On Ianov's Program Schemata*

J. D. Rutledge

International Business Machines Corp.,† Yorktown Heights, N. Y.

Abstract. Ianov has defined a formal abstraction of the notion of program which represents the sequential and control properties of a program but suppresses the details of the operations. For these schemata he defines a notion corresponding to computation and defines equivalence of schemata in terms of it. He then gives a decision procedure for equivalence of schemata, and a deductive formalism for generating schemata equivalent to a given one. The present paper is intended, first as an exposition of Ianov's results and simplification of his method, and second to point out certain generalizations and extensions of it. We define a somewhat generalized version of the notion of schema, in a language similar to that used in finite automata theory, and present a simple algorithm for the equivalence problem solved by Ianov. We also point out that the same problem for an extended notion of schema, considered rather briefly by Ianov, is just the equivalence problem for finite automata, which has been solved, although the decision procedure is rather long for practical use. A simple procedure for generating all schemata equivalent to a given schema is also indicated.

There has recently been considerable interest in and some confusion about the work of Iu. I. Ianov, on the theory of programs, as reported in [1, 2, 3, 4]. It is hoped that the following summary and slight strengthening of Ianov's results will be of use to those interested.

Ianov deals with *program schemata*, which are abstract algorithms or programs.

Paper [3] consists of three sections summarized in [1, 2]. The first two sections are discussed in [4, 5]. In these Ianov uses a linear notation for schemata which specializes the notion which will be used here in inessential ways to give a form very much like the program format of a single-address digital computer with a "transfer on zero" instruction as its only transfer. In this notation, each operation is followed by the evaluation of a truth function β_i on a set of predicates P, the next operation or β-evaluation depending on the value of $\beta_i(P)$.

In the third part of paper [3] Ianov presents a formalism much closer to the one used here. A schema is represented by a square matrix M having one row and column for each operation A_i, where M_{ij} is a truth function in the predicates such that A_j follows A_i whenever, following the execution of A_i, M_{ij} is true for the current set of values of the predicates. Ianov shows that every schema expressed in the linear notation has an equivalent expression in the matrix notation and, with the appropriate restriction on the matrix to make the successor function single-valued, any matrix schema has an equivalent linear notation expression. Ianov's main results are that equivalence of two schemata in either notation is decidable, and the presentation of a formal set of transformation rules and equivalences such that any schema may be transformed into any equivalent schema by use of the rules—i.e. he gives a formal deductive system which is

* Received May 1, 1963.

† Watson Research Center.

Journal of the Association for Computing Machinery, Vol. 11, No. 1 (January, 1964), pp. 1–9

complete for equivalence of schemata. It should be noted here that by "equivalence" is meant the very strong equivalence defined below, and not equivalence in the sense of computing the same function under a particular interpretation. Carr [4] gives this deductive system and gives what purports to be an application of it; unfortunately, his nine-step deduction includes four steps which are not allowable in the system, and the two schemata, which he claims are equivalent, are not equivalent in the strong sense, although they do compute the same function when interpreted as intended (but not for other interpretations).

In the following, we shall (1) give a canonical form for equivalence and an effective method for reducing any given schema to canonical form, and (2) sketch an effective method for generating, without repetition, all schemata equivalent to a given schema. These will be done in the notation introduced below and are most natural there, but there are corresponding processes for the linear notation. First, however, we must give an exact definition of the notion of program schema used here, and of the equivalence of two schemata, as well as other associated notions.

A *schema* is a sextuple $\mathfrak{A} = \langle A, P, f, G, A_0, T\rangle$ where

A is a finite set $\{A_i\}$ of *operators*, all of which are considered to be distinct;

P is a finite set $\{p_j\}$ of *predicates*, also distinct;

f: $(A - T) \times 2^\omega \to A$ is the (partial) successor function, interpreted as giving, for some of the A_i and sets of truth values of the p_j, a (unique) A_j which comes next;

G: $A \times 2^\omega \to S(2^\omega)$ is the *shift relation*, specifying, for each operator and (incoming) evaluation, a set of permissible (outgoing) evaluations.[1,2]

While f and G are defined on $A \times 2^\omega$, they are required to depend only on $A \times 2^P$; that is, for all $A_l \in A$ and $\Delta_i, \Delta_k \in 2^\omega$,

$$[\bigwedge_j (p_j \in P \Rightarrow (\Delta_i)_j = (\Delta_k)_j)] \Rightarrow f(A_l, \Delta_i) = f(A_l, \Delta_k), \tag{1a}$$

and similarly for G,

$$\bigwedge_j [p_j \in P \Rightarrow ((\Delta_i)_j = (\Delta_k)_j \text{ and } (\Delta_i')_j = (\Delta_k')_j)] \Rightarrow [\Delta_i' \in G(A_l, \Delta_i) \Leftrightarrow \Delta_k' \in G(A_l, \Delta_k)]. \tag{1b}$$

$A_0 \in A$ is interpreted as *initial operator*.

$T = \{t_1, \cdots, t_n\} \subseteq A$ is a finite set of *terminal operators*.

We will assume that there exists a set $\mathcal{P}$ of all predicates which can appear in schemata, and that this set is indexed by the natural numbers. It is the indexing

[1] This is a generalization of Ianov's "shift distribution." He treats only shift relations of the form $\Delta_k' \in G(A_i, \Delta_k) \Leftrightarrow (\Delta_k')_j = (\Delta)_j$ for all j in a specified subset depending on i, (where $(\Delta)_j = j$th component of Δ). He calls the mapping g: $A \to 2^P$ which defines these subsets the "shift distribution." He notes the desirability of a more general form of shift restriction, but does not elaborate on it. Since the general form used here does not complicate the arguments required for the more restricted form used by Ianov, we adopt it.

[2] We are using 2^ω to denote the set of all infinite strings on $\{0, 1\}$, and $S(2^\omega)$ to denote the set of all subsets of that set. Alternatively, we might describe G as a relation, that is a subset of $(A \times 2^\omega) \times 2^\omega$.

of $\mathcal{P}$ which gives the correspondence between predicates p_j and positions in a string Δ_i. This will be needed in comparing two schemata which have possibly different sets $P \subseteq \mathcal{P}$.

An *evaluation sequence* is an infinite sequence $\Delta = \{\Delta_i\}_{i \subset Nn}$ of infinite strings on $\{0, 1\}$, interpreted as giving the successive truth values of the p_j by $(\Delta_i)_j$ = ith value of p_j. To *apply* $\mathfrak{A}$ *to* Δ, associate with each member Δ_i of the evaluation sequence Δ a member of A by: $A_0 \leftrightarrow \Delta_0$; $A_k \leftrightarrow \Delta_j$ iff $A_{k'} \leftrightarrow \Delta_{j-1}$ and $A_k = f(A_{k'}, \Delta_j)$. Then Δ is an *admissible evaluation sequence for* $\mathfrak{A}$ iff $A_k \leftrightarrow \Delta_i \Rightarrow \Delta_{i+1} \in G(A_k, \Delta_i)$. The sequence of pairs (A^0, Δ_0), (A^1, Δ_1), $\cdots$ so obtained will be called an (*admissible*) *application sequence.*

Now we define the *value* $\mathfrak{A}(\Delta)$ of a schema $\mathfrak{A}$ for an admissible evaluation sequence Δ as the sequence $\{A_i^j\}_{0 \leq j \leq m}$ (if it exists) of elements of A satisfying (i) $A^0 = A_0$, (ii) $A^{k+1} = f(A^k, \Delta_{k+1})$, (iii) $A^m \in T$. That is, the value of a schema applied to an evaluation sequence is the sequence of operators one obtains by running through the schema, beginning at the initial operator and at each step selecting the successor dictated by the evaluation sequence. An admissible evaluation sequence, in turn, is any sequence of sets of truth values for the predicates which can result from the schema under the restriction associated with the shift relation G.

We are now ready to define equivalence for two schemata. $\underline{\mathfrak{A} \equiv \mathfrak{A}'}$ iff for all evaluation sequences Δ, $\mathfrak{A}(\Delta) \simeq \mathfrak{A}'(\Delta)$, where $\simeq$ means, as usual, that if either value exists (i.e. Δ is admissible and the algorithm terminates), the two values are the same.

This is a very strong notion of equivalence; it requires that two equivalent schemata result in exactly the same sequence of operations for all inputs and in all interpretations.

While the reader should at this point have a fairly clear idea of what constitutes an interpretation of a program schema, it will perhaps be useful to give a formal definition and show that the interpretation of "equivalence" as defined above is a natural one. An *interpretation* is a triple $I = \langle D, h, p \rangle$ where

D is the argument domain, which may be any set, finite or infinite;

h: $Nn \rightarrow D^D$ is an assignment of (partial) functions on D to the natural numbers, and thus to the members of any set, in particular A, subscripted by a subset of the natural numbers. $h(A_0)$ = the identity function.

p: $Nn \rightarrow 2^D$ is similarly an assignment of predicates over D to the natural numbers and thus to members of P.

D might be, for example, the set of natural numbers, or the set of finite strings from some alphabet, or the set of real numbers, or the set of states of a digital computer.

An interpretation I is *applicable* to a schema $\mathfrak{A} = \langle A, P, f, G, A_0, T \rangle$ provided that the functions and predicates assigned to the members of A and P are such as to satisfy the shift relation G and that the identity function is assigned to each member of T. Briefly, G specifies the changes in the properties of the argument specified by the predicates which may be made by the function assigned to each operator A_i—for example, in a computer program, an arithmetic order may

change properties of some but not all of the arithmetic registers, but will not change properties of main memory or the tape system; similarly, in Davis's formulation of Turing machines, a quadruple may change the position of the tape or its contents, but not both. Formally, an interpretation I is applicable to $\mathfrak{A}$ iff

$$(d \in D)(A_i \in A): \quad (p(P))(d) = \Delta_j \quad \text{and} \quad (p(P))(h(A_i)(d)) = \Delta_k \\ \Rightarrow \Delta_k \in G(A_i, \Delta_j) : \text{and} : A_i \in T \Rightarrow (h(A_i))(d) = d. \tag{2}$$

If I is applicable to $\mathfrak{A}$, then the result of its application, $I(\mathfrak{A})$, is an interpreted program schema, or *program.*[3]

$I(\mathfrak{A})$ defines a partial function of D to $D \times T$ in the natural way—that is, $(I(\mathfrak{A}))(d)$ is that member of $D \times T$, if any, which is obtained from d by the following recursion:

$$d_0 = d, \quad A^0 = A_0, \quad \Delta_0 = (p(P))(d)$$

$$d_{i+1} = (h(A^i))(d_i) \quad \text{if defined; otherwise,}$$

$$d_{i+1} \text{ and } (I(\mathfrak{A}))(d) \text{ are undefined.} \tag{3}$$

$$\Delta_{i+1} = (p(P))(d_{i+1})$$

$$A^{i+1} = f(A^i, \Delta_{i+1})$$

If $A^i = t_k$ for any i and d_j is defined for all $j \leqq i$, then $(I(\mathfrak{A}))(d)$ is defined and equal to (d_i, t_k); otherwise, it is undefined. (The reader will note the very close correspondence of this recursion to the basic instruction execution cycle of a digital computer.) We can now make the following observation.

THEOREM. *For any two program schemata* $\mathfrak{A}$ *and* $\mathfrak{A}'$, $\mathfrak{A} \equiv \mathfrak{A}'$ *iff for all interpretations* I, *applicable to both,* $I(\mathfrak{A}) \simeq I(\mathfrak{A}')$ *as functions.*

PROOF. Suppose $\mathfrak{A} \equiv \mathfrak{A}'$. This means that for all evaluation sequences which produce values, the values of the two schemata are the same. Now I assigns the same function over D to similarly named operators in the two schemata; that is, $h(A_i) = h(A_i')$, and by the applicability of I, the sequence of evaluations obtained by applying $I(\mathfrak{A})$ or $I(\mathfrak{A}')$ to any member of D must be admissible as an evaluation sequence for $\mathfrak{A}$ or $\mathfrak{A}'$ respectively. Thus the equality of the values for admissible evaluation sequences implies that the sequence of functions applied to an argument $d \in D$ is the same for $I(\mathfrak{A})$ and $I(\mathfrak{A}')$ provided the sequence is finite. Consequently, the two are equal as partial functions on D

[3] This formalization is a bit more restrictive than is really wanted, since it rejects as inapplicable an interpretation in which an operation violates its shift relation for input evaluations with which it can never be associated in admissible evaluation sequences. An example would be an interpretation involving a square root operator following a branch which prevents negative arguments from reaching the operator, a shift relation for that operator which requires that all results be positive, while the operator produces negative result for negative input. However, this difficulty is inessential in that for any such interpretation I, there is always a second interpretation I', applicable by this definition, such that $(d \in D).(I(\mathfrak{A}))(d) \simeq (I'(\mathfrak{A}))(d)$. The definition of applicable interpretation has been chosen as having a strictly local character, in this sense being simpler.

into $D \times T$. Conversely, if $I(\mathfrak{A}) \simeq I(\mathfrak{A}')$ for all applicable interpretations I, they are equal in particular for the interpretation I_v defined by:

D_v is the set of infinite strings of form $u\Delta$, where u is a (possibly empty) finite string from the alphabet $A \cup A'$, and Δ is, as usual, an infinite string of infinite strings Δ_i on $\{0, 1\}$.

$$\begin{aligned}(h_v(n))(u\Delta_k\Delta_{k+1}\cdots) &= (uA_n\Delta_{k+1}\cdots) \text{ for } A_n \notin T \cup T' \text{ if } \Delta_{k+1} \in G(A_n, \Delta_k);\\ &\quad \text{undefined otherwise.}\\ h_v(n) &= \text{identity function for } A_n \in T \cup T'\\ p_v(n)(u\Delta_k\cdots) &= (\Delta_k)_n\end{aligned} \tag{4}$$

If $d \in D_v$ has empty initial segment, it may be interpreted as an evaluation sequence; if this sequence is admissible and results in a value for schema $\mathfrak{A}$, then the first component of $(I_v(\mathfrak{A}))(d)$ has an initial segment which is just the value of $\mathfrak{A}$ for this sequence. Clearly, if $I_v(\mathfrak{A}) \simeq I_v(\mathfrak{A}')$, then the values of the two schemata are the same for all admissible evaluation sequences, and $\mathfrak{A} \equiv \mathfrak{A}'$.

This example suggests a very close connection between program schemata and finite automata which we will develop more fully below, where we will show that an extended form of the equivalence problem for schemata can be reduced to the equivalence problem for finite automata. However, the resulting algorithms for testing equivalence and obtaining canonical forms are much longer than that which follows here.

We now give an algorithm which produces, for any given schema $\mathfrak{A}$, a unique canonical schema $\mathfrak{A}_c$, such that $\mathfrak{A} \equiv \mathfrak{A}' \Leftrightarrow \mathfrak{A}_c$ is identical to $\mathfrak{A}_c'$. Represent f by a matrix M in which the entry M_{ij} is the set of terms of the form $\wedge_{p_i \in r} p_i^*$, where $p_i^* = p_i$ or $\bar{p}_i$, such that if one member of the set is satisfied at the completion of operator i, operator j follows operator i. We can change M_{ij}, obtaining a schema equivalent to the original one, only by adding or deleting a term which could not in fact occur at that point in any admissible evaluation sequence for the schema which leads to a value, or by adding or deleting rows and columns representing inaccessible operators.

To identify these combinations we construct a sequence of matrices M^k, $k = 0, 1, \cdots \bar{k}$ according to the recursion (5). This construction traces all the paths (i.e. sequences of pairs A^i, Δ_i) starting at A_0 which are permissible under G, and which contains each pair (A^i, Δ_i) at most once.

$$\begin{aligned}M_{ij}^0 &= \emptyset \quad \text{for} \quad i \neq 0, \qquad M_{0j}^k = M_{0j} \quad \text{for all } k,\\ M_{ij}^{k+1} &= M_{ij} \cap \bigcup_l G(A_l, M_{li}^k) \quad \text{for } i \neq 0,\end{aligned} \tag{5}$$

where the definition of G is extended to set arguments in the usual manner. We see easily that for some $\bar{k}$ and all n, $M^{\bar{k}} = M^{\bar{k}+n}$, since for all ij, $M_{ij}^k \subseteq M_{ij}^{k+1} \subseteq M_{ij}$, and $M^{k+1} = M^k \Rightarrow M^{k+n} = M^k$ for all n.

As will be shown below this process has given us all operation-evaluation pairs which can occur in admissible application sequences. However, it may be the

case that some of these pairs occur only in sequences which do not produce values; that is, there may be pairs such that if an evaluation sequence results in that pair, then no continuation of the sequence can result in a pair of form (t_i, Δ_j). Equivalent schemata can differ on such sequences. To remove these pairs, we must start at T and back-track to determine which pairs have continuations leading to members of T, in much the same manner in which we have found the pairs accessible from A_0. The sequence of matrices this time is defined by:

$$\begin{aligned} &(M')^0_{ij} = \emptyset \quad \text{for} \quad A_j \notin T, \qquad (M')^k_{ij} = M^k_{ij} \quad \text{for } A_j \in T \text{ and all } k, \\ &(M')^{k+1}_{ij} = M^k_{ij} \frown G^{-1}(A_j, \bigcup_l (M')^k_{jl}) \quad \text{for } A_j \notin T. \end{aligned} \tag{6}$$

Again, the process must terminate with some $M'^{k'}$. Now delete from $(M')^{k'}$ all rows and columns i (other than row 0) such that $\bigcup_j (M')^{k'}_{ji} = \emptyset$, repeating this operation until no further change results to obtain $M_c{}^*$. $M_c{}^*$ is the matrix of the canonical schema except that it may include reference to predicates which are actually superfluous. The operators corresponding to the deleted rows are deleted from the set A to give the operator set A_c for the canonical schema. The deleted operators are just those which can never occur in values of $\mathfrak{A}$. We must now restrict G to G_c so that the admissible evaluation sequences for $\mathfrak{A}_c$ will be just those Δ satisfying:

(i) $(\exists \Delta'). \Delta'' = (\Delta_0, \Delta_1, \cdots, \Delta_i, \Delta'_{i+1}, \Delta'_{i+2}, \cdots) \rightarrow \mathfrak{A}_c(\Delta'')$ is defined.

This is obtained as

$$G_c = G \cap \{((A_{c_i}, \Delta_l), \Delta_k) \mid (\Delta_l \in \bigcup_j M^*_{c_{ji}} \quad \text{and} \quad \Delta_k \in \bigcup_j M^*_{c_{ij}}) \quad \text{or} \quad (i = 0 \quad \text{and} \quad \Delta_k \in \bigcup_j M^*_{c_{0j}})\},$$

where, in accordance with (1a), we write $\Delta_l \in M^*_{ij}$ iff the projection of Δ_l on P is in M^*_{ij}. Now let P_c be the smallest subset of P for which (1a) and (1b) hold with A_c in place of A, G_c in place of G, and f_c as defined by $M_c{}^*$ in place of f. Finally, $T_c = T \cap A_c$, and $\mathfrak{A}_c = \langle A_c, P_c, f_c, G_c, A_0, T_c \rangle$.

Definition. A schema $\mathfrak{A}'$ is a *restriction* of $\mathfrak{A}$ iff each element of $\mathfrak{A}'$ is a subset of the corresponding element of $\mathfrak{A}$.

LEMMA. *If $\mathfrak{A}'$ is a restriction of $\mathfrak{A}$ and $\mathfrak{A}'(\Delta)$ is defined, then $\mathfrak{A}'(\Delta) = \mathfrak{A}(\Delta)$.*

PROOF. Immediate from the definition of value.

From the lemma it follows that $\mathfrak{A} \equiv \mathfrak{A}_c$ can fail to be true only in case, for some Δ, $\mathfrak{A}(\Delta)$ is defined and $\mathfrak{A}_c(\Delta)$ is not. Furthermore, if $\mathfrak{A}_c$ applied to Δ produces an infinite sequence of pairs $\Lambda^i \leftrightarrow \Delta_i$, this sequence must be the same for $\mathfrak{A}$ and $\mathfrak{A}_c$ and hence, since $t_j \leftrightarrow \Delta_i$ is a pair for $\mathfrak{A}$ applied to Δ, $\mathfrak{A}_c(\Delta)$ must be defined. The only remaining possibility for $\mathfrak{A} \not\equiv \mathfrak{A}_c$ is that for some Δ $\mathfrak{A}(\Delta)$ is defined, but in applying $\mathfrak{A}_c$ to Δ a pair $A_m{}^i \leftrightarrow \Delta_i$ is reached such that either $\Delta_{i+1} \notin G_c(A_m{}^i, \Delta_i)$ or $f_c(A_m{}^i, \Delta_{i+1})$ is not defined. Let Δ_i be the first such pair. The proof that $f_c(A_m{}^i, \Delta_i)$ is defined is by induction, following the construction

of f_c by (5) and (6). First we show that f^k, defined by M^k, is defined on $(A_m{}^i, \Delta_{i+1})$. By the first clause of (5), defining $M^k_{0j} = M_{0j}$, i cannot be zero. Since Δ_i is the first term of Δ for which f_c is undefined, we have $f^k(A_l^{i-1}, \Delta_i) = A^i$, so $\Delta_i \in M^k_{li}$ for some l. Further, $\Delta_{i+1} \in G(A_m{}^i, \Delta_i)$, so $\Delta_{i+1} \in M^k_{mj}$ for some j, and $f^k(A_m{}^i, \Delta_{i+1}) = f(A_m{}^i, \Delta_{i+1})$, contradicting the assumption that $f^k(A_m{}^i, \Delta_i)$ is undefined. A similar argument, using (6), shows that $f_c(A_m{}^i, \Delta_i)$ is defined. Now observe, from the definition of G_c, that $[\Delta_{i+1} \in G(A^i, \Delta_i)$ and $(\exists l)(A_l \in A_c$ and $A^i = f_c(A_l, \Delta_i))$ and $f_c(A^i, \Delta_{i+1})$ defined$] \rightarrow \Delta_{i+1} \in G_c(A^i, \Delta_i)$, and we have $\mathfrak{A}(\Delta) = \mathfrak{A}_c(\Delta)$ and $\mathfrak{A} \equiv \mathfrak{A}_c$.

We must now show that $\mathfrak{A}_c$ has been adequately pruned; that is, that any alteration of $\mathfrak{A}_c$ will necessarily affect the value of $\mathfrak{A}_c(\Delta)$ for some Δ, or else render it noncanonical, and thus that for two canonical schemata $\mathfrak{A}$ and $\mathfrak{A}'$, $\mathfrak{A} \equiv \mathfrak{A}'$ iff $\mathfrak{A} = \mathfrak{A}'$. Assume $\mathfrak{A} \equiv \mathfrak{A}'$, where $\mathfrak{A}$ and $\mathfrak{A}'$ are both canonical schemata. Since every element of A appears in some value of $\mathfrak{A}$, and similarly for A' and $\mathfrak{A}'$, we must have $A = A'$. Since only members of $T(T')$ are terminal elements of values, and are always terminal when they occur, we have also $T = T'$, and, of course, $A_0 = A_0{}'$. If $f \neq f'$, then for some Δ_i, A_l, $f(A_l, \Delta_i)$ is defined, while $f'(A_l, \Delta_i)$ is undefined or different (or vice versa). Since $\mathfrak{A} \equiv \mathfrak{A}'$, it follows that no pair $f(A_l, \Delta_i) \leftrightarrow \Delta_i$ can ever occur in an application of $\mathfrak{A}$ which leads to a value. But since $\mathfrak{A}$ is canonical and $f(a_l, \Delta_i)$ is defined, the construction according to (5) and (6) forbids this. Also, it cannot be the case that f, respectively G, depends on any predicate on which f', respectively G', does not depend, since then a change in the corresponding position of some term of some Δ to give Δ' would make $\mathfrak{A}'(\Delta') = \mathfrak{A}(\Delta) \neq \mathfrak{A}(\Delta')$, so we have $P = P'$ to complete the proof. Thus we have

THEOREM. *For all schemata* $\mathfrak{A}$, $\mathfrak{A}'$, $\mathfrak{A} \equiv \mathfrak{A}_c$, $\mathfrak{A}' \equiv \mathfrak{A}_c{}'$, *and* $\mathfrak{A} \equiv \mathfrak{A}'$ *iff* $\mathfrak{A}_c = \mathfrak{A}_c{}'$.

This theorem clearly settles the decision problem. We may also note that since any schema is an extension (inverse restriction) of its canonical form, we may obtain any schema equivalent to a given one by first finding the canonical form and then performing some extension of that canonical schema which does not in fact extend the function from evaluation sequences to values which it represents. The problem of enumerating all schemata equivalent to a given one, then, is just the problem of enumerating all such extensions. Any equivalent extension of a canonical schema $\mathfrak{A}$ may be obtained by the following operations:

(1) arbitrarily augment the set P.
(2) augment the set A with additional operators in one or both of two classes; (a) operators which are to be inaccessible from A_0; that is, they can never appear in application sequences, and (b) operators from which no member of T can be reached.
(3) augment the functions f and G in any manner consistent with the restriction imposed in (2) subject to the condition that no new successor relations for admissible sequences may be added between members of the original operator set. That is, a member of the inaccessible set may be a successor of an operation not in that set only for an evaluation which is not allowed by the augmented relation G, and an operation from which T can be reached can be a successor of a member of the second set only for a similarly forbidden evaluation.
(4) augment T by any subset of the set of inaccessible operators.

Up to this point we have followed the general point of view adopted by Ianov. However, the reader who is familiar with the theory of finite automata will have recognized that there is a close resemblance between finite automata and program schemata. We will now indicate how this resemblance can be made explicit and how an extended version of the foregoing results can be obtained directly from automata theory. We wish to associate with each program schema a finite automaton in such a way that equivalence of schemata corresponds to (behavioral) equivalence of automata. Something must clearly be done about the non-finite nature of the evaluation sequences, which must correspond to strings on a finite alphabet. For any given schema (or finite set of schemata) the set of predicates P (union of sets P_i) is finite, and evaluations which agree on this set are equivalent for the given schema (schemata). Thus we may consider a set of 2^P evaluations. Now associate with the schema $\mathfrak{A}$ the finite automaton $B = \langle A, 2^P, N, f, h, A_0\rangle$, where:

A, P, f, A_0 are as in $\mathfrak{A}$, except that f is considered to be defined on 2^P as a set of equivalence classes on 2^ω.

N, the output alphabet, is a subset of A (so far, the improper subset).

h, the output function, maps $A \to N$ (so far, the identity function).

This correspondence takes no account of the shift relation G of the schema, or of the set of terminal operators. However, it is clear that every admissible evaluation sequence for $\mathfrak{A}$ which produces a value $v \in A^*$ has a unique initial segment which corresponds to an input string $\Delta \in (2^P)^*$ such that $B(\Delta) = v$. We must define an automaton B_1, similar to B, which will not only have this property, but will also give output only for input strings corresponding to initial segments of admissible evaluation sequences for which $\mathfrak{A}$ yields values. This is

$$B_1 = \langle (A \times 2^P) \cup \{\alpha, \rho\}, 2^P, N, f_1, h_1, \alpha\rangle,$$

where:

$$f_1'((A_i, \Delta_k), \Delta_j) = \begin{cases} (f(A_i, \Delta_j, \Delta_j) & \text{if } \Delta_j \in G(A_i, \Delta_k) \text{ and } f(A_i, \Delta_j) \text{ defined} \\ \rho & \text{otherwise} \end{cases}$$

$$f_1((A_i, \Delta_k), \Delta_j) = \begin{cases} f_1'((A_i, \Delta_k), \Delta_j) \text{ if there exists a path (in } f_1') \text{ from that} \\ \quad \text{state to some state in } T \times 2^P) \\ \rho \quad \text{otherwise} \end{cases}$$

$$f_1(\alpha, \Delta_i) = (A_0, \Delta_i)$$

$$f_1(\rho, \Delta_i) = \rho$$

$$h_1(A_i, \Delta_j) = h(A_i)$$

$$h_1(\rho) = h_1(\alpha) = \text{(blank)}$$

Now we have the

THEOREM. *For any two schemata* $\mathfrak{A}$, $\mathfrak{A}'$, $\mathfrak{A} \equiv \mathfrak{A}'$ *iff* B_1 *is (finite automation) equivalent to* B_1'.

PROOF. It is clear that if $\mathfrak{A}(\Delta) = \mathfrak{A}'(\Delta)$, where both are defined, then for any initial segment u of the string on 2^P corresponding to Δ, $B_1(u) = B_1'(u)$,

and that if B_1 is equivalent to B_1' then $\mathfrak{A} \equiv \mathfrak{A}'$. It remains to show that if $\mathfrak{A}(\Delta)$ is undefined and $\mathfrak{A} \equiv \mathfrak{A}'$, then $B_1(u) = B_1'(u)$ for all strings u corresponding to initial segments of Δ. Let v be the longest initial segment of Δ which is also an initial segment of an evaluation sequence on which $\mathfrak{A}$ is defined, and let v' be the corresponding string on 2^P. Clearly $B_1(v') = B_1'(v')$. If $v\Delta_k$ is the initial segment of Δ of length one greater than that of v, it must be the case that either Δ_k is the first evaluation which renders Δ inadmissible, that is $\Delta_k \notin G(A^{k-1}, \Delta_{k-i})$, or else Δ_k produces a transition to an operator from which no member of T is accessible. In either case, the string corresponding to $v\Delta_k$ takes B into state ρ. It is also clear that v must also be the longest initial segment of Δ which is an initial segment of a sequence on which $\mathfrak{A}'$ is defined, else some extension of $v\Delta_k$ would be an evaluation sequence for which $\mathfrak{A}'$ is defined but $\mathfrak{A}$ is not, contrary to the hypothesis of their equivalence. Thus for any initial segment w of Δ, $B_1(w) = B_1'(w)$, and B_1 is equivalent to B_1'.

Since methods for deciding equivalence and finding canonical (minimal) forms for finite automata are well known [6], the above Theorem gives a solution of these problems for schemata. In fact, it goes somewhat farther, since the restriction on the output function of B and B_1 used above is not necessary. If we let N be a proper subset of A, and let h be an arbitrary function from A into N, B and the correspondingly altered B_1 are still respectable finite automata, and the same machinery is available for manipulating them. The corresponding operations on the schema give what Ianov calls a "schema with identification of operators." Values are defined as usual for such schemata, but for the definition of equivalence, two values are considered equal if one can be obtained from the other by substitution of operators for others with which they are identified. This sort of equivalence is a special case of equivalence of strings under substitution of substrings according to a finite set of equivalence relations between finite strings, which is, of course, the word problem for semigroups. The full problem is unsolvable, but the special case of interest here, in which the defining equivalence relations are on single letters only, is solvable, and the equivalence problem for schemata under equivalences of this type has been solved by the connection to finite automata just demonstrated. Schemata of this type are of interest as being closer to actual programs, since they represent programs in which the same operation may occur in several places.

REFERENCES

1. Ianov, Iu. I. On the equivalence and transformation of program schemes. *Doklady, A. N. USSR 113* (1957), 39–42; Trans., *Comm. ACM 1*, 10 (Oct. 1958), 8–12.
2. ——. On matrix program schemes. *Doklady A.N. USSR 113* (1957), 283–286; Trans., *Comm. ACM 1*, 12 (Dec. 1958), 3–6.
3. ——. On the logical schemata of algorithms. *Problems of Cybernetics 1* (1958), 75–127.
4. Carr J. W., III. In *Handbook of Automation, Computation and Control, Vol. II*, Ch. 2, 2–228. Wiley, New York, 1959.
5. Fells, E. M. Kaluzhnin graphs and Yanov writs. In *Logik und Logikkalkül*, Verlag Karl Alber, Munich, 1962.