JOURNAL OF COMPUTER AND SYSTEM SCIENCES: 4, 220–249 (1970)

On Formalised Computer Programs

D. C. LUCKHAM

Stanford University (Artificial Intelligence project), Stanford, California 94305

D. M. R. PARK

University of Warwick, Coventry, England

AND

M. S. PATERSON

Massachusetts Institute of Technology, Cambridge, Massachusetts, 02139

Received August 29, 1969

1. INTRODUCTION

In [2] Ianov introduced a simple abstract model of a computer program (called a program schema) based on the notion of a program as a finite linear sequence of instructions of two types: computational instructions and conditional binary transfer instructions. The transfers, controlling the order in which instructions are executed, depend on the value of propositional truth functions of a finite number of variables, the values (true or false) of which may be changed by the execution of any computation. Each computational instruction and Boolean function is uninterpreted (i.e., its meaning or interpretation is left open) so that a program schema may be thought of as representing a family of computer programs, each member being a possible interpretation of the schema. Properties of abstract schemas of this type are independent of any particular computer or programming language, and will be true of programs written in any language which possesses the basic sequential and control characteristics assumed in the model. Ianov's principal results are that there exist algorithms for deciding whether or not, under all interpretations, a given pair of schemas represent the same programs (i.e., are equivalent), and for reducing a schema to an equivalent simple canonical form. These results are concisely presented by

Rutledge [8] who also mentions a direct relation between Ianov schemas and finite automata (so that results from other areas of the theory of computation may be interpreted in a "program-oriented" way).

One of the principal aims of investigating the properties of these abstract schemas is to approach the problem of optimizing programs on a general basis. Algorithms for eliminating instructions from schemas or for merging and shortening loops while preserving equivalence should provide a basis for practical simplification. However, Ianov's model is too elementary, and further "general" properties of computer programs must be incorporated—for example, the dependence or independence of computational instructions within the program.

In Section 2 we present a natural extension of Ianov schemas by introducing free variables. Computational and transfer instructions are expressed as functions of variables, and the value of each computation is assigned to a variable. Intuitively, the abstract schema now represents also the "flow of information" in the program.

Although our formalism possesses certain Algol-like features, these are easily shown to be inessential; program schemas can be represented, for example, by sets of recursion equations, so that the results of the following sections hold true of programs written in this formalism too.

The question of the solvability of the equivalence of program schemas is closely connected with the existence of algorithms for simplification. If the equivalence problem was solvable, then an algorithm for reducing a schema to simplest (in some sense) possible form would exist in principle. In Sections 4 and 5 we shall show that for almost any reasonable notion of equivalence between computer programs, the two questions of equivalence and nonequivalence of pairs of schemas are *not* partially decidable. (Here, "partially decidable" means "recursively enumerable": a relation $r(a, b)$ *is* partially decidable but *not* decidable if there is an algorithm for generating the list of all pairs $\langle a, b \rangle$ such that $r(a, b)$ holds, but no algorithm exists for listing those pairs such that $r(a, b)$ does not hold.) It follows from this that optimizing algorithms for completely reducing a schema to, e.g., shortest possible form do not exist. In fact such algorithms will be shown not to exist even in the case of highly restricted classes of schemas. Section 4 establishes the basic nonpartial-decidability result for strong equivalence, ([3] contains an earlier proof), and Section 5 extends the result to weaker notions of equivalence. The proofs depend on some properties of multihead automata which are presented in Section 3.

In Section 6 we show the equivalence between a particularly simple class of schemas and multitape automata. It follows that Ianov's schemas are equivalent to program schemas containing *one* free variable. Finally, to counteract the rather negative results of Sections 4 and 5, we give a brief survey of classes of schemas for which the equivalence problem is decidable and optimizing algorithms do exist, and we list some of the open questions.

2. Program Schemas

We adopt a formal language containing the following symbols:

(i) numerals,
(ii) $F_1^1, F_2^1, F_3^1, \ldots, F_1^2, F_2^2, F_3^2, \ldots$ (operator symbols),
(iii) $L_1, L_2, L_3, \ldots$ (location symbols),
(iv) $T_1, T_2, T_3, \ldots$ (transfer symbols),
(v) $:=$ (assignment symbol),
(vi) STOP,
(vii) (,), and comma (auxiliary symbols).

The permissible statements (or instructions) of the language are of three types (symbols other than L, F, STOP, denote numerals; in each case k is called the prefix or address of the instruction):

(i) *Assignment instructions*[1]

$$k.\ L_i := F_j^n(L_{k_1}, L_{k_2}, \ldots, L_{k_n}).$$

(ii) *Transfer instructions*

$$k.\ T_i(L_j)\ m,\ n,$$

where m, n are numerals (transfer addresses).

(iii) *Stop instruction*

$$k.\ \text{STOP}.$$

A Program Schema, P, is a finite sequence of permissible instructions such that (i) the prefix of each instruction is its position in the sequence, (ii) all transfer addresses are prefixes of P, and (iii) the last instruction is either a transfer or STOP.

It will often be convenient to represent schemas by flow diagrams; Diagram 1 is a representation of the schema,

1. $L_2 := F_1(L_2)$,
2. $T_1(L_1)\ 5,5$,
3. $L_2 := F_2(L_1, L_2)$,
4. $L_1 := F_3(L_1)$,
5. $T_1(L_1)\ 6,3$,
6. STOP.

Note that instruction 2 is an "unconditional" transfer and is not explicitly represented in the flow diagram.

[1] Also called computational or operator instructions.

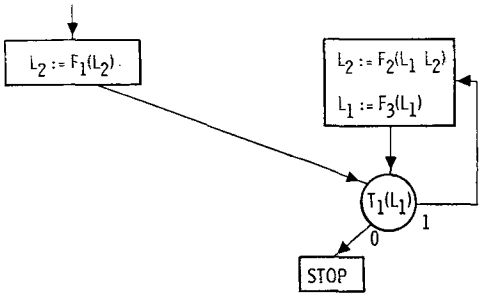


DIAGRAM 1

Intuitively, a program schema is intended to represent a family of possible computer programs in the following sense. If the operators and transfers of a schema P are given a particular interpretation (as, respectively, functions and characteristic functions, not necessarily constructive) the schema becomes a program which can be executed by an idealized computer. The computation proceeds sequentially by executing the instructions in the order in which they occur in the schema (except when transfer instructions are encountered) starting with the first instruction and an initial set of data assigned to the locations of P, say $L_1, L_2, \ldots, L_n$. At each step one of two types of instruction is executed: if $L_w := F_j(L_{u_1}, \ldots, L_{u_v})$ is executed then the result of a certain computation involving the contents of locations $L_{u_1}, \ldots, L_{u_v}$ (which remain unchanged), is assigned to L_w; if $T_j(L_w)\, u, v$ is executed then a decision is made to execute either the u-th or the v-th instruction next, depending on the contents of L_w. If the computer executes $T_j(L_w)\, u, v$, and $T_j(L_{w'})\, u', v'$ at different steps with the same contents of L_w and $L_{w'}$, then it must decide to execute S_u (the u-th instruction) at one step if and only if it decides to execute $S_{u'}$ at the other step; in other words the decisions must be made consistently. If the computation stops (by executing a STOP instruction) its value is the final vector of values assigned to the locations $L_1, \ldots, L_n$. The computation sequence chosen will in general depend on the interpretation as will the nature of the individual steps. We shall say that program schemas are intended to represent uninterpreted computer programs, in the sense that we are interested in their behaviour over a very wide class of interpretations.

The schema of diagram 1 computes the factorial function under some interpretations over the integers; for example, $I(F_1)(x) = 1$, $I(F_2)(x, y) = x \times y$, $I(F_3)(x) = x - 1$, and $I(T_1)(x) = 0$ if $x = 0$ and 1 if $x > 0$. Under other interpretations, over list structures, it computes the function reverse(x) McCarthy [4]; for example, $I(F_1)(x) = \mathrm{NIL}$, $I(F_2)(x, y) = \mathrm{Cons}(\mathrm{Car}(x), y)$, $I(F_3)(x) = \mathrm{Cdr}(x)$ and $I(T_1)(x) = 0$ if $\mathrm{Null}(x)$ and 1 otherwise.

Formally, an *interpretation* I of P is a mapping from the location, operator, and

transfer symbols of P into a set D and the set of functions and characteristic functions such that

(i) Each location symbol L_i is assigned an element $I(L_i) \in D$.

(ii) Each operator symbol F_i^n is assigned an n-ary function, $I(F_i^n) : D^n \to D$.

(iii) Each transfer symbol T_i is assigned a characteristic function on D, $I(T_i) : D \to \{0, 1\}$.

Notation

σ and $P(\sigma)$ denote, respectively, a possible sequence of instructions of the schema P (i.e., a path through the flow diagram of P), and the sequence of vectors of values assigned to $\langle L_1, \ldots, L_n \rangle$ when σ is executed. $\sigma(i)$ is the i-th member of σ, $P(\sigma)(i)$ is the vector after $\sigma(i)$ is executed and $P(\sigma)(i, j)$ is its j-th component. S_u denotes the u-th statement of P. The execution and computation sequences, σ_I and $P_I(\sigma_I)$, corresponding to a given interpretation I are defined inductively as follows:

(i) $\sigma_I(1) = S_1$.

(ii) $P_I(\sigma_I)(0) = \langle I(L_1), \ldots, I(L_n) \rangle$.

(iii) If $\sigma_I(i + 1)$ is the assignment

$$k.\ L_w := F_t^v(L_{u_1}, \ldots, L_{u_v}),$$

then

$$\sigma_I(i + 2) = S_{k+1},$$

$$P_I(\sigma_I)(i + 1, j) = P_I(\sigma_I)(i, j) \quad \text{if} \quad j \neq w,$$

and

$$P_I(\sigma_I)(i + 1, w) = I(F_t^v)(P_I(\sigma_I)(i, u_1), \ldots, P_I(\sigma_I)(i, u_v)).$$

(iv) If $\sigma_I(i + 1)$ is the transfer

$$T_j(L_w)\ u, v,$$

then

$$\sigma_I(i + 2) = S_u \quad \text{if} \quad I(T_j)(P_I(\sigma_I)(i, w)) = 0,$$

$$\sigma_I(i + 2) = S_v \quad \text{if} \quad I(T_j)(P_I(\sigma_I)(i, w)) = 1,$$

and

$$P_I(\sigma_I)(i + 1) = P_I(\sigma_I)(i).$$

If σ_I is finite (of length m say) then the *value* of the computation of P under I [notation: Val(P_I)] is defined to be the final vector of $P_I(\sigma_I)$, i.e., Val$(P_I) = P_I(\sigma_I)(m)$.

If a sequence σ is executed under some interpretation, we shall call it an *execution sequence.*

Consider an interpretation I, over the class of strings of operator and locations symbols of P, such that $I(L_i) = L_i$, and, if $\alpha_1, ..., \alpha_v$ are any strings, then $I(F_t^v)(\alpha_1, ..., \alpha_v) = F_t^v\alpha_1 \cdots \alpha_v$, the result of concatenating $F_t^v, \alpha_1, ..., \alpha_v$. Such an interpretation will be called a *free interpretation.*

It is easily seen that any execution sequence σ is executed under some free interpretation; for consider the associated free computation sequence $P(\sigma)$:

(i) $P(\sigma)(0) = \langle L_1, ..., L_n \rangle$.

(ii) If $\sigma(i + 1)$ is the assignment,

$$L_w := F_t^v(L_{u_1}, ..., L_{u_v}),$$

then $P(\sigma)(i + 1, j) = P(\sigma)(i, j)$ if $j \neq w$, and $P(\sigma)(i + 1, w)$ is the string obtained by concatenating F_t^v, $P(\sigma)(i, u_1), ..., P(\sigma)(i, u_v)$ in that order.

With each transfer T_j in P we associate two sets, $\mathscr{L}_j(\sigma)$ and $\mathscr{R}_j(\sigma)$: for each i, if $\sigma(i)$ is a transfer instruction of the form $T_j(L_w)\, u, v$ and $u \neq v$, then $P(\sigma)(i, w) \in \mathscr{L}_j(\sigma)$ if and only if $\sigma(i + 1)$ is S_u, and $P(\sigma)(i, w) \in \mathscr{R}_j(\sigma)$ if and only if $\sigma(i + 1)$ is S_v. Since σ is an execution sequence, $\mathscr{L}_j(\sigma) \cap \mathscr{R}_j(\sigma) = \varnothing$. Therefore there is a free interpretation I such that if $\alpha \in \mathscr{L}_j(\sigma)$, then $I(T_j)(\alpha) = 0$ and if $\alpha \in \mathscr{R}_j(\sigma)$, then $I(T_j)(\alpha) = 1$. Clearly $\sigma_I = \sigma$. We shall denote the free interpretation value of σ by $\mathrm{Val}(P(\sigma))$.

DEFINITION 1. Program schemas P and P' are *strongly equivalent* ($P \equiv P'$) if and only if for all interpretations I, $\mathrm{Val}(P_I) = \mathrm{Val}(P_I')$ whenever either value is defined.

This appears to be the strongest possible notion of equivalence and may be thought of intuitively as demanding that equivalent program schemas behave in the same way on all idealized computers. It might be argued that this is too restrictive a notion although it is easy to see that it coincides with the notion obtained by considering only free interpretations or (by Godel-numbering the formal strings) interpretations over the domain of natural numbers. We may obtain more "constructively acceptable" notions of equivalence by restricting the interpretations in Definition 1. Among these possibilities we mention *finite equivalence* ($P \equiv_F P'$ if $P \equiv P'$ for all interpretations on finite domains[2]) and *recursive equivalence* ($P \equiv_R P'$ if $P \equiv P'$ for all interpretations in which the domain and functions are general recursive). Finally, it may be of interest in discussing the problem of simplification to consider a notion of *weak equivalence*: P and P' are weakly equivalent ($P \simeq P'$) if and only if for all interpretations I, $\mathrm{Val}(P_I) = \mathrm{Val}(P_I')$ whenever *both* values are defined. (Note that weak equivalence is *not* an equivalence relation.) For the moment we restrict our attention to strong equivalence.

[2] Henceforth called *finite interpretations.*

There is a simple syntactic characterisation of strong equivalence in terms of the execution sequences through P and P'. A set of sequences $\{\sigma_i\}$ over some schema or schemas is said to be *consistent* if, for each T_j occurring in the schema(s)

$$\bigcup_i \mathscr{L}_j(\sigma_i) \cap \bigcup_i \mathscr{R}_j(\sigma_i) = \varnothing$$

Thus σ is an execution sequence if and only if $\{\sigma\}$ is consistent.

THEOREM 2.1. *$P \equiv P'$ if and only if for all consistent pairs of sequences σ through P and σ' through P', $\mathrm{Val}(P(\sigma)) = \mathrm{Val}(P'(\sigma'))$ whenever either value is defined.*

Proof. ($\Rightarrow$) Suppose that $P \equiv P'$ and that σ and σ' are consistent.

The consistency of σ and σ' implies that there is a single free interpretation, I, over the domain D of components of the vectors of $P(\sigma)$ and $P'(\sigma')$ such that $\sigma_I = \sigma$ and $\sigma_I' = \sigma'$. (Simply define $I(T_j)$ for any $\alpha \in D$ by: $I(T_j)(\alpha) = 0$ if and only if $\alpha \in \mathscr{L}_j(\sigma) \cup \mathscr{L}_j(\sigma')$, $I(T_j)(\alpha) = 1$ otherwise.) Since $\mathrm{Val}(P_I) = \mathrm{Val}(P_I')$ if either value is defined, it follows that either $\mathrm{Val}(P(\sigma)) = \mathrm{Val}(P'(\sigma'))$ or both values are undefined.

($\Leftarrow$) If I is any interpretation of P and P' then the pair $\{\sigma_I, \sigma_I'\}$ is consistent; hence $\mathrm{Val}(P(\sigma_I)) = \mathrm{Val}(P'(\sigma_I'))$ if either value is defined. But $\mathrm{Val}(P_I)$ is the interpretation of $\mathrm{Val}(P(\sigma_I))$ (i.e., if α is a component of a vector of $P(\sigma_I)$, say $F_i^m F_j^n \cdots L_k \cdots$, then its interpretation is $I(\alpha) = I(F_i^m)(I(F_j^n)(\cdots I(L_k) \cdots))$, and if $\alpha = \langle \alpha_1, \ldots, \alpha_n \rangle$ is a vector of $P(\sigma_I)$, then its interpretation is $I(\alpha) = \langle I(\alpha_1), \ldots, I(\alpha_n) \rangle$.) Therefore $\mathrm{Val}(P_I) = \mathrm{Val}(P_I')$ if either value is defined.

It might be hoped that strong equivalence would have a "compactness" property (i.e., that finite equivalence and strong equivalence should define the same relation on schemas); but this is not the case.

THEOREM 2.2. $P \equiv_F P' \not\Rightarrow P \equiv P'$.

Proof. Let P be the schema of diagram 2 and P' be the simple chain,

1. $L_1 := F(L_3)$,
2. $L_2 := F(L_3)$,
3. STOP.

A straightforward induction argument establishes that if σ_I is an infinite execution sequence through P, then the successive values of $I(T)$ on the sequence $F^2L_1, F^3L_1, F^4L_1, \ldots$ are 0, 1, 0, 1, 1, 0, 1, 1, 1, 0,... (F^nL_1 denotes F concatenated with $F^{n-1}L_1$). Therefore I cannot be a finite interpretation, and P halts on all finite interpretations with the same values as P'. Thus $P \equiv_F P'$ but $P \not\equiv P'$.

The four relations on schemas defined so far can be put in a sequence of strictly increasing strength.

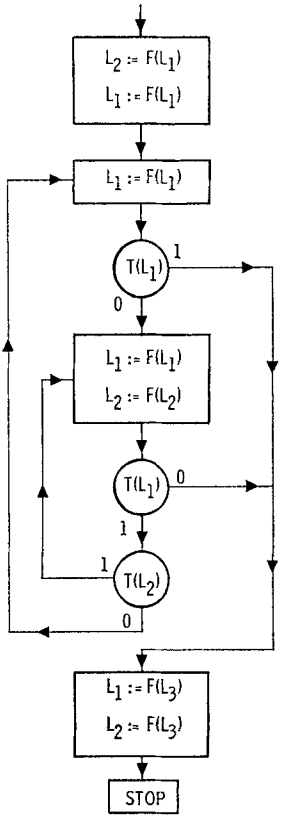


DIAGRAM 2

THEOREM 2.3.

$$\simeq \not\supseteq =_F \not\supseteq =_R \not\supseteq \equiv$$

Proof. The first strict inclusion is obvious, the second follows from the proof of Theorem 2.2, and the third is proved in [5].

Finally we note that although we have adopted an ALGOL-like formalism, it is possible to represent program schemas by systems of recursion equations without given (basis) functions, for example, McCarthy's system [4], so that the undecidability results of Section 4 hold true for the latter formalism too. Given a schema P with n locations $L_1, \ldots, L_n$, we construct recursion equations, $E(P)$, on n variables $\bar{X} = \langle x_1, \ldots, x_n \rangle$. First introduce a function symbol f_k for each operator F_k in P, and a predicate P_k for each transfer T_k. $E(P)$ contains equations defining functions $g_{i1}, \ldots, g_{in}$ for each instruction S_i as follows: if S_i is the assignment,

$$i.\ L_w := F_k(L_{u_1}, \ldots, L_{u_q})$$

$E(P)$ contains equations,

$$g_{ij}(\bar{X}) = g_{i+1,j}(x_1, \ldots, x_{w-1}, f_k(x_{u_1}, \ldots, x_{u_v}), \ldots, x_n), \qquad j \leqslant n;$$

for each transfer,

$$i.\ T_k(L_w)\ u, v,$$

$E(P)$ contains the conditional forms,

$$g_{ij}(\bar{X}) = (P_k(x_w) \rightarrow g_{uj}(\bar{X}), T \rightarrow g_{vj}(\bar{X})), \qquad j \leqslant n.$$

If S_i is the stop instruction, the corresponding equations are $g_{ij}(\bar{X}) = x_j$, $j \leqslant n$. The values of the functions $g_{i1}(\bar{X}), \ldots, g_{in}(\bar{X})$ when defined, are the final values assigned to $L_1, \ldots, L_n$, respectively, in a halting computation of P starting at S_i under the interpretation I such that $I(L_j) = x_j$, $I(F_k) = f_k$ and $I(T_k) = P_k$. It is easily seen that $P \equiv P'$ if and only if, for every $i \leqslant n$, $g_{1i} = g'_{1i}$ is true.

3. Multitape and Multihead Automata

Rosenberg [7], and independently the authors, has discovered a number of undecidability results concerning "two-headed" automata. It turns out that two-location schemas may be used to "simulate" these automata, in a certain sense, and we will use this result to establish the undecidability of a number of forms of the halting problem and equivalence problem for program schemas.

The *multihead automata* considered by Rosenberg are closely related to the multitape one-way finite automata introduced by Rabin and Scott ([6], Definition 15). One obtains a multihead automaton from a multitape automaton by requiring that the tapes be identical. Clearly such a machine can also be regarded as a one-tape automaton, with the heads independently scanning the tape, all being positioned initially over the start of the tape.

A multitape or multihead automaton, A, is specified by an alphabet, $\Sigma = \{b_1, b_2, \ldots, b_n\}$, a finite set of states, Q, and a transition table. Q is partitioned into a set of *live* states, $Q' = \{q_0, q_1, \ldots, q_m\}$ with a distinguished initial state, q_0, and a set of *dead* states, $Q - Q'$, with a distinguished *accept* state, $\underline{a}$. Q' is partitioned into disjoint subsets, Q_j, one for each reading head (the heads being on separate tapes in the multitape case, and on the same tape in the multihead case); if $q_i \in Q_j$ we shall use the notation q_i^j to indicate that A reads the symbol scanned by the j-th head when in state q_i. The rows of the transition table of A have the form,

$$q_i^j, s_1 \rightarrow r_{i1}, \qquad q_i^j, s_2 \rightarrow r_{i2}, \ldots \qquad q_i^j, s_n \rightarrow r_{in}$$

for $0 \leqslant i \leqslant m$. Each row specifies the next state $r_{ik} \in Q$ depending on the symbol s_k read from head j. After reading a symbol, the head j moves on to the next symbol on the tape (the head may only move in one direction). On reaching a dead state the automaton "stops".

Our automata differ in one important respect from those of Rabin and Scott or Rosenberg; they are considered as operating on infinite tapes (elements of Σ^ω; i.e., infinite sequences from Σ). For each tape there are three possibilities distinguished here. If $\underline{a}$ is reached, the tape is *accepted.* If any other dead state is reached, the tape is *rejected.* Otherwise, the automaton *diverges* on the tape. An automaton A therefore determines a threefold partition of Σ^ω into $\mathscr{I}_a(A)$, $\mathscr{I}_r(A)$, $\mathscr{I}_d(A)$, the classes of tapes which A accepts, rejects, and on which it diverges.

Given a Turing machine M and an initial tape configuration IT, there is an effective method for obtaining a *two-headed* automaton A which checks that its tape describes the computation of M from IT. A tape is accepted if it has an initial segment which, subject to certain conventions, "describes" a completed computation of M from IT; A diverges if the tape "describes" a nonterminating computation of M from IT; otherwise the automaton reaches a dead state $\underline{r}$, and the tape is rejected. To "describe" the computation, in the sense intended here, the segment of tape takes the form of the sequence of "instantaneous descriptions" (following Davis [1]) of successive steps of the computation, satisfying (for future convenience) the following constraints:

(a) The initial description of IT is fixed in such a way that the initial state symbol is not its first or last symbol.

(b) Each subsequent description is obtained by adding a "blank" symbol to both ends of the description which results from applying one of the rules of M to its predecessor (and which is the same length).

(c) Descriptions are separated by some symbol β not in the alphabet of M.

The action of A is as follows [for an alternative description, see Rosenberg [7] Theorem 9(a)]: A first checks the initial description with head 1; A then compares each description in turn against the previous one, using head 1 to scan the succeeding description while head 2 scans the predecessor. After each comparison the heads are correctly positioned for the next. The comparison of two descriptions proceeds as follows: a blank is read by head 1; symbols from the two descriptions are then compared in turn, until a state symbol is read under either head; the succeeding pairs of symbols in the two descriptions are then read. The tape is rejected unless the resulting triples satisfy the appropriate rule of M; for example if Q_j, Q_l are state symbols of M, then the pair must be of the form $\langle Q_j s_k s_p , s_k Q_l s_p \rangle$ or $\langle s_p Q_j s_k , Q_l s_p s_k \rangle$ or $\langle Q_j s_k s_p , Q_l s_i s_p \rangle$, depending on whether the appropriate quadruple (in the notation of Davis [1]) is $Q_j s_k R Q_l$ or $Q_j s_k L Q_l$ or $Q_j s_k s_i Q_l$. If the triple from head 1 indicates

a terminal configuration, the tape is accepted; otherwise symbols from the two descriptions are compared again until a β is read under head 2; head 1 then reads a "blank" and a β, and the comparison process is entered again. During this process there are only a fixed finite number of situations which need be distinguished, and so given M and IT, we can construct a finite automaton A to perform the comparison. Moreover this construction is effective on M and IT. Further details are omitted here.

The automaton A has the properties:

$$\mathscr{I}_a(A) = \varnothing \Leftrightarrow \mathscr{I}_d(A) \neq \varnothing$$
$$\Leftrightarrow M \text{ fails to halt from } IT.$$

It is well-known that the last property is not partially decidable. This establishes:

THEOREM 3.1. *For two-headed automata A, the properties*

(a) $\mathscr{I}_a(A) = \varnothing$,

(b) $\mathscr{I}_d(A) \neq \varnothing$,

are not partially decidable.

The simulation of automata by schemas is most easily applied to *binary* automata, automata over the alphabet $\{0, 1\}$. 3.1 must first be established for binary automata A alone.

Notation. Denote by $\underline{i}$, for any $i \geqslant 0$, the string consisting of i occurrences of "1" followed by one "0".

For strings, tapes, sets of tapes, etc. over $\Sigma = \{s_1, s_2, \ldots, s_n\}$ let $\mathscr{B}$ be the transformation to binary strings, tapes, etc., generated, in the natural way, by taking

$$\mathscr{B}(s_i) = \underline{i}$$
$$\mathscr{B}(s_i w) = \underline{i}\mathscr{B}(w) \text{ for strings of the form } s_i w, \text{ etc.}$$

LEMMA 3.2. *There is an effective transformation $\bar{\mathscr{B}}$ from automata over Σ to binary automata, such that*

(a) $\mathscr{I}_a(\bar{\mathscr{B}}(A)) = \varnothing \Leftrightarrow \mathscr{I}_a(A) = \varnothing$,

(b) $\mathscr{I}_d(\bar{\mathscr{B}}(A)) = \mathscr{B}(\mathscr{I}_d(A))$.

Proof. Rows of the table specifying A have the form

$$q_i^j, s_1 \to r_{i1}, \qquad q_i^j, s_2 \to r_{i2}, \ldots, \qquad q_i^j, s_n \to r_{in}.$$

The table for $\bar{\mathscr{B}}(A)$ is to be obtained by including, for each such row of A's table the $n+1$ rows

$$\begin{aligned} &q_i^j, 0 \to \underline{r}', && q_i^j, 1 \to q_{i1}^j, \\ &q_{i1}^j, 0 \to r_{i1}, && q_{i1}^j, 1 \to q_{i2}^j, \\ &q_{i2}^j, 0 \to r_{i2}, && q_{i2}^j, 1 \to q_{i3}^j, \\ &\cdots \\ &q_{ij}^j, 0 \to r_{in}, && q_{in}, 1 \to \underline{r}'', \end{aligned}$$

where the q_{ik}^j, $1 \leqslant k \leqslant n$, are new states, not occurring in the table for A, and $\underline{r}', \underline{r}''$ are (for future convenience) additional dead states. Otherwise the live and dead states of $\bar{\mathscr{B}}(A)$ are those of A.

Clearly, A, starting from state q_i, with its heads over final segments t_1, t_2 of its tape, moves immediately to state r_{ik} with its heads over t_1', t_2' if and only if $\bar{\mathscr{B}}(A)$ starting from state q_i over $\mathscr{B}(t_1), \mathscr{B}(t_2)$ moves, after $k+1$ state transitions, to state r_{ik} with its heads over $\mathscr{B}(t_1'), \mathscr{B}(t_2')$. So, by induction on the number of state transitions of A, A accepts or diverges on t if and only if $\bar{\mathscr{B}}(A)$ accepts or diverges on $\mathscr{B}(t)$. Hence $\mathscr{I}_d(\bar{\mathscr{B}}(A)) \supseteq \mathscr{B}(\mathscr{I}_d(A))$ and $\mathscr{I}_a(\bar{\mathscr{B}}(A)) \supseteq \mathscr{B}(\mathscr{I}_a(A))$.

On the other hand, if a binary tape t' is not rejected by $\bar{\mathscr{B}}(A)$, neither $\underline{r}'$ nor $\underline{r}''$ are reached, so that the initial segment scanned is an initial segment of some $\mathscr{B}(t)$. If $\bar{\mathscr{B}}(A)$ accepts t', then $\bar{\mathscr{B}}(A)$ also accepts $\mathscr{B}(t)$, which differs from t' only on an unscanned portion; hence A accepts t. Therefore $\mathscr{I}_a(\bar{\mathscr{B}}(A)) \neq \varnothing \Rightarrow \mathscr{I}_a(A) \neq \varnothing$. If $\bar{\mathscr{B}}(A)$ diverges on t', then $t' = \mathscr{B}(t)$ and A diverges on t. Therefore $\mathscr{I}_d(\bar{\mathscr{B}}(A)) \subseteq \mathscr{B}(\mathscr{I}_d(A))$. Lemma 3.2 follows.

4. Basic Undecidability Results

The way in which schemas may be used to simulate two-headed binary automata can now be described. For these results it will suffice to consider schemas involving one monadic function letter F, one transfer letter T, and registers $L_1, L_2, \ldots$. The notions "interpretation", "expression", etc. are to be understood as limited to interpretations, expressions, etc. arising in connection with such schemas.

Notation. $\mathscr{S}_i$ denotes the class of schemas containing at most the storage locations $L_1, L_2, \ldots, L_i$.

Definition. Let E be an expression and I an interpretation, the *I-tape* of E is the binary tape

$$t_I(E) = \epsilon_0\epsilon_1\epsilon_2, \ldots,$$

where

$$\epsilon_0 = T_I(I(E))$$

and

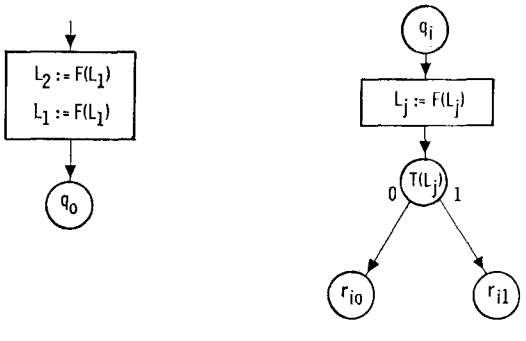
$$\epsilon_i = T_I(F_I^i(I(E))) \qquad i \geqslant 1.$$

Given a binary two-headed automaton B, the construction described below is to produce an incomplete schema, denoted by $P(B)$, which contains a pair of instructions (an assignment followed by a transfer) corresponding to each live state of B, and is incomplete in that some of its transfer addresses are represented by "symbolic labels" to be specified when the schema is completed. For any interpretation I, $P(B)$ simulates the behaviour of B on the tape $t_I(F(L_1))$ in the following sense. The computation of B is the state-symbol sequence, $q_0\epsilon_1 q_{i_1}\epsilon_2 q_{i_2}\epsilon_3 \cdots$ if and only if the execution sequence σ_I of $P(B)$ is the sequence of those pairs of instructions corresponding to $q_0, q_{i_1}, q_{i_2}, \ldots$. $P(B)$ eventually reaches a label corresponding to the dead state reached by B on $t_I(F(L_1))$, or diverges if B diverges on this tape. By incorporating $P(B)$ in other schemas, various equivalence-preserving transformations of automata into schemas will be obtained.

Suppose the rows of the table specifying B are

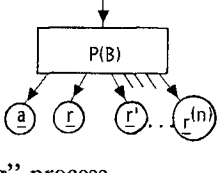
$$q_i^j, 0 \to r_{i0}\ ,\ q_i^j, 1 \to r_{i1}$$

for $0 \leqslant i \leqslant N$. Then $P(B)$ is to be obtained by combining the following $N + 2$ incomplete schemas



using the states of B as labels. $P(B)$ results from identifying labels in the above incomplete schemas, and has "terminating labels" corresponding to the dead states of B.

If the dead states of B are $\underline{a}, \underline{r}, \underline{r}', \ldots, \underline{r}^{(n)}$, we use the notation



for the result of this "combining" process.

It is easy to show, by induction on j, that if $P(B)$ reaches label q_i after executing $j+2$ assignment statements, and at that point L_1, L_2 "hold" expressions E_1, E_2 respectively, then, after j state-transitions of B on $t_I(F(L_1))$, B is in state q_i, with its heads over $t_I(E_1)$, $t_I(E_2)$ respectively. Therefore $P(B)$ simulates the action of B on $t_I(F(L_1))$ in the sense outlined above.

On the other hand, given a binary tape $t = \epsilon_1\epsilon_2\cdots$, for B, $P(B)$ can simulate the action of B on it, by choosing I with the property that $t = t_I(F(L_1))$. To achieve this, I may be chosen as any free interpretation (whose domain is the class of expressions), with $T_I(F^{i+1}L_1) = \epsilon_i$, $i \geqslant 1$. Therefore all and only computations of B are simulated by $P(B)$.

Notation. Denote by D the trivial schema,

1. $T(L_1), 1, 1.$

Denote by Z the schema,

1. $L_1 := F(L_3),$
2. $L_2 := F(L_3),$
3. STOP.

THEOREM 4.1. *The following properties of schemas P in $\mathscr{S}_2$ are not partially decidable.*

(a) *P diverges under all interpretations.*

(b) *P diverges under all finite interpretations.*

(c) *P diverges under some interpretations.*

(d) *P halts under all finite interpretations.*

(e) $P \equiv D.$

(f) $P \equiv_F D.$

The following are not partially decidable for schemas P in $\mathscr{S}_3$.

(g) $P \not\equiv Z.$

(h) $P \equiv_F Z.$

(i) $P \simeq Z.$

Proof. (a) Consider the schema $P_1(B)$ of Diagram 3. This is effective on B, and diverges for all interpretations if and only if $\mathscr{I}_a(B) = \varnothing$. The result follows from 3.1(a) and 3.2(a).

(b) is equivalent to (a). If P ever halts it does so on a finite interpretation.

(c) The schema $P_2(B)$ of Diagram 3 is effective on B, and diverges for some interpretation if and only if $\mathscr{I}_d(B) \neq \varnothing$. The result follows from 3.1(b) and 3.2(b).

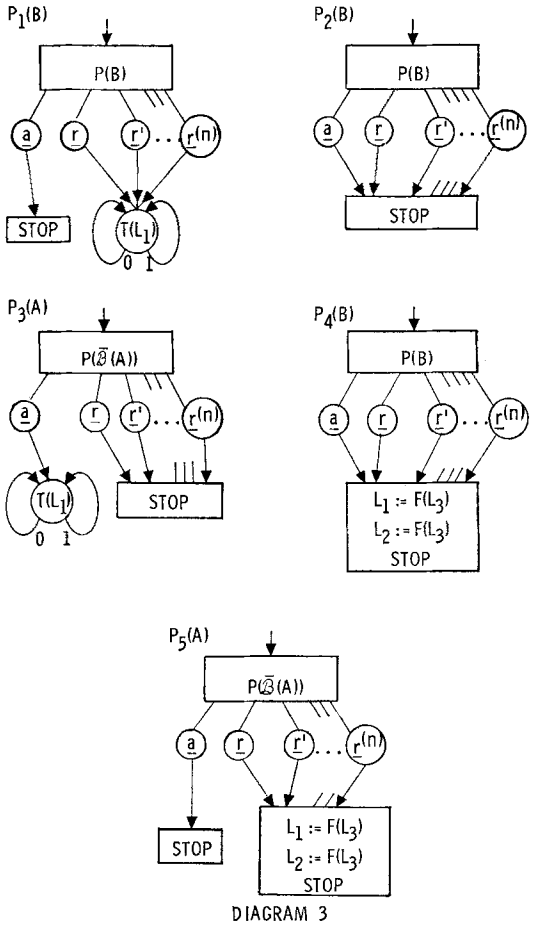


DIAGRAM 3

(d) Recall the automaton A constructed for 3.1, and effective on a Turing machine and initial configuration. A could diverge only if its tape described a divergent Turing machine computation. Note that A does not diverge on an ultimately periodic tape since the number of symbols between successive occurrences of the delimiter β grows without bound. Therefore A halts on every ultimately periodic tape. 3.1 therefore holds when restricted to automata A with this property. The transformation $\bar{\mathscr{B}}$ of 3.2 preserves this property of A. Finally, note that if I is a finite interpretation, then $t_I(E)$ is ultimately periodic, for any E, since for some $i \neq j, F_I^i(I(E)) = F_I^j(I(E))$. So $P(\bar{\mathscr{B}}(A))$ can diverge on no finite interpretation, if A has the above property. Also if $\underline{a}$ can be reached in this schema, it can be reached in a finite interpretation. Therefore, if A halts on every ultimately periodic tape, then the schema $P_3(A)$ of

diagram 3 halts under all finite interpretations if and only if $\mathscr{I}_a(\bar{\mathscr{B}}(A)) = \varnothing$. The result follows from 3.1 restricted as above, and 3.2(a).

(e), (f) are equivalent to (a), (b).

(g) The schema $P_4(B)$ is effective on B, and $P_4(B) \not\equiv Z \Leftrightarrow \mathscr{I}_d(B) \neq \varnothing$. Hence the result, as in (c).

(h), (i) Apply the argument of (d) above to $P_5(A)$; $P_5(A) \simeq Z \Leftrightarrow P_5(A) \equiv_F Z \Leftrightarrow \mathscr{I}_a(\bar{\mathscr{B}}(A)) = \varnothing$. If $\underline{a}$ can be reached, it can be reached in a finite interpretation which spoils equivalence ($\equiv_F$ or $\simeq$) to Z, by suitable choice of $I(L_3)$.

Note that $P \simeq D$ is always true, therefore decidable. It is worth noting that the complementary properties to those of 4.1 are all partially decidable. These are implied by the following results, which hold for schemas P without restriction.

THEOREM 4.2. *Let Q be any schema which halts under all interpretations, and R be any schema. The following properties of schemas P are partially decidable.*

(a) *P halts under some interpretation.*

(b) *P halts under some finite interpretation.*

(c) *P halts under all interpretations.*

(d) *P diverges under some finite interpretations.*

(e) $P \not\equiv D$.

(f) $P \not\equiv_F D$ (*a special case of* (*h*)).

(g) $P \equiv Q$.

(h) $P \not\equiv_F R$.

(i) $P \not\simeq R$.

Proof (in outline). On a finite interpretation I of n elements, P may be viewed as a finite state machine with $p \cdot n^k$ states, where k is the number of locations and p the number of instructions in P. Each property except (c) and (g) may be verified by enumerating the class of finite interpretations until one is found with an appropriate property. The tests applied to each finite interpretation are recursive, by well known properties of finite-state machines. The tests needed are, for (a), (b), (d), (e), (f) tests for halting or divergence, for (h), (i) tests for "strong" and "weak" equivalence, respectively, for finite-state machines (defining these notions by analogy with those defined above).

To verify properties (c) and (g), one can effectively enumerate all possible execution sequences through P. If P converges on all interpretations, this process terminates (one may list finite paths from the start of P, discarding any which are inconsistent in the sense of Section 2, and only choosing to examine paths which extend previously retained ones; if this process continues indefinitely, the hypotheses of Koenig's

Lemma are satisfied, implying that there is an infinite consistent path, and hence a possible divergence, by Section 2.) Property (c) is then verified. For property (g), one must check that terminal expressions are identical for consistent pairs of execution sequences through P and Q.

Note that the equivalence problem for the class of schemas which always halt *is* decidable, although membership of this class is not.

5. 'Reasonable' Notions of Equivalence

Theorem 4.1(e), (f), (h), (i) established the basic results concerning the partial undecidability of equivalences between schemas. In this section some more general results will be obtained. Firstly (5.1), the basic result is extended to cover a wider class of equivalence concepts. Secondly (5.2), this result is established also for a more restricted class of schemas, schemas which halt on all finite interpretations.

It seems reasonable to assume of a general notion of equivalence between schemas, both that it should hold between schemas which behave identically on all interpretations, finite or infinite, i.e., between schemas which are strongly equivalent, and that it should fail to hold between schemas which can demonstrably produce different results on the same interpretation (finite, if need be), i.e., between schemas which are not weakly equivalent. Some equivalence notions which do not satisfy these constraints are the various notions of "provable" equivalence (relative to some recursively axiomatisable formal system). In particular, "provable" strong equivalence is strictly stronger than strong equivalence, since the device of enumerating theorems of the formal system would otherwise constitute a partial decision procedure for $P \equiv D$, in contradiction to 4.1(e). We will not pursue these notions here.

Definition. Let $P \sim Q$ be any relation between schemas P, Q. $\sim$ *is reasonable on* a class of schemas, $\mathscr{S}$, if, for all P, Q in $\mathscr{S}$,

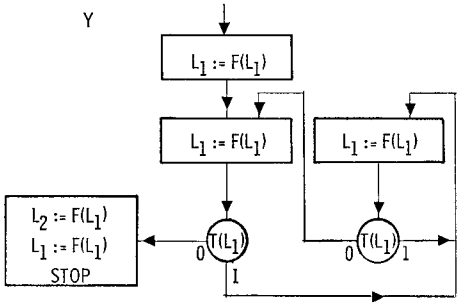
(i) $P \equiv Q \Rightarrow P \sim Q$,

(ii) $P \sim Q \Rightarrow P \simeq Q$.

Note that $\equiv$, $\equiv_R$, $\equiv_F$, $\simeq$ are reasonable relations on the class of all schemas. A reasonable relation is not assumed to be symmetric or transitive, although it must be reflexive, by virtue of (i). For example, the relation $\simeq$ is symmetric but not transitive. The relation $>$ is transitive but not symmetric, where $P > Q$ if, whenever $\mathrm{Val}(Q_I)$ is defined, then so is $\mathrm{Val}(P_I)$, and $\mathrm{Val}(P_I) = \mathrm{Val}(Q_I)$.

Definition. A binary tape (or finite string) is *terminated* if it starts with "0", or contains two adjacent occurrences of "0".

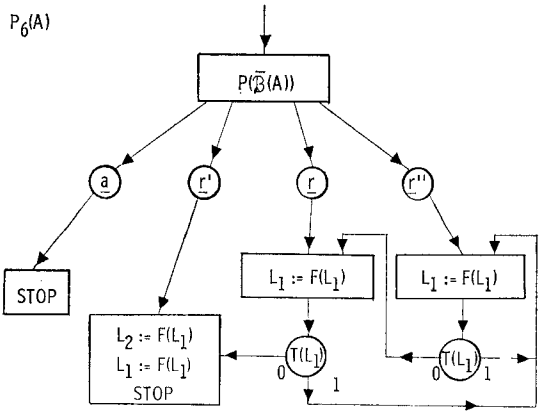
Let Y denote the following schema:



Then clearly Y diverges just on those interpretations I for which $t_I(F(L_1))$ is unterminated.

THEOREM 5.1. *If $\sim$ is any reasonable relation on $\mathscr{S}_2$, then $P \sim Y$ is not partially decidable.*

Proof. Let A be any two-headed automaton constructed in 3.1: Recall that A has just one rejecting state $\underline{r}$, and in any computation of A, at no time does head 2 lead head 1.



Consider now the schema $P_6(A)$, effective on A. ($\underline{r}'$, $\underline{r}''$ are the dead states introduced by the construction $\bar{\mathscr{B}}$ of 3.2). Note that on any interpretation I for which $\bar{\mathscr{B}}(A)$ does not accept $t_I(F(L_1))$, $P_6(A)$ and Y are arranged to produce the same results, or to diverge together in case $t_I(F(L_1))$ is not terminated; note that $\underline{r}'$ is reached if a termination is encountered in the computation of $\bar{\mathscr{B}}(A)$; if $\underline{r}$ is reached, the preceding symbol was "0", or no symbol has been read; if $\underline{r}''$ is reached, the preceding symbol was "1". Therefore

$$\mathscr{I}_a(\bar{\mathscr{B}}(A)) = \varnothing \Rightarrow P_6(A) \equiv Y. \tag{1}$$

On the other hand, suppose $\bar{\mathscr{B}}(A)$ accepts some tape t. Choose a free interpretation I such that $t_I(F(L_1))$ agrees with t on the segment scanned by $\bar{\mathscr{B}}(A)$, and is terminated (at some point beyond the segment). Then both $\mathrm{Val}(Y_I)$, $\mathrm{Val}(P_6(A)_I)$ are defined, and $\mathrm{Val}(Y_I) \neq \mathrm{Val}(P_6(A)_I)$. Therefore

$$\mathscr{I}_a(\bar{\mathscr{B}}(A)) \neq \varnothing \Rightarrow P_6(A) \not\doteq Y. \tag{2}$$

But if $\sim$ is a reasonable relation, then from the definition and (1), (2)

$$\mathscr{I}_a(\bar{\mathscr{B}}(A)) = \varnothing \Rightarrow P_6(A) \sim Y,$$
$$\mathscr{I}_a(\bar{\mathscr{B}}(A)) \neq \varnothing \Rightarrow P_6(A) \not\sim Y.$$

So that $\mathscr{I}_a(\bar{\mathscr{B}}(A)) = \varnothing \Leftrightarrow P_6(A) \sim Y$. The former property of A is not partially decidable by 3.1, 3.2, and $P_6(A)$ is effective on A. The result follows.

It is essential to 5.1 that Y be a schema which diverges on some interpretation, since if Q always halts, $P \equiv Q$ is partially decidable, from 4.2(g). However, we remark in passing that even in this case at least one of $P \sim Q$, $P \not\sim Q$ is not partially decidable, for any reasonable $\sim$. The proof (details omitted) proceeds by describing an effective construction, given an integer x, of a schema P_x such that

$$\varphi_x(x) = 0 \Rightarrow P_x \equiv Q$$

and

$$\varphi_x(x) = 1 \Rightarrow P_x \not\doteq Q,$$

$\varphi_0, \varphi_1, \varphi_2, \ldots$ being a Godel-numbering of all partial recursive functions. If $P_x \sim Q$ is decidable, there is a total recursive function φ_z such that

$$P_x \sim Q \Rightarrow \varphi_z(x) = 1$$

and

$$P_x \not\sim Q \Rightarrow \varphi_z(x) = 0.$$

But then $\varphi_z(z) = 0 \Leftrightarrow \varphi_z(z) = 1$. Essentially this is a proof of the "recursive inseparability" of the sets $\{P \mid P \equiv Q\}$ and $\{P \mid P \not\doteq Q\}$, and in fact goes through for any Q which halts on some interpretation (though of course not if Q always diverges).

In the following result we show that the fact that Y diverges on some interpretations is not necessarily a flaw in the stronger result 5.1. What is shown is that an analogue of 5.1 holds over a particular class of schemas which halt on all finite interpretations. Membership in this particular class is decidable. It follows that it could not be a subclass of any class of schemas which escapes the nonpartial-decidability result; so that any such class must exclude some schemas which, among other properties, always halt when interpreted as digital computer programs.

THEOREM 5.2. *There is a schema $\overline{Y}$, and a class $\mathscr{P} \subseteq \mathscr{S}_2$ of schemas such that*

(i) *$\mathscr{P}$ consists of schemas which halt on all finite interpretations.*
(ii) *$P \in \mathscr{P}$ is decidable.*
(iii) *$\overline{Y} \sim P$ is not partially decidable, for any reasonable $\sim$, and for $P \in \mathscr{P}$.*

Proof. As in 5.1 we must find an automaton $\bar{A}$ and a class of automata $\{A'\}$ which can check Turing machine computations, such that any A' always has the action of $\bar{A}$, unless its Turing machine computation converges. In addition, $\bar{A}$ must converge on all tapes $t_I(E)$, such that I is a finite interpretation.

In the proof of 4.1(d) we pointed out that the automata A used in 3.1 had this latter property, since successive segments between occurrences of the marker symbol, β, increase in length (by exactly two symbols) in any tape on which A diverges, and also that $\bar{\mathscr{B}}$ preserves this property so that such a tape cannot correspond to a finite interpretation. So we might choose $\bar{A}$ to be an automaton which checks the spacing of β's along its tape, diverging if the spacing grows in the correct way, and rejecting if it does not. However, there are two further considerations:

(a) Since $\bar{A}$ has a finite alphabet, the class of Turing machine computations checked by $\{A'\}$ must have a finite number of states and symbols, and still have an undecidable halting problem. This can be achieved by restricting the class to computations of a particular universal Turing machine M_0. The alphabet Σ_0 of $\bar{A}$ and of the automata $\{A'\}$ then consists of tape-symbols and state-symbols for M_0, together with the delimiter symbol β. We will suppose $\Sigma_0 = \{\beta, s_2, s_3, \ldots, s_n\}$. The design of the A' is then to be effective on an initial configuration IT, and have the property that

$$\mathscr{I}_a(A') = \varnothing \Leftrightarrow M_0 \text{ diverges on } IT.$$

The latter property, again, is well-known not to be partially decidable.

(b) The initial instantaneous descriptions checked by A' are to be unbounded in length as A' varies. One must avoid constructing $\bar{A}$ so that it would diverge, say, on a tape containing no occurrences or β, which it would be forced to do if the tapes for A' were to have the form assumed for the construction of 3.1. To avoid this difficulty, $\bar{A}$, A' are to obey the following rules.

For any string w over Σ_0, let $l(w)$ denote the length of w.

(i) $\bar{A}$ is to diverge just on tapes of the form

$$t = w_0\beta w_1\beta w_2\beta \cdots,$$

where the w_i are strings over $\Sigma_0 - \{\beta\}$, and $l(w_i) = 2i + 1$, $i \geqslant 0$. $\bar{A}$ rejects every other tape.

(ii) Given IT, choose in some effective manner an initial instantaneous descrip-

tion w of IT such that $l(w) = 2K + 1$, for suitable K. Then A' is to accept any tape with an initial segment of the form

$$w_0\beta w_1\beta \cdots w_{N-1}\beta w_N$$

where the w_i are strings over $\Sigma_0 - \{\beta\}$, and $l(w_i) = 2i + 1$, $0 \leqslant i < N$, such that $w = w_K$, and $w_K\beta w_{K+1}\beta \cdots \beta w_N$ describes a complete computation of M_0 from IT according to the constraints sketched in the proof of 3.1. On any other tape, A' behaves as $\bar{A}$.

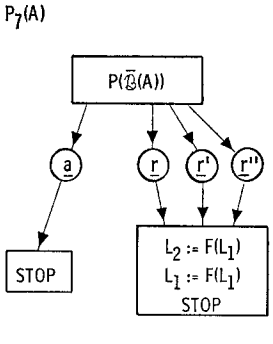
The transition table for $\bar{A}$ can now be specified. This consists of the following rows:

$$\begin{array}{ll} q_0^1, \beta \to \underline{r}, & q_0^1, s_i \to q_1, \\ q_1^1, \beta \to q_2, & q_1^1, s_i \to \underline{r}, \\ q_2^1, \beta \to \underline{r}, & q_2^1, s_i \to q_3, \\ q_3^2, \beta \to q_0, & q_3^2, s_i \to q_2, \end{array} \qquad 2 \leqslant i \leqslant n.$$

A' will only be described insofar as we contrast its action with that of A in the proof of 3.1. A' first checks with head 1 that its tape has an initial segment $w_0\beta w_1\beta \cdots w_{K-1}\beta$, satisfying the above conditions; if not, state $\underline{r}$ is entered. In the meantime head 2 keeps pace with 1. If this initial check succeeds, both heads are then over the start of w_K. From this position A' simulates the computation of A (in 3.1) unless a move to the reject state, $\underline{r}$, is reached. If β is encountered anywhere but as the terminator of a complete instantaneous description, or if an instantaneous description is not terminated by β, A' moves to state $\underline{r}$; otherwise, wherever A moves to $\underline{r}$, A' continues to compute, but now simulates $\bar{A}$ from that point on. Just as in 3.1, there is a finite number (depending effectively on IT) of situations which need to be distinguished in the operation of A', so that A' can be designed as a finite-state automaton.

Note that, if a tape is rejected by both $\bar{A}$ and A', head 1 comes to rest at the same point on both tapes; this holds also for $\bar{\mathscr{B}}(\bar{A})$, $\bar{\mathscr{B}}(A')$.

Now let $P_7(A)$ be the transformation from automata A into schemas of the following form



Let $\overline{Y} = P_7(\bar{A})$. Then, from the above construction,

$$M_0 \text{ diverges from } IT \Rightarrow \mathscr{S}_a(\bar{\mathscr{B}}(A')) = \varnothing \Rightarrow P_7(A') \equiv \overline{Y}.$$

On the other hand, if M_0 halts from IT, then A' accepts some tape t which $\bar{A}$ rejects; for example, let t be a tape which departs from the format rule (i), and has an initial segment describing the computation from IT. But then, choosing I such that $t_I(F(L_1)) = t$, both of $\mathrm{Val}(P_7(A')_I)$ and $\mathrm{Val}(\overline{Y}_I)$ are defined (and differ), so that $P_7(A') \not\simeq \overline{Y}$. Therefore M_0 halts from $IT \Rightarrow P_7(A') \not\simeq \overline{Y}$.

Combining the results of the last two paragraphs, we have M_0 diverges from $IT \Leftrightarrow P_7(A') \sim \overline{Y}$, for any reasonable $\sim$, arguing as in 5.1. This establishes 5.2.

We end this section with a result which summarises the consequences of 5.1, 5.2 so far as program simplification is concerned. Clearly any simplification algorithm which enables one to obtain a canonical form for schemas, under a reasonable $\sim$ which is an equivalence relation in the strict sense, is excluded by 5.1, 5.2, under not very stringent conditions on $\sim$ which would permit such an algorithm to be embodied in a partial decision procedure for $P \sim Q$. On the other hand, one might expect any "exhaustive" simplification algorithm to provide such a canonical form, by reducing a schema to its "simplest" equivalent. Below we point out certain assumptions about $\sim$ and about the simplification method which enable this argument to go through. Not every reasonable $\sim$ will satisfy these assumptions. The algorithm which consists of choosing D, the one-instruction schema which halts on no interpretation, is an exhaustive simplification algorithm with respect to $\simeq$, in a suitable sense of "simplification", so that $\simeq$ will not satisfy the assumptions to be given. We assume as little as we can about the notion of "simplicity" involved.

By a *simplification method for* $\sim$ we will understand an effective method, given P, of listing an infinite sequence $P', P'', \ldots$ of (not necessarily distinct) "simplifications" of P such that for all n, $P \sim P^{(n)}$. An *exhaustive* method is one such that whenever $P \equiv Q$, there exist m, n such that $P^{(m)} = Q^{(n)}$. In particular, if some method always produces a "simplest" schema, if $\sim$ is an equivalence relation, and if $P \sim Q$ is (partially) decidable for "simplest" schemas, then it can be extended to one which is exhaustive in the above sense. Similar conditions can be formulated for those cases in which there do not necessarily exist "simplest" versions of schemas.

THEOREM 5.3. *If $\sim$ is reasonable, and*

$$P \sim R \,\&\, Q \sim R \Rightarrow P \simeq Q, \qquad (*)$$

then there is no exhaustive simplification method for $\sim$.

Proof. Suppose there is such a method; we obtain a contradiction to 5.2 (or alternatively to 5.1). Note first that for $P \in \mathscr{P}$, $\overline{Y} \sim P \Leftrightarrow \overline{Y} \equiv P \Leftrightarrow \overline{Y} \simeq P$.

Consider the partial procedure of simultaneously listing the sequences P', P'',..., and $\bar{Y}'$, $\bar{Y}''$,..., and checking for a common element. If $\bar{Y} \sim P$, then $\bar{Y} \equiv P$, so the process is successful for P, from the assumption of exhaustiveness; on the other hand if the process succeeds for P, then $\bar{Y} \simeq P$, from (*), but then $\bar{Y} \sim P$ also. Hence $\bar{Y} \sim P$ if and only if the process succeeds for P, contra 5.2. This establishes 5.3.

Note that (*) holds for any reasonable $\sim$ stronger than the relation "$<_F$", where $P <_F Q$ if $P \simeq Q$ and if Q halts on every finite interpretation on which P halts, but does not hold for $\simeq$, or for the relation $>_F$ which is the inverse of $<_F$ (though simplification with respect to these latter notions is not a particularly desirable end).

6. Monadic Program Schemas

As regards the goal of developing practical techniques for optimizing computer programs and formal systems for proving statements about them, the main results of the previous sections can only be looked on as negative. In this section we present some results in a more positive direction.

We first consider the relationship between program schemas and other abstract models of computational processes. Schemas all of whose operators are one-place symbols will be called *monadic schemas*. A particularly simple subclass of the monadic schemas is obtained if one makes the further restriction that all assignment statements have the property that the assignment location is the same as the retrieval location (e.g., $L_j := F_i(L_j)$); these are called the *independent location schemas*. This class of schemas turns out to be strongly related to the multitape Rabin–Scott automata [6]. We shall show that not only are the equivalence problems for multitape automata and independent location schemas interchangeable (i.e., either problem may be reduced to the other), but that there is an algorithm for constructing a schema which simulates the behaviour of a given automaton, and vice versa. Thus it is reasonable to claim that multitape automata are models for those computer programs in which the result of any computation on one set of data is never used by any computation on another set of data.

We represent the multitape Rabin–Scott automata as a subclass $\mathscr{RS}$ of the multitape automata defined in Section 3. The alphabets of automata in $\mathscr{RS}$ are to contain an "end-marker" symbol ϵ, and these automata have the properties:

(i) A computation by an automaton $A \in \mathscr{RS}$ terminates if and only if each tape contains an occurrence of ϵ.

(ii) Any such computation terminates with each head immediately past the first occurrence of ϵ on its tape.

$\mathscr{RS}$ then "represents" the Rabin–Scott automata of [6] in the obvious sense; there is an effective correspondence between the two classes with the property that

a Rabin–Scott automaton accepts or rejects the n-tuple $\langle \hat{t}_1, \hat{t}_2, \ldots, \hat{t}_n \rangle$ of finite tapes if and only if the corresponding automaton in $\mathscr{RS}$ accepts or rejects any $\langle t_1, t_2, \ldots, t_n \rangle$ of infinite tapes such that each t_i has an initial segment $\hat{t}_i\epsilon$.

DEFINITION. Two automata A, B are *equivalent* (written $A \equiv B$) if $\mathscr{I}_a(A) = \mathscr{I}_a(B)$ and $\mathscr{I}_r(A) = \mathscr{I}_r(B)$.

In particular for $A, B \in \mathscr{RS}$, $A \equiv B$ if and only if $\mathscr{I}_a(A) = \mathscr{I}_a(B)$, since $\mathscr{I}_r(A) = X - \mathscr{I}_a(A)$, where X is the set of all n-tapes over Σ each of whose components has an endmarker occurrence.

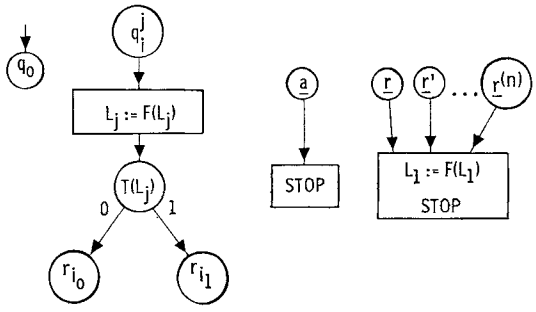
Firstly, to reduce the equivalence problem for $\mathscr{RS}$ automata to the (strong) equivalence problem for independent location schemas, we show how to construct, from an n-tape automaton $A \in \mathscr{RS}$ an n independent location schema $\tilde{P}(A)$. This construction proceeds along lines very similar to those in Sections 3, 4.

For $A \in \mathscr{RS}$, consider $\bar{\mathscr{B}}(A)$, where $\bar{\mathscr{B}}$ is the transformation of Lemma 3.2; note that for A, A'

$$A \equiv A' \Leftrightarrow \bar{\mathscr{B}}(A) \equiv \bar{\mathscr{B}}(A') \Leftrightarrow \mathscr{I}_a(\bar{\mathscr{B}}(A)) = \mathscr{I}_a(\bar{\mathscr{B}}(A')),$$

from properties (i), (ii) above. ((i) ensures that $A \equiv A' \Leftrightarrow \mathscr{I}_a(A) = \mathscr{I}_a(A')$; (ii) ensures that $\bar{\mathscr{B}}(A), \bar{\mathscr{B}}(A')$ enter $\underline{r}', \underline{r}''$ on precisely the same set of tapes, where these states are the dead states, introduced by $\bar{\mathscr{B}}$.)

Analogous to the construction of $P(B)$ in Theorem 4.1, let $\tilde{P}(A)$ be obtained from the table of the binary automaton $\bar{\mathscr{B}}(A)$ by identifying labels in the following incomplete schemas:



Now let $t = \langle t_1, t_2, \ldots, t_n \rangle$ be any binary n-tape and I any free interpretation, such that $t_I(L_i) = t_i$, $1 \leqslant i \leqslant n$. Clearly given any binary t there is an I related to it in this way, and vice versa. Let m_i be the length of the shortest initial segment of t_i of the form $\mathscr{B}(w\epsilon)$ for some word w. Assuming each m_i is defined, let $V(t)$ denote the n-tuple $\langle F^{m_1+1}L_1, F^{m_2+1}L_2, \ldots, F^{m_n+1}L_n \rangle$ of expressions. From the construction of $\tilde{P}(A)$, we have the following:

LEMMA 6.1.

(i) *$\bar{\mathscr{B}}(A)$ accepts t if and only if $\tilde{P}_I(A)$ halts and* $\mathrm{Val}(\tilde{P}_I(A)) = V(t)$.
(ii) *If $\bar{\mathscr{B}}(A), \bar{\mathscr{B}}(A')$ reject t, then $\tilde{P}_I(A)$ halts and* $\mathrm{Val}(\tilde{P}_I(A)) = \mathrm{Val}(\tilde{P}_I(A'))$.
(iii) *If $\bar{\mathscr{B}}(A)$ diverges on t, then $(\tilde{P}_I A)$ diverges.*

From 6.1, and the remark above concerning $\bar{\mathscr{B}}$ when restricted to $\mathscr{RS}$, we conclude:

LEMMA 6.2. *For $A, A' \in \mathscr{RS}$, $A \equiv A'$ if and only if $\tilde{P}(A) \equiv \tilde{P}(A')$. (More strongly, $A \equiv A' \Leftrightarrow \tilde{P}(A) \sim \tilde{P}(A')$, for any reasonable $\sim$, since $\tilde{P}(A) \equiv \tilde{P}(A') \Leftrightarrow \tilde{P}(A) \simeq \tilde{P}(A')$).*

The converse problem is to describe an effective equivalence-preserving transformation $\mathscr{A}$ from independent location schemas to $\mathscr{RS}$ automata. Let P be an independent location schema involving at most the location symbols $L_1, L_2, \ldots, L_n$, monadic operator symbols $F_1, F_2, \ldots, F_m$ and, for convenience, just one transfer symbol T (the construction below can be modified in an obvious way to cover schemas involving more than one transfer symbol). Let I be any free interpretation and suppose $\mathrm{Val}(P_I) = \langle E_1, E_2, \ldots, E_n \rangle$; then $\mathscr{A}(P)$ is to be an n-tape automaton over the alphabet $\Sigma = \{0, 1, L_1, L_2, \ldots, L_n, F_1, F_2, \ldots, F_m, \epsilon\}$ which accepts any n-tape $t = \langle t_1, t_2, \ldots, t_n \rangle$ satisfying a certain condition determined just by I and the expressions $E_1, E_2, \ldots, E_n$, and independent of P. This condition is that each t_i have an initial segment of the form $w(I, E_i)\epsilon$, where $w(I, E)$ for any expression E is determined as follows: Suppose E is the expression

$$F_{j_k} F_{j_{k-1}} \cdots F_{j_1} L_{j_0},$$

then $w(I, E)$ is the word

$$L_{j_0} \delta_0 F_{j_1} \delta_1 \cdots F_{j_k} \delta_k$$

such that $\delta_i = T_I(F_{j_i} F_{j_{i+1}} \cdots F_{j_1} L_{j_0})$. The essential properties which $\mathscr{A}(P)$ is to have are summarised in the following:

LEMMA 6.3. *$\mathscr{A}(P)$ accepts $\langle t_1, t_2, \ldots, t_n \rangle$, if and only if each t_i has an initial segment of the form $w(I, E_i)\epsilon$ such that I is a free interpretation, P_I halts and* $\mathrm{Val}(P_I) = \langle E_1, E_2, \ldots, E_n \rangle$.

The construction $\mathscr{A}$ is complicated by a technical difficulty; $\mathscr{A}(P)$ will be constrained to move any tape immediately after reading a symbol from it; consequently if more than one state transition is to be governed by a symbol on the tape, special steps must be taken to record the symbol in the automaton's internal state. In designing $\mathscr{A}(P)$ to simulate P, this difficulty occurs over those symbols specifying values for the branch function T_I. To overcome this, either P may be transformed first of all into an equivalent more restricted schema, in which locations are tested only

immediately after being assigned to (this is in effect the process of "freeing" a "liberal" schema, of Paterson [5]), or $\mathcal{A}(P)$ may be taken first of all as an automaton with some inessential additional facility for recording previously read symbols, or one could describe a direct transformation. The second approach is adopted here. We describe first of all an automaton $\mathcal{A}^*(P)$ which is an automaton in the sense of Section 3, but with the addition of certain "starred" states whose transformations obey special rules. These states are indicated below by the presence of a superscript "*" in transition table entries. The transitions of a state indicated by q^{i*}, say, depend on the previous character read from tape i; obeying such a transition does not involve any tape movement; if no character has yet been read from tape i, the "previous" character is assumed to be "0", say. The details of $\mathcal{A}^*(P)$ are as follows:

(i) To ensure that $\mathcal{A}(P)$ will be $\mathcal{RS}$, each tape is read up to its end-marker before $\mathcal{A}^*(P)$ stops. The following $3n$ rows do this:

$$
\left.\begin{array}{ll}
r_1^{1*}, \epsilon \to r_2, & r_1^{1*}, \xi \to \tilde{r}_1 \\
\tilde{r}_1^{\,1}, \epsilon \to r_2, & \tilde{r}_1^{\,1}, \xi \to \tilde{r}_1 \\
\vdots & \\
\tilde{r}_{n-1}^{\,n-1}, \epsilon \to r_n, & \tilde{r}_{n-1}^{\,n-1}, \xi \to \tilde{r}_{n-1} \\
r_n^{n*}, \epsilon \to \underline{r}, & r_n^{n*}, \xi \to \tilde{r}_n \\
\tilde{r}_n^{\,n}, \epsilon \to \underline{r}, & \tilde{r}_n^{\,n}, \xi \to \tilde{r}_n
\end{array}\right\} \xi \neq \epsilon.
$$

$$
\left.\begin{array}{ll}
a_1^{\,1}, \epsilon \to a_2, & a_1^{\,1}, \xi \to \tilde{r}_1 \\
\vdots & \\
a_{n-1}^{\,n-1}, \epsilon \to a_n, & a_{n-1}^{\,n-1}, \xi \to \tilde{r}_{n-1} \\
a_n^{\,n}, \epsilon \to \underline{a}, & a_n^{\,n}, \xi \to \tilde{r}_n
\end{array}\right\} \xi \neq \epsilon.
$$

All transitions not specified below are to r_1.

(ii) Corresponding to the commands $S_1, S_2, \ldots, S_p$ of P, $\mathcal{A}^*(P)$ has states $s_1, s_2, \ldots, s_p$ determined as follows:

(a) If S_j is a STOP instruction, s_j is the state

$$a_1 \text{ in (i).}$$

(b) If S_j is a transfer

$$j.\ T(L_i)\ u, v$$

s_j has the transitions

$$s_j^{i*}, 0 \to s_u, \quad s_j^{i*}, 1 \to s_v.$$

(c) If S_j is an assignment

$$j.\ L_i := F_k(L_i),$$

the corresponding rules are

$$\begin{aligned} &s_j^i, F_k \to \tilde{s}_j\ , \\ &\tilde{s}_j^i, \delta \to s_{j+1}\ , \end{aligned} \qquad \delta = 0, 1.$$

(iii) $\mathcal{A}^*(P)$ starts in state t_1 with rules

$$\begin{aligned} t_1^1, L_1 &\to \tilde{t}_1 \\ \tilde{t}_1^1, \delta &\to t_2 \\ &\vdots \\ \tilde{t}_{n-1}^{n-1}, \delta &\to t_n \\ t_n^n, L_n &\to \tilde{t}_n \\ \tilde{t}_n^n, \delta &\to s_1 \end{aligned} \qquad \delta = 0, 1.$$

$\mathcal{A}^*(P)$ is then obtained as a "starred" automaton with $5n + 2m + l$ states, where l, m are, respectively, the numbers of transfer and assignment instructions in P.

$\mathcal{A}^*(P)$ "simulates" P in an obvious sense: suppose at some point in a computation $P_I, L_1, L_2, \ldots, L_n$ "hold" $E_1', E_2', \ldots, E_n'$, respectively, and instruction j is about to be executed; then, if each t_i has an initial segment $w(I, E_i')$, a point is reached in the computation of $\mathcal{A}^*(P)$ on $\langle t_1, t_2, \ldots, t_n \rangle$ when the heads lie just past these initial segments, and $\mathcal{A}^*(P)$ is about to enter state s_j. Conversely, in any computation of $\mathcal{A}^*(P)$, whenever s_j is reached the heads lie just past initial segments of the form $w(I, E_i')$ such that there is a point in the computation P_I when the locations "hold" $E_1', E_2', \ldots, E_n'$ and instruction j is about to be executed. These two properties of $\mathcal{A}^*(P)$ are established by induction on the lengths of incomplete computations of P, $\mathcal{A}^*(P)$, respectively; 6.3 follows for $\mathcal{A}^*(P)$, considering just those j which are addresses of STOP instructions in P.

It suffices now to reduce $\mathcal{A}^*(P)$ to an $\mathcal{A}(P) \in \mathcal{RS}$ which accepts the same tapes. This is done as follows: The states of $\mathcal{A}(P)$ are to be $\underline{a}, \underline{r}$, and states $q_{q',\underline{x}}$ indexed by unstarred live states q of $\mathcal{A}^*(P)$ and n-tuples $\underline{x} \in \Sigma^n$. Let $f_{\underline{x}}(q)$ for arbitrary states q of $\mathcal{A}^*(P)$ and n-tuples $\underline{x} \in \Sigma^n$ be defined by

$$f_{\underline{x}}(q) = \begin{cases} p, \text{ if } q \text{ is starred, where } p \text{ is obtained from the rule } q^{i*}, \underline{x}(i) \to p; \\ q, \text{ otherwise.} \end{cases}$$

Let N be the number of starred states in $\mathcal{A}^*(P)$. Note that, if $f_{\underline{x}}^N(q)$ is unstarred, then it is the first unstarred state to be entered in the sequence of transitions which

starts at q when the last character read by each head i is $\underline{x}(i)$, and that otherwise $\mathscr{A}^*(P)$ enters a "loop stop" in this situation (i.e., diverges without moving any tape). Let $g(q, \underline{x})$ be the following function to states of $\mathscr{A}(P)$:

$$g(q, \underline{x}) = \begin{cases} \underline{a}, & \text{if } q' = \underline{a}, \\ \underline{r}, & \text{if } q' = \underline{r}, \\ q_{q', \underline{x}}, & \text{if } q' \text{ is unstarred}, \\ q_{f_{\underline{x}}(r_1), \underline{x}}, & \text{otherwise}, \end{cases}$$

where $q' = f_{\underline{x}}{}^N(q)$ and r_1 is the state introduced in (i) above (note that $f_{\underline{x}}(r_1)$ is unstarred). Suppose now a typical unstarred rule of $\mathscr{A}^*(P)$ is

$$q^i, \xi \rightarrow p,$$

then for each $\underline{x} \in \Sigma^n$ the corresponding rule of $\mathscr{A}(P)$ is

$$q^i_{q, \underline{x}}, \xi \rightarrow g(p, \underline{y})$$

where $\underline{y}$ results from $\underline{x}$ by changing $\underline{x}(i)$ to ξ. This completes the construction of $\mathscr{A}(P)$. Note that on any n-tape the sequence of states followed by $\mathscr{A}(P)$ correspond to the sequence of unstarred states followed by $\mathscr{A}^*(P)$ neglecting $\underline{x}$ components of the state of $\mathscr{A}(P)$, with the possible exception of some n-tapes which $\mathscr{A}(P)$ rejects and on which $\mathscr{A}^*(P)$ diverges. Therefore $\underline{a}$ is reached by $\mathscr{A}(P)$ if and only if $\underline{a}$ is reached by $\mathscr{A}^*(P)$ on the same n-tape, so that both automata accept the same tapes. This completes the proof of 6.3.

Since $\mathscr{A}(P) \in \mathscr{RS}$, $\mathscr{A}(P) \equiv \mathscr{A}(Q)$ if and only if $\mathscr{I}_a(\mathscr{A}(P)) = \mathscr{I}_a(\mathscr{A}(Q))$; an immediate corollary of 6.3 is therefore:

LEMMA 6.4. *$P \equiv Q$ if and only if $\mathscr{A}(P) \equiv \mathscr{A}(Q)$.*

If P contains r transfers the construction can be modified by requiring the words $w(I, E)$ to contain binary sequences of length r after each operator symbol; the j-th member of such a sequence on tape i is interpreted as giving the "new" value of the j-th transfer on L_i.

We may speak rather loosely of two computational models being *equivalent* if the problems of equivalence within the two models are interchangeable, and if given an element in one model there is an algorithm for producing an element in the other model which "simulates" it in some suitable sense. The discussion above may then be summarized by,

THEOREM 6.5. *The n-tape Rabin–Scott automata are equivalent to the independent location schemas with n locations.*

COROLLARY. *Ianov schemas are equivalent to the monadic schemas with* one *location.*

The corollary may be proved directly by constructions similar to those above, or by appeal to the equivalence of Ianov schemas with finite automata, Rutledge [8].

At the time of writing it is not known whether the strong equivalence problem for independent location schemas with n locations ($n > 1$) is solvable; the corresponding problem for n-tape automata is also open!

Below we present a brief account of some classes of schemas for which decision procedures for strong equivalence are known, and some for which the problem remains open. Details of proofs are omitted and can be found in Paterson [5].

THEOREM 6.6. *The equivalence problem for monadic schemas with nonintersecting loops is decidable.*

We have not yet been able to eliminate the restriction to monadic operators, but we believe that the result would still hold. The decision procedure of the theorem expresses the equivalence of two given schemas as a formula of the additive theory of the natural numbers, a decidable theory.

A characteristic feature of the schemas with unsolvable decision problems that we have been considering, is that most expressions which are calculated are recalculated later on, a feature which would be unusual and undesirable in an actual computer program. We therefore look at various restrictions which can be imposed on schemas to prevent this repetitive behaviour.

DEFINITION. A schema P is *free* if any sequence through P is an execution sequence.

DEFINITION. A schema P is *liberal* if, in any sequence through P, no expression is computed more than once.

THEOREM 6.7. *Given any liberal schema there is an effective construction of an equivalent free schema.*

It seems likely that the decision problem for the equivalence of free schemas is solvable, and certainly none of the techniques we have used so far to prove unsolvability is obviously applicable. Theorem 6.7 shows that, for practical purposes, any liberal schema can be considered to be free, but the reverse is not the case, as is demonstrated by the free schema:

$$
\begin{array}{ll}
 & L_2 := F(L_1), \\
\text{a.} & L_1 := F(L_1), \\
 & L_1 := F(L_1), \\
 & L_2 := F(L_2), \\
 & T(L_1)\ b,\ a. \\
\text{b.} & \text{STOP.}
\end{array}
$$

The difference between the two classes is apparent too in the next theorems.

THEOREM 6.8. *Freedom is not a decidable property.*

THEOREM 6.9. *Liberality is a decidable property.*

The proof of Theorem 6.8 involves a reduction from the Post correspondence problem. For Theorem 6.9 we can show that there can be an effective and exhaustive search for the *first* occurrence of illiberality.

To date, we have a decision procedure for only a subclass of liberal schemas.

DEFINITION. A schema is *progressive* if, in any sequence through it, the assignment location of each computation instruction is taken as a retrieval location by the next computation instruction, if any.

THEOREM 6.10. *The equivalence problem for progressive schemas is decidable.*

The decision procedure is similar to that used in deciding equivalence for finite automata, but also involves the symmetric group of all permutations of the schema locations.

REFERENCES

1. M. DAVIS, "Computability and Unsolvability," McGraw–Hill Book Co., New York, 1958.
2. IU. I. IANOV, The logical schemes of algorithms, *Problems of Cybernetics (USSR)* 1(1960), 82–140.
3. D. LUCKHAM AND D. PARK, The undecidability of the equivalence problem for program schemata, Bolt, Beranek and Newman Inc., Report No. 1141, Santa Ana, Calif., 1964.
4. J. MCCARTHY, A basis for a mathematical theory of computation, *in* "Computer Programming and Formal Systems," (P. Braffort and D. Hirschberg, Eds.), pp. 33–70, North–Holland, Amsterdam, 1963.
5. M. PATERSON, "Equivalence Problems in a Model of Computation," Doctoral Dissertation, Cambridge University, 1967.
6. M. RABIN AND D. SCOTT, Finite automata and their decision problems, *IBM J. Res. Develop.* 3 (1959), 114–125.
7. A. ROSENBERG, On multi-head finite automata, Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design, pp. 221–228, 1963.
8. J. RUTLEDGE, On Ianov's program schemata, *J. Assoc. Comput. Mach.* 11 (1964), 1–9.