a part of $\dot{\Gamma}$ and such that every $\dot{P}$-state occurring in $\beta^{\omega}(\dot{\Gamma})$ occurs in $\beta^{\omega}(\Gamma)$.

There is a proof of this theorem which consists of two steps. In the first step the theorem is established for the special case where no two complete states of the given condition $\dot{\Gamma}$ agree on both of their projections. In terms of formulas (Section 4.2), this means that the theorem is proved for conditions which are M-formulas. The second step is to extend this justification to ΣM-formulas, that is, to extend the proof of the theorem to the general case, where the given condition $\dot{\Gamma}$ is an arbitrary two-projection sequence generator. The second step of the proof involves a construction which is of some interest in its own right. Let $\Gamma = (S,G,R,P)$ and $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P})$. The cross product of these two sequence generators $\Gamma \times \dot{\Gamma} = \ddot{\Gamma}$ $(\ddot{S},\ddot{G},\ddot{R},\ddot{P})$ is defined by

$$\ddot{S} = \{< s,\dot{s} >| s \in S \;\&\; \dot{s} \in \dot{S} \;\&\; P(s) = \dot{P}(\dot{s})\}$$

$$\ddot{G} = (G \times \dot{G}) \cap \ddot{S}$$

$$\ddot{R}(< s_1, \dot{s}_1 >, < s_2, \dot{s}_2 >) = [R(s_1, s_2) \;\&\; \dot{R}(\dot{s}_1, \dot{s}_2)]$$

$$\ddot{P}(< s, \dot{s} >) = P(s).$$

The preceding construction can be used to establish the following lemma: Let $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$ be a sequence generator which is semi-deterministic with regard to $\dot{P}\times\dot{Q}$ and $\dddot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{I})$, where $\dot{I}(\dot{s}) = \dot{s}$. There is a P-deterministic sequence generator $\Gamma = (S,G,R,P,Q)$ which is a solution of $\dot{\Gamma}$ if and only if there is a P-deterministic $\ddot{\Gamma}$ which is a solution of $\dddot{\Gamma}$.

This theorem leads directly to a combined solvability-synthesis algorithm for $< \alpha,\beta >$, where α is the class of all two-projection sequence generators which are deterministic with respect to the first projection, and β is any class of two-projection sequence generators; the essential steps of the algorithm are to apply the subset sequence generator operation to the given condition and to examine each part of the result for determinism. By the results of Section 4.2, this gives us a combined solvability-synthesis algorithm for $< \alpha,\beta >$, where α is the class of systems of restricted recursions (i.e., representations of deterministic information processing systems) and β is any class of ΣA-formulas, the free predicate variables of all formulas being divided into input variables and output variables. This extends a result of Church, 1960, pp. 33a ff., 36 ff., which is a solvability-synthesis algorithm for $< \alpha,\beta >$, where α is the class of systems of restricted recursions but β is any class of A-formulas.

Wang, 1959, p. 312, Theorem 6, has a result which is (in our terms) a solvability-synthesis algorithm for $< \alpha,\beta >$, where α consists of systems of unrestricted singulary recursions and β is any set of 0-formulas. Our last stated lemma holds with "uniquely solvable" replacing "P-deterministic."

By applying this lemma, Wang's result can be extended to provide a solvability-synthesis algorithm for $< \alpha,\beta >$, where α consists of systems of unrestricted singulary recursions and β is any set of ΣA-formulas, the free predicate variables of all formulas being divided into input variables and output variables, as before.

We give next an example (already to be found in the literature) of the use of conditions. Sometimes an automaton is given with "input restrictions" by drawing an internal state diagram which does not provide transitions for all inputs (Aufenkamp and Hohn, 1957, Section IV). One is then interested in an automaton which has the same behavior as this diagram with respect to all input sequences which are provided for. This situation may be described in sequence-generator terms. The internal state diagram with input restrictions may be converted into a $\Gamma = (S,G,R,P,Q)$ which is semi-deterministic with respect to P (see Section 1.2); in fact, the semi-determinism is not required for what we are going to do. We define a $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P},\dot{Q})$ which will have Γ as a part. Let B consist of complete states $< p,q >$ for all P-states p and Q-states q. $\dot{S} = S \cup B$. $\dot{R}$ is R extended so that (1) $\dot{R}(\dot{s}_1, \dot{s}_2)$ for all $\dot{s}_1, \dot{s}_2 \in B$ and (2) for any $s \in S$ and any P-state p, if there is no $s_1 \in S$ such that $R(s, s_1)$ and $P(s_1) = p$, then $\dot{R}(s, \dot{s})$ for every $\dot{s} \in B$. $\dot{P}$ and $\dot{Q}$ are P and Q extended to cover the elements of B so that $\dot{P}(< p,q >) = p$ and $\dot{Q}(< p,q >) = q$. The original problem now becomes that of finding a $\ddot{P}$-deterministic $\ddot{\Gamma} = (\ddot{S},\ddot{G},\ddot{R},\ddot{P},\ddot{Q})$ which is a solution of $\dot{\Gamma}$ (cf. the penultimate paragraph of Section 2.1). An answer will always be given by the solvability-synthesis algorithm for deterministic sequence generators described two paragraphs back.

The solution, solvability, and synthesis algorithms we have been discussing are defined in terms of behavior. It is worth noting that other solution, solvability, and synthesis algorithms may be defined by using different concepts in place of behavior in the definition of "Γ is a solution of $\dot{\Gamma}$";

for example, the set of Γ-sequences of a sequence generator, the computation of a sequence generator (Section 4.1), or the behavior of the final state sequence generators to be introduced next. We do not have space to discuss these algorithms.

4.4. SEQUENCE GENERATORS AND REGULARITY

We now extend the concept of sequence generator to include a set F of final states. $\Gamma = (S,G,F,R,P)$ is defined as in Section 1.1, with the additional proviso that $F \subseteq S$. The definition of Γ-sequence is altered to require that the last complete state of a finite Γ-sequence belong to F (be a final state) and to require that an infinite Γ-sequence contain infinitely many occurrences of members of F. (Contrast the concept of an infinite computation element of Section 4.1 and note that the functions of the sets G and F could be performed by two two-valued projections; for an illustration of this, see the last example of Section 1.3.) Behavior is defined as in Section 2.1, using the new concept of Γ-sequence. It will be convenient to have a concept of finite behavior $\mathfrak{B}^f$ for both ordinary and final state sequence generators: $\mathfrak{B}^f(\Gamma) = \mathfrak{B}(\Gamma) - \mathfrak{B}^\omega(\Gamma)$. We will assume that those earlier concepts (e.g., determinism) which are easily extended to cover sequence generators with final states are in fact so extended.

Final-state sequence generators are of interest in connection with "regular sets." A regular set is a set of finite sequences defined in terms of certain algebraic operations on a given finite set of finite sequences. It has been shown that the following three sets are equivalent:

(1) The set of regular sets

(2) The set of finite behaviors $\mathfrak{B}^f(\Gamma)$ of final state sequence generators $\Gamma = (S,G,F,R,P)$.

(3) The set of finite behaviors $\mathfrak{B}^f(\Gamma)$ of final-state sequence generators

$\Gamma = (S,G,F,R,P)$ which are deterministic.

See Kleene, 1956, Copi, Elgot, and Wright, 1958, and Myhill, 1957.

The concept of a final-state sequence generator is a bona fide generalization of the concept of a sequence generator without final states. For every sequence generator (S,G,R,P) of the latter kind there is a behaviorally equivalent sequence generator of the former variety, namely, (S,G,S,R,P). But not every behavior of a final-state sequence generator is the behavior of some sequence generator without final states. This may be shown by means of the concept of "open behavior." A behavior (finite or unrestricted) of a sequence generator is said to be "open" if every initial segment of an element of the behavior belongs to the behavior. The Infinity Theorem (2.1-3) implies that the behavior of a sequence generator without final states is open. The corresponding theorem does not hold for final-state sequence generators, as the simple example of Fig. 17 shows; $\mathcal{B}(\Gamma)$ consists of the single sequence p_0,p_0 and hence is not open. Clearly, then, no sequence generator without final states has the behavior of Fig. 17.

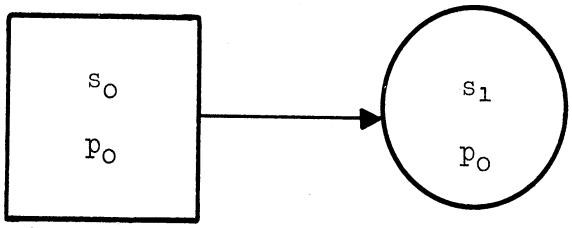


Fig. 17. Final-state sequence generator $\Gamma = (S,G,F,R,P)$. $F = \{s_1\}$. $\mathcal{B}(\Gamma)$ is not open.

Though the behavior of a final-state sequence generator is not in general the behavior of a sequence generator without final states, any open finite behavior of a final-state sequence generator is the finite behavior of some sequence generator without final states. That is, for any $\Gamma = (S,G,F,R,P)$, if $\mathcal{B}^f(\Gamma)$ is open, then there is a $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P})$ such that $\mathcal{B}^f(\dot{\Gamma}) = \mathcal{B}^f(\Gamma)$. Given any $\Gamma = (S,G,F,R,P)$, whose $\mathcal{B}^f(\Gamma)$ is open, the desired $\dot{\Gamma}$ may be constructed

as follows. By the results of paragraph two of this subsection, there exists a $\ddot{P}$-deterministic sequence generator $\ddot{\Gamma} = (\ddot{S},\ddot{G},\ddot{F},\ddot{R},\ddot{P})$ whose $\mathcal{B}^f(\ddot{\Gamma}) = \mathcal{B}^f(\Gamma)$. Since $\mathcal{B}^f(\Gamma)$ is open, $\mathcal{B}^f(\ddot{\Gamma})$ is open. Now construct $\dot{\Gamma} = (\ddot{S},\dot{G},\dot{R},\ddot{P})$, where $\dot{G} = \ddot{G} \cap \ddot{F}$ and $\dot{R}$ is $\ddot{R}$ cut down to $\ddot{F} \times \ddot{F}$. It can be shown from the openness of $\mathcal{B}^f(\ddot{\Gamma})$ and the determinism of $\ddot{\Gamma}$ that every complete state which belongs to a $\dot{\Gamma}$-sequence is an $\ddot{F}$. Consequently, $\mathcal{B}^f(\dot{\Gamma}) = \mathcal{B}^f(\ddot{\Gamma})$. Hence $\mathcal{B}^f(\dot{\Gamma}) = \mathcal{B}^f(\Gamma)$, and $\dot{\Gamma}$ is the desired sequence generator without final states.

The result just established for open finite behaviors does not hold for open behaviors in general. That is, there is a $\Gamma = (S,G,F,R,P)$ with open $\mathcal{B}(\Gamma)$ for which there is no behaviorally equivalent sequence generator without final states. An example is given in Fig. 18. $\mathcal{B}(\Gamma)$ consists of all finite sequences of p_o and is thus open. But if the behavior of a sequence generator without final states contains all finite sequences of p_o, by the Infinity Theorem (2.1-3) it must also contain the infinite sequence $p_o, p_o, p_o, \ldots$. Hence $\mathcal{B}(\Gamma)$ is not the behavior of any sequence generator without final states.

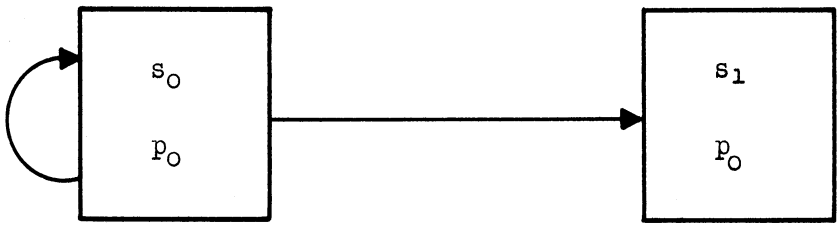


Fig. 18. Final-state sequence generator $\Gamma = (S,G,F,R,P)$. $F = \{s_1\}$. $\mathcal{B}(\Gamma)$ is open, but it is not the behavior of any sequence generator without final states.

We will conclude this subsection with a discussion of Boolean operations on sequence generator behaviors. Let $\mathcal{P}(\Gamma)$, $\mathcal{P}^f(\Gamma)$, $\mathcal{P}^\omega(\Gamma)$ be the sets of all, all finite, all infinite sequences of P-states, respectively. The complement of a behavior is defined with respect to the appropriate one of these: $\sim\mathcal{B}(\Gamma) = \mathcal{P}(\Gamma) - \mathcal{B}(\Gamma)$, $\sim\mathcal{B}^f(\Gamma) = \mathcal{P}^f(\Gamma) - \mathcal{B}^f(\Gamma)$, and $\sim\mathcal{B}^\omega(\Gamma) = \mathcal{P}^\omega(\Gamma) - \mathcal{B}^\omega(\Gamma)$.

Consider first sequence generators without final states. Let $\ddot{\Gamma}$ be the

sequence generator represented by the state diagram obtained by juxtaposing the state diagrams for Γ and $\dot{\Gamma}$, treating the complete states of Γ and $\dot{\Gamma}$ as distinct. Then clearly

$$\begin{aligned} \mathcal{B}(\ddot{\Gamma}) &= \mathcal{B}(\Gamma) \cup \mathcal{B}(\dot{\Gamma}) \\ \mathcal{B}^f(\ddot{\Gamma}) &= \mathcal{B}^f(\Gamma) \cup \mathcal{B}^f(\dot{\Gamma}) \\ \mathcal{B}^\omega(\ddot{\Gamma}) &= \mathcal{B}^\omega(\Gamma) \cup \mathcal{B}^\omega(\dot{\Gamma}). \end{aligned}$$

Let $\ddot{\Gamma} = \Gamma \times \dot{\Gamma}$, where "x" is the cross product operation defined in Section 4.3. It may be proved that

$$\begin{aligned} \mathcal{B}(\ddot{\Gamma}) &= \mathcal{B}(\Gamma) \cap \mathcal{B}(\dot{\Gamma}) \\ \mathcal{B}^f(\ddot{\Gamma}) &= \mathcal{B}^f(\Gamma) \cap \mathcal{B}^f(\dot{\Gamma}) \\ \mathcal{B}^\omega(\ddot{\Gamma}) &= \mathcal{B}^\omega(\Gamma) \cap \mathcal{B}^\omega(\dot{\Gamma}). \end{aligned}$$

Thus the union and intersection of sequence-generator behaviors are always sequence-generator behaviors. But the complement of a sequence-generator behavior is not generally a sequence-generator behavior. This is shown by Fig. 19; it can be proved by means of the openness of the behavior of a sequence generator without final states and the Infinity Theorem (2.1-3) that no sequence generator without final states has $\sim\mathcal{B}(\Gamma)$, $\sim\mathcal{B}^f(\Gamma)$, or $\sim\mathcal{B}^\omega(\Gamma)$ as its behavior.

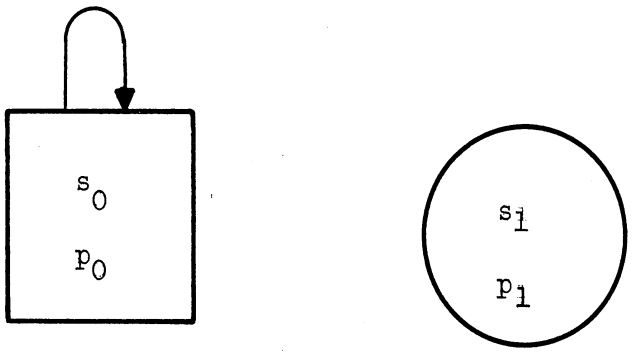
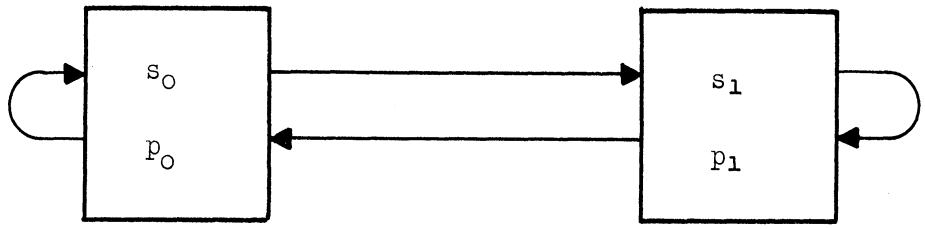


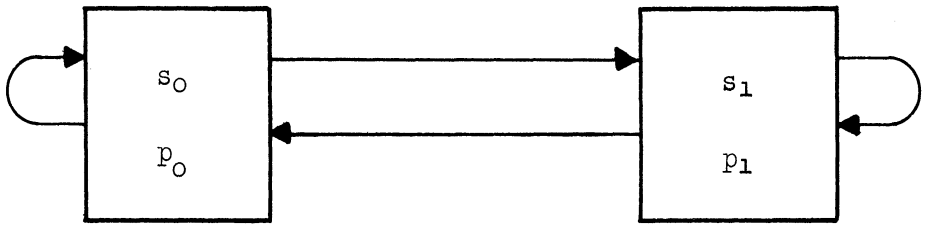
Fig. 19. $\Gamma = (S,G,R,P)$. No sequence generator without final states has $\sim\mathcal{B}(\Gamma)$, $\sim\mathcal{B}^f(\Gamma)$, or $\sim\mathcal{B}^\omega(\Gamma)$ as its behavior.

The situation is very different with final-state sequence generators. The class of finite behaviors of sequence generators with final states is closed under union, intersection, and complement (Rabin and Scott, 1959). Consider next infinite behaviors. In Section 4.2 we showed how to pass between various kinds of formulas and sequence generators without final states. There are also formulas which correspond to sequence generators with final states. J. Richard Büchi (unpublished) discusses such formulas which he calls "quasi-Σ_1-formulas" and establishes that the complement of a set represented by a quasi-Σ_1-formula may be represented by a quasi-Σ_1-formula. The same result is, in sequence-generator terms, that for every $\Gamma = (S,G,F,R,P)$ there is a $\dot{\Gamma} = (\dot{S},\dot{G},\dot{F},\dot{R},\dot{P})$ such that $\mathcal{B}^{\omega}(\dot{\Gamma}) = \sim \mathcal{B}^{\omega}(\Gamma)$. As before, if $\ddot{\Gamma}$ is the result of juxtaposing Γ and $\dot{\Gamma}$, $\mathcal{B}^{\omega}(\ddot{\Gamma}) = \mathcal{B}^{\omega}(\Gamma) \cup \mathcal{B}^{\omega}(\dot{\Gamma})$. Hence by De Morgan's theorem, the class of infinite behaviors is closed under union, intersection, and complement. Consider finally (complete) behaviors. As before, if $\ddot{\Gamma}$ is the result of juxtaposing Γ and $\dot{\Gamma}$, $\mathcal{B}(\ddot{\Gamma}) = \mathcal{B}(\Gamma) \cup \mathcal{B}(\dot{\Gamma})$. However, the intersection of the behaviors of sequence generators with final states is not always a sequence-generator behavior, and hence by De Morgan's theorem the complement of the behavior of a sequence generator with final states is not always a sequence-generator behavior. This is shown by Fig. 20. $\mathcal{B}(\Gamma)$ of Fig. 20(a) consists of all finite sequences terminating in p_1 and all infinite sequences with infinitely many occurrences of p_1. $\mathcal{B}(\dot{\Gamma})$ of Fig. 20(b) consists of all finite sequences terminating in p_0 and all infinite sequences with infinitely many occurrences of p_0. Hence $\mathcal{B}(\Gamma) \cap \mathcal{B}(\Gamma)$ contains only infinite sequences. But no sequence-generator behavior contains only infinite sequences, and therefore no sequence generator has the behavior $\mathcal{B}(\Gamma) \cap \mathcal{B}(\dot{\Gamma})$.

Section 2.3 contains a decision procedure for the inclusion of the behavior of one sequence generator without final states in the behavior of another. This is also a procedure for the inclusion of finite behaviors and



(a)



(b)

Fig. 20. (a) $\Gamma = (S,G,F,R,P)$. $F = \{s_1\}$.
(b) $\dot{\Gamma} = (S,G,\dot{F},R,P)$. $\dot{F} = \{s_0\}$. No sequence generator has the behavior $\mathcal{B}(\Gamma) \cap \mathcal{B}(\dot{\Gamma})$.

a simple modification of it gives a procedure for the inclusion of infinite behaviors. It follows from the fact that both the class of finite behaviors and the class of infinite behaviors of final state sequence generators are effectively closed under union and complement that there are decisions for "Is $\mathcal{B}^{\omega}(\Gamma) \subseteq \mathcal{B}^{\omega}(\Gamma)$?", "Is $\mathcal{B}^{f}(\Gamma) \subseteq \mathcal{B}^{f}(\dot{\Gamma})$?", "Is $\mathcal{B}(\Gamma) \subseteq \mathcal{B}(\dot{\Gamma})$?", where Γ and $\dot{\Gamma}$ are final-state sequence generators.

4.5. INFINITE-SEQUENCE GENERATORS

In this subsection we will discuss briefly a generalization of the original concept of sequence generator obtained by dropping the requirement that the set of complete states S be finite (Section 1.1). The resultant generalization is called an "infinite-sequence generator."

Actually much of the content of this paper applies to infinite-sequence generators as well as finite ones. All our concepts apply to infinite-sequence generators except for the various decision procedures and the reduced form $\Gamma^{\dagger}$, and it is easy to define the reduced form of an infinite-sequence generator.

Many of our theorems apply to infinite-sequence generators too, with only minor modifications being needed in the proofs we have given to cover this extension. Of course the various decision procedures we have given make essential use of the finitude of the number of complete states of sequence generator, and the Time-shift Theorem (3.3-2) applies only to finite-sequence generators.

In the first part of this paper we have made implicit use of infinite sequence generators. It will be seen on examination that Lemma 1.3-2 is in fact about an infinite-sequence generator (S, δ_0, ρ), where S is any set (finite or infinite) on which ρ is defined. The following theorem is very close to Lemma 2.1-2. Let $\Gamma = (S,G,R)$ be an infinite-sequence generator with G finite and R(s) finite for every $s \in S$; if every set in the ω-sequence $G, R(G), R^2(G), R^3(G), \ldots$ is non-empty, then there is an infinite Γ-sequence. By means of this lemma the proof of the Infinity Theorem (2.1-3) can be rewritten, with minor modifications, to yield the following extension of the Infinity Theorem: Let $\Gamma = (S,G,R,P)$ be an infinite sequence generator with G finite and R(s) finite for every $s \in S$; then any sequence of P-states belongs to $B(\Gamma)$ if and only if every finite initial segment of it belongs to $B(\Gamma)$.

The tree operator to be defined next was used implicitly in the proof of the Infinity Theorem as well as in step (2) of the Reduced Form Algorithm. The tree operator "t" applies to any (infinite or finite) sequence generator $\Gamma = (S,G,R,P^1, \ldots P^n)$ and produces its "tree generator" $\Gamma^t = \dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{P}^1, \ldots \dot{P}^n)$:

$\dot{S}$ = the set of finite Γ-sequences

$\dot{G} = G$

$\dot{R}(\dot{s}, [s](0,j+1)) \equiv [([s](0,j+1) \in \dot{S}) \;\&\; (\dot{s} = [s](0,j))]$

$P^i([s](0,k)) = P^i(s(k))$, for $i = 1, \ldots, n$.

Note that if R(s) is always finite, then R($\dot{s}$) is always finite. Moreover, if S is finite, then $\dot{S}$ is infinite if and only if Γ has a Γ-admissible complete state.

We conclude with some examples of infinite-sequence generators which have already been discussed in the literature, though not under this name. A Turing machine as defined by Turing, 1936-37, consists of a finite automaton to which is attached an infinite tape; odd-numbered squares are used for writing the digits of a real number. Such a machine corresponds to a sequence generator (S,G,R,Q), where S is the set of states of the machine (including its tape), G is the set of initial states of the machine, R is given by the transition rule, and the output projection Q when applied to a complete state s gives the number written on the odd-numbered squares of the tape when the machine is in that state. R(s) is always a unit set. For the special-purpose machine (whose tape is initially blank), G is a unit set. For the universal machine, G is an infinite set consisting of those states whose tape part represents a program for a special purpose machine. The concept of computation (Section 4.1) can be extended to infinite-sequence generators. In particular, this can be done for Turing machines in such a way that two machines (one of these may be universal) are computationally equivalent if and only if they compute the same number in Turing's sense. This fact was part of the motivation for defining the concept of computation inasmuch as a similar statement cannot be made in terms of behavior.

The basis von Neumann uses for his construction of a self-reproducing automaton consists of an infinite number of cells each capable of 29 states, with the state of each cell at time t + 1 determined by the states of itself and its neighbors at time t (Shannon, 1953; Burks, 1960, Section 4). This basis corresponds to a sequence generator (S,G,R), with S and G both infinite, and R(s) finite but unbounded. Burks, 1959, Section 6, Church, 1960, p. 21 ff.,

and Holland, 1960, have given definitions of infinite automata with inputs and outputs. These correspond to infinite-sequence generators with two projections; in the latter two formulations, infinitely many input states are allowed, with the consequence that R(s) is infinite. Any Post canonical language (Post, 1943) in which each production rule has only one premise may be represented by an infinite-sequence generator (S,G,R). S is the set of strings, G is the set of axioms, and R is given by the production rules.

Although we will not attempt to give a definition of "effective sequence generator," it should be noted that all the examples of infinite-sequence generators given above are effective in the sense that in each case integers may be assigned to the states in such a way that S,G,R, and the P^i's are all recursive.

4.6. PROBABILISTIC SEQUENCE GENERATORS

For the sake of completeness we will discuss the relation of probabilistic to non-probabilistic sequence generators before concluding this paper.

Let $\Gamma = (S,G,R,P^1, \ldots, P^n)$, $n = 0, 1, 2, \ldots$, be an infinite-sequence generator with non-null G. Let W be a weight function which assigns positive initial probabilities summing to one to the elements of G and, for each $s \in S$, assigns positive transition (conditional) probabilities summing to one to the elements of $\{< s, s_1 > | R(s, s_1)\}$, provided this set is non-null.[5] $(S,G,R,W,P^1, \ldots, P^n)$ is a "probabilistic sequence generator." Note that, as we use the terms, "probabilistic" and "deterministic" are not contradictories. "Deterministic" was defined in Section 2.1 for non-probabilistic sequence generators; its opposite is "indeterministic," not "probabilistic." Moreover, given a deterministic sequence generator (S,G,R,P) one can form a

[5]There is an alternative concept which is more difficult to define but easier to use in some applications: for each P-state p the initial probabilities sum to one over $\{s | s \in G \ \& \ P(s) = p\}$, and for each p & $s \in S$ the transition probabilities sum to one over $\{< s, s_1 > | R(s,s_1) \ \& \ P(s_1) = p\}$.

probabilistic sequence generator (S,G,R,W,P) from it by adding a weight function W.

The weight function W induces a probability distribution on the finite Γ-sequences of Γ = (S,G,R,P) and hence on the elements of $\mathcal{B}^f(\Gamma)$. The probability of a finite Γ-sequence [s](0,k) is the initial probability of s(0) multiplied by the probabilities of the transitions $< s(0), s(1) >, < s(1), s(2) >, \ldots, < s(k-1), s(k) >$. The probability of any element of $\mathcal{B}^f(\Gamma)$ is the sum of the probabilities of those Γ-sequences which produce that behavior element.

A stochastic process in which the probability of a state occurring at time t depends only on the preceding states of the sequence can be represented by a sequence generator whose states are finite sequences of states of the stochastic process; compare the tree sequence generator Γ^t of Section 4.5. A Markov chain with constant transition probabilities is a probabilistic sequence generator without projections (S,G,R,W) such that (S,G,R) has no terminal states (see, for example, Feller, 1957, p. 340). If S is finite, then (S,G,R,W) is a finite Markov chain.

It is worth noting that many of the concepts employed in analyzing a finite Markov chain (S,G,R,W) depend only on the underlying non-probabilistic sequence generator (S,G,R) and not on the probability function W. We will give some examples. s is an absorbing state if and only if R(s) = {s}. (It should be recalled in this connection that every probability assigned by W is positive; hence if the probability of s succeeding s is one, then R(s) contains only s.) Let $R^\omega(s_1, s_2) \equiv s_2 \in \bigcup_{i=1}^{\omega} R^i(s_1)$. s is a transient state if and only if there is a state $s_1 \in S$ such that $R^\omega(s, s_1)$ but not $R^\omega(s_1, s)$; s is a persistent state if s is not transient. A sequence generator (S,G,R) which satisfies these two conditions is ergodic: first, for any two complete states $s_1, s_2 \in S$, $R^\omega(s_1, s_2)$ and $R^\omega(s_2, s_1)$, and second, the greatest common

divisor of the lengths of all cycles of complete states is one (cf. Shannon, 1948, p. 435).

It was remarked immediately after Corollary 2.1-1 that the concept of a sequence generator with projections is a bona fide generalization of the concept of a sequence generator without projections; for example, the sequence 001001001 ... belongs to the behavior of a sequence generator but it is not a Γ-sequence of any sequence generator (cf. the discussion of various ways of generalizing sequence generators in Section 4.2). Similarly, probabilistic sequence generators with projections are bona fide extensions of probabilistic sequence generators without projections. In any probabilistic sequence generator (S,G,R,W,P) the probability of a given complete state occurring at any time depends only on what complete state occurred at the preceding time. This is not so for the P-states, for since the same P-state may be assigned to different complete states the occurrence of a P-state at t can be made to depend on what P-states occurred at times prior to t - 1. An example is given in Fig. 21; the probability that p_0 will occur, given that the three preceding P-states are p_1, p_0, p_0, in that order, is 0.8, while the probability that p_0 will occur, given that the three preceding states are p_1, p_1, p_0, in that order, is 0.5.

We give some examples of probabilistic sequence generators. Shannon, 1948, p. 384 ff., defines a "discrete information source." A discrete information source is a finite Markov chain $\Gamma = (S,G,R,W)$ to which has been added a set of symbols, each transition of Γ producing one of these symbols as outputs. One can construct another sequence generator $\dot{\Gamma} = (\dot{S},\dot{G},\dot{R},\dot{W},\dot{P})$, where $\dot{S}$ consists of pairs of elements of S and $\dot{P}$ is an output projection, such that $\dot{\Gamma}$ will produce the same output sequences with the same probabilities as the original discrete information source.

Von Neumann, 1956, p. 61 ff., investigates finite probabilistic nets

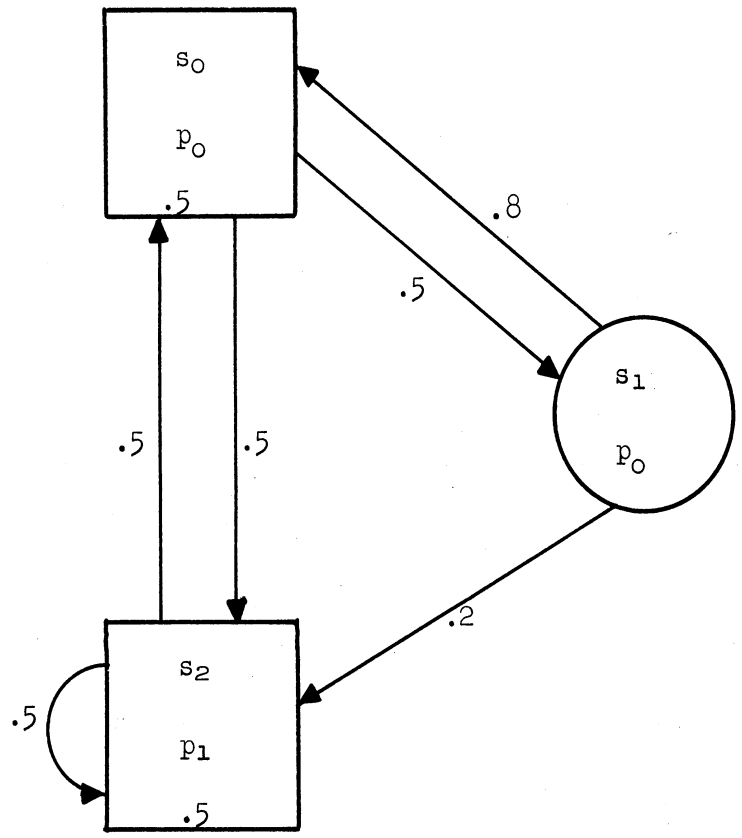


Fig. 21. Probabilistic sequence generator (S,G,R,W,P).

(cf. Moore and Shannon, 1956). These are well-formed nets composed of combined switch-delay elements. An element produces the correct (desired) output at each time with probability $1 - \epsilon$ and the complement (incorrect) output with probability ϵ. Thus in Fig. 22(a), the output B is defined probabilistically as

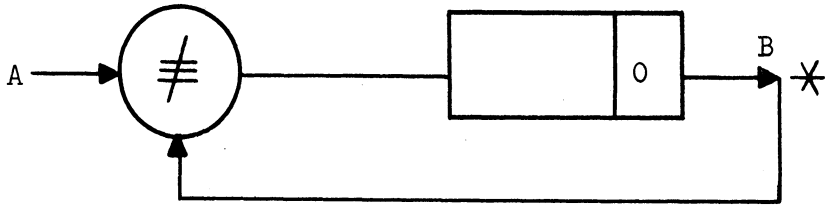
$$B(0) \text{ with probability } 1 - \epsilon, \ \overline{B(0)} \text{ with probability } \epsilon$$

$$B(t+1) \equiv [A(t) \not\equiv B(t)] \text{ with probability } 1 - \epsilon,$$

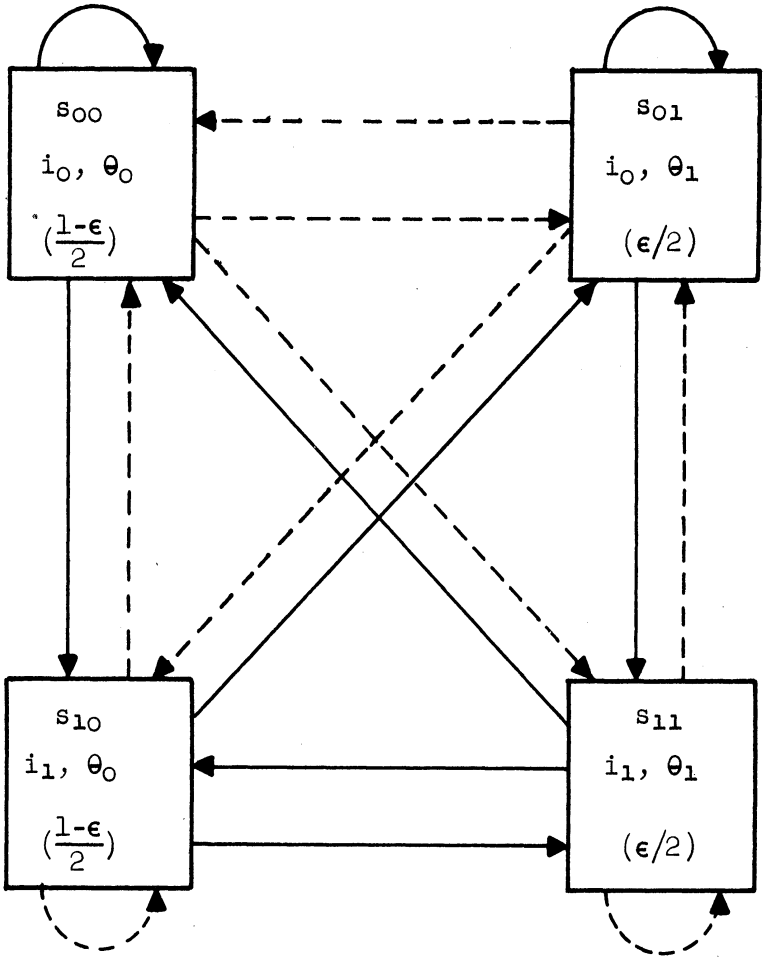
$$B(t+1) \not\equiv [A(t) \not\equiv B(t)] \text{ with probability } \epsilon.$$

Given a probabilistic net, one can derive a probabilistic sequence generator from it by methods similar to those of Section 1.2. Figure 22(b) is the probabilistic sequence generator (S,G,R,W,I,Θ) for the binary counter of Fig. 22(a); I is the input projection and Θ is the output projection. The solid

lines represent the desired (correct) transitions, the dotted lines the erroneous transitions; cf. Fig. 1. Note that (S,G,R,I,Θ) of Fig. 22(b) is not deterministic. In a similar way, a probabilistic sequence generator (S,G,R,W,Q), may be obtained from a probabilistic Turing machine; here Q is the output projection, S is infinite; see de Leeuw et al., 1956.



(a)



(b)

Fig. 22. (a) Binary counter (probabilistic). (b) Probabilistic sequence generator (S,G,R,W,I,Θ) for binary counter (a). Solid lines represent transitions with probability $(1 - \epsilon)/2$. Dotted lines represent transitions with probability $\epsilon/2$.

BIBLIOGRAPHY

Aufenkamp, D. D., and Hohn, F. E., "Analysis of Sequential Machines," Institute of Radio Engineers, Transactions on Electronic Computers, 1957, EC-6, 276-285.

Büchi, J. R., Elgot, C. C., and Wright, J. B., "Non-Existence of Certain Algorithms of Finite Automata Theory," Abstract, Notices of the American Mathematical Society, April 1958, 5, No. 2, issue 30.

Burks, A. W., "Computation, Behavior, and Structure in Fixed and Growing Automata," in M. Yovits and S. Cameron (eds.), Self-Organizing Systems, Pergamon Press, New York, 1960, pp. 282-311.

Burks, A. W., "The Logic of Fixed and Growing Automata," Proceedings of an International Symposium on the Theory of Switching, 2-5 April 1957, Harvard Univ. Press, Cambridge, 1959, Part I, pp. 147-188.

Burks, A. W., and Wang, H., "The Logic of Automata," Journal of the Association for Computing Machinery, 1957, 4, 193-218, 279-297.

Burks, A. W., and Wright, J. B., "Theory of Logical Nets," Proceedings of the Institute of Radio Engineers, 1953, 41, 1357-1365.

Chomsky, N., and Miller, G. A., "Finite State Languages," Information and Control, 1958, 1, 91-112.

Church, A., "Application of Recursive Arithmetic to the Problem of Circuit Synthesis," Summaries of talks presented at the Summer Institute for Symbolic Logic, Cornell University, 1957, Institute for Defense Analysis, Princeton, 1960.

Church, A., Review of Edmund C. Berkeley: "The Algebra of States and Events," The Journal of Symbolic Logic, 1955, 20, 286-287.

Copi, I. M., Elgot, C. C., and Wright, J. B., "Realization of Events by Logical Nets," Journal of the Association for Computing Machinery, 1958, 5, 181-196.

de Leeuw, K., Moore, E. F., Shannon, C. E., and Shapiro, N., "Computability by Probabilistic Machines," in C. E. Shannon and J. McCarthy (eds.), Automata Studies, Princeton Univ. Press, Princeton, 1956, pp. 183-212.

Feller, W., An Introduction to Probability Theory and Its Applications, 2nd Edition, Wiley, New York, 1957.

Fitch, F. B., "Representation of Sequential Circuits in Combinatory Logic," Philosophy of Science, 1958, 25, 263-279.

Harary, F., and Paper, H. H., "Toward a General Calculus of Phonemic Distribution," Language, 1957, 33, 143-169.

Heyting, A., Intuitionism, an Introduction, North Holland, Amsterdam, 1956.

Holland, J. H., "Iterative Circuit Computers," Proceedings of the 1960 Western Joint Computer Conference, 1960.

Kleene, S. C., "Representation of Events in Nerve Nets and Finite Automata," in C. E. Shannon and J. McCarthy (eds.), Automata Studies, Princeton Univ. Press, Princeton, 1956, pp. 3-41.

König, D., Theorie der Endlichen und unendlichen Graphen, Akademische Verlagsgesellschaft M.B.H., Leipzig, 1936.

Mealy, G. H., "A Method for Synthesizing Sequential Circuits," The Bell System Technical Journal, 1955, 34, 1045-1079.

McKinsey, J. C. C., Introduction to the Theory of Games, McGraw-Hill, New York, 1952.

Medvedev, I. T., "On a Class of Events Representable in a Finite Automaton," translated by J. J. Schorr-Kon from a supplement to the Russian translation of Automata Studies, C. E. Shannon and J. McCarthy (eds.), Group Report 34-73, Lincoln Laboratory, Lexington, Mass., 1958.

Moore, E. F., "Gedanken Experiments on Sequential Machines," in C. E. Shannon and J. McCarthy (eds.), Automata Studies, Princeton Univ. Press, Princeton, 1956, pp. 129-153.

Moore, E. F., and Shannon, C. E., "Reliable Circuits Using Less Reliable Relays," Journal of the Franklin Institute, 1956, 262, 191-208, 281-287.

Myhill, J., "Finite Automata and Representation of Events," in Fundamental Concepts in the Theory of Systems, WADC Technical Report 57-624, ASTIA Document No. AD 1557 41, 1957.

Post, E. L., "Formal Reductions of the General Combinatorial Decision Problem," American Journal of Mathematics, 1943, 65, 197-215.

Putnam, H., "Decidability and Essential Undecidability," The Journal of Symbolic Logic, 1957, 22, 39-54.

Rabin, M. O., and Scott, D., "Finite Automata and Their Decision Problems," IBM Journal of Research & Development, 1959, 3, 114-125.

Shannon, C. E., "Computers and Automata," Proceedings of the Institute of Radio Engineers, 1953, 41, 1235-1241.

Shannon, C. E., "A Mathematical Theory of Communication," The Bell System Technical Journal, 1948, 27, 379-423, 623-656.

Turing, A. M., "On Computable Numbers, with an Application to the Entscheidungsproblem," Proceedings of the London Mathematical Society, Series 2, 1936, 42, 230-265, and 1937, 43, 544-546.

von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," in C. E. Shannon and J. McCarthy (eds.), Automata Studies, Princeton Univ. Press, Princeton, 1956, pp. 43-98.

von Neumann, J., "The General and Logical Theory of Automata," Cerebral Mechanisms in Behavior, Wiley, New York, 1951, pp. 1-41.

Wang, H., "Circuit Synthesis by Solving Sequential Boolean Equations," Zeitschrift für mathematische Logik und Grundlagen die Mathematik, 1959, 5, 291-322.

DISTRIBUTION LIST
(one copy unless otherwise noted)

Assistant Sec. of Def. for (2)
Res. and Eng.
Information Office Library Branch
Pentagon Building
Washington 25, D. C.

Armed Services Technical Informa-(10)
tion Agency
Arlington Hall Station
Arlington 12, Virginia

Chief of Naval Research (2)
Department of the Navy
Washington 25, D. C.
Attn: Code 437, Information
Systems Branch

Chief of Naval Operations
OP-07T-12
Navy Department
Washington 25, D. C.

Director, Naval Research (6)
Laboratory
Technical Information Officer
Washington 25, D. C.
Attn: Code 2000

Commanding Officer (10)
Office of Naval Research
Navy No. 100, Fleet Post Office
New York, New York

Commanding Officer
ONR Branch Office
346 Broadway
New York 13, New York

Commanding Officer
ONR Branch Office
495 Summer Street
Boston 10, Massachusetts

Office of Technical Services
Technical Reports Section
Department of Commerce
Washington 25, D. C.

Bureau of Ships
Department of the Navy
Washington 25, D. C.
Attn: Code 671 NTDS

Chief, Bureau of Ships
Department of the Navy
Washington 25, D. C.
Attn: Code 280

Chief, Bureau of Ships
Department of the Navy
Washington 25, D. C.
Attn: Code 687E

Naval Ordnance Laboratory
White Oaks
Silver Spring 19, Maryland
Attn: Technical Library

David Taylor Model Basin
Washington 7, D. C.
Attn: Technical Library

Naval Electronics Laboratory
San Diego 52, California
Attn: Technical Library

University of Illinois
Control Systems Laboratory
Urbana, Illinois
Attn: D. Alpert

University of Illinois
Digital Computer Laboratory
Urbana, Illinois
Attn: Dr. J. E. Robertson

Technical Information Officer
U.S. Army Signal Research
and Dev. Lab.
Fort Monmouth, New Jersey
Attn: Data Equipment Branch

Director (3)
National Security Agency
Fort Geo. G. Meade, Maryland
Attn: Chief, REMP

Naval Proving Ground
Dahlgren, Virginia
Attn: Naval Ordn. Computation Center

National Bureau of Standards
Washington 25, D. C.
Attn: Dr. S. N. Alexander

Aberdeen Proving Ground, BRL
Aberdeen Proving Ground, Maryland
Attn: Chief, Computation Lab.

Office of Naval Research
Resident Representative
University of Michigan
820 E. Washington Street
Ann Arbor, Michigan

Commanding Officer
ONR, Branch Office
John Crerar Library Bldg.
86 East Randolph Street
Chicago 1, Illinois

Commanding Officer
ONR Branch Office
1030 E. Green Street
Pasadena, California

Commanding Officer
ONR Branch Office
1000 Geary Street
San Francisco 9, California

National Bureau of Standards
Washington 25, D. C.
Attn: Mr. R. D. Elbourn

Naval Ordnance Laboratory
Corona, California
Attn: H. H. Weider

George Washington University
Washington, D. C.
Attn: Prof. N. Grisamore

Dynamic Analysis and Control Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts
Attn: D. W. Baumann

Burroughs Corporation
Research Center
Paoli, Pennsylvania
Attn: A. J. Meyerhoff

Hermes Incorporated
75 Cambridge Parkway
Cambridge 42, Massachusetts
Attn: Mr. Reuben Wasserman

Lockheed Missiles and Space Division
3251 Hanover Street
Palo Alto, California
Attn: D. G. Willis

Univ. of Michigan
Ann Arbor, Michigan
Attn: Dept. of Philosophy,
Prof. A. W. Burks

Census Bureau
Washington 25, D. C.
Attn: Office of Asst. Director for
Statistical Services
Mr. J. L. McPherson

National Science Foundation
Program Director for Documentation
Research
Washington 25, D. C.
Attn: Helen L. Brownson

Univ. of California - LA
Los Angeles 24, California
Attn: Dept. of Engineering,
Prof. Gerald Estrin

Columbia University
New York 27, New York
Attn: Dept. of Physics,
Prof. L. Brillouin

Hebrew University
Jerusalem, Israel
Attn: Prof. Y. Bar-Hillel

Massachusetts Institute of Technology
Cambridge, Massachusetts
Attn: Prof. W. McCulloch

Benson-Lehner Corporation
1860 Franklin Street
Santa Monica, California
Attn: Mr. Bernard Benson

Atomic Energy Commission
Washington 25, D. C.
Attn: Div. of Research

Naval Research Laboratory
Washington 25, D. C.
Attn: Security Systems
Code 5266, Mr. G. Abraham

Cornell University
Department of Mathematics
Ithaca, New York
Attn: Prof. Mark Kac

Dr. A. M. Uttley
National Physical Laboratory
Teddington, Middlesex
England

Diamond Ordnance Fuze Laboratory
Washington 25, D. C.
Attn: Library

U.S. Army Signal Research
and Dev. Lab.
Fort Monmouth, New Jersey
Attn: M. Tenzer

Harvard University
Cambridge, Massachusetts
Attn: School of Applied Science,
Dean Harvey Brook

The University of Chicago
Institute for Computer Research
Chicago 37, Illinois
Attn: Mr. Nicholas C. Metropolis,
Director

Commander
Wright Air Development Division
Wright Patterson Air Force Base,
Ohio
Attn: WCLJR, Maj. L. M. Butsch

Laboratory for Electronics, Inc.
1079 Commonwealth Ave.
Boston 15, Massachusetts
Attn: Dr. H. Fuller

Stanford Research Institute
Computer Laboratory
Menlo Park, California
Attn: H. D. Crane

General Electric Co.
Schnectady 5, New York
Attn: Library, L.M.E. Dept.,
Bldg. 28-501

The Rand Corp.
1700 Main St.
Santa Monica, California
Attn: Numerical Analysis Dept.,
Willis H. Ware

Hunter College
New York 21, New York
Attn: Dean Mina Rees

General Electric Research Laboratory
P. O. Box 1088
Schenectady, New York
Attn: Information Studies Section
R. L. Shuey, Manager

Radio Corporation of America
Moorestown, New Jersey
Attn: Missile and Surface Radar
Division, Sidney Kaplan

University of Pennsylvania
Institute of Co-operative Research
Philadelphia, Pennsylvania
Attn: Dr. John O Conner

Stanford Research Institute
Menlo Park, California
Attn: Dr. Charles Rosen
Applied Physics Group

Northeastern University
360 Huntington Avenue
Boston, Massachusetts
Attn: Prof. L. O. Dolansky

Marquardt Aircraft Company
16555 Saticoy Street
P. O. Box 2013 - South Annex
Van Nuys, California
Attn: Dr. Basun Chang,
Research Scientist

Texas Technological College
Lubbock, Texas
Attn: Paul G. Griffith
Department of Electrical Engineering

IBM Corporation
Military Products Division
Owego, New York
Attn: Dr. S. Winkler

Post Office Department
Office of Research and Engineering
12th and Pennsylvania Avenue
Washington 25, D. C.
Attn: Mr. R. Kopp, Research and
Development Division

Air Force Cambridge Research Center
L. G. Hanscom Field,
Bedford, Massachusetts
Attn: Chief, CRRB

Office of Chief Signal Officer
Department of the Army
Washington, D. C.
Attn: R and D Division SIGRO-6D
Mr. L. H. Geiger

Bell Telephone Laboratories
Murray Hill Laboratory
Murray Hill, New Jersey
Attn: Dr. Edward F. Moore

National Biomedical Research Inst.
9301 19th Avenue
Hyattsville, Maryland
Attn: Dr. R. S. Ledley

National Bureau of Standards
Washington 25, D. C.
Attn: Mrs. Frances Neeland

University of Pennsylvania
Moore School of Electrical Engineering
200 South 33rd Street
Philadelphia 4, Pennsylvania
Attn: Miss Anna Louise Campion

Varo Manufacturing Company
2201 Walnut Street
Garland, Texas
Attn: Fred P. Granger, Jr.

Data Processing Systems Staff
Department of State
Washington 25, D. C.
Attn: F. P. Diblasi

Dr. Saul Gorn, Director
Computer Center
University of Pennsylvania
Philadelphia 4, Pennsylvania

Applied Physics Laboratory
Johns Hopkins University
8621 Georgia Avenue
Silver Spring, Maryland
Attn: Supervisor of Technical Reports

Bureau of Supplies and Accounts, Chief
Navy Department
Washington, D. C.
Attn: Cdr. J. C. Busby, Code W3

Auerbach Electronics Corporation
1634 Arch Street
Philadelphia 3, Pennsylvania

National Aeronautics and Space
Administration
Goddard Space Flight Center
Greenbelt, Maryland
Attn: Chief, Data Systems Division

Federal Aviation Agency
Bureau of Research and Development
Washington 25, D. C.
Attn: RD-375, Mr. Harry Hayman

Mr. Donald F. Wilson
Code 5144
Naval Research Laboratory
Washington 25, D. C.

David Taylor Model Basin
Washington 7, D. C.
Attn: Aerodynamics Laboratory, Code 628
Miss Cravens

Chief, Bureau of Ships
Code 671A2
Washington, D. C.
Attn: Lcdr. E. B. Mahinske, USN

Lincoln Laboratory
Massachusetts Institute of Technology
Lexington 73, Massachusetts
Attn: Library