JOURNAL OF COMPUTER AND SYSTEM SCIENCES 8, 387–408 (1974)

Translatability of Schemas over Restricted Interpretations*

A. J. KFOURY

Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

Received February 8, 1973

We study a notion of translatability between classes of schemas when restrictions are placed on interpretations. Unlike the usual notion of translatability, our translatability lets the translation depend on the interpretation.

We consider five classes of schemas. Three of these have been extensively studied in the literature: flow-chart, flow-chart with counters, and recursive. The two others are defined herein; the first of which is a class of "maximal power" and is equivalent to similarly motivated classes of other investigators; while the second is, in some sense, a nontrivial class of "minimal power."

Our main results specify restrictions on interpretations that will allow the translatability of a class $\mathscr{S}_1$ into a class $\mathscr{S}_2$, $\mathscr{S}_1$ not being translatable into $\mathscr{S}_2$ over all (or unrestricted) interpretations. Additional results specify restrictions on interpretations under which $\mathscr{S}_1$ is not translatable into $\mathscr{S}_2$; the proofs of these clarify the mechanisms in the main results. Last, we consider the notion of effective computability in algebraic structures in the light of the main results.

1. INTRODUCTION

Schemas are abstract representations of computer programs. Given a set of variables (denoted by $x_0, x_1, x_2, \ldots$), a set of basic operations (denoted by $f_0, \ldots, f_m$) which may be used to assign new values to the variables, and a set of basic relations (denoted by $r_0, \ldots, r_l$), a schema specifies the order in which the computations involving the basic operations are to be performed according to truth values taken on by the basic relations. We shall call the finite sequence $\langle r_0, \ldots, r_l ; f_0, \ldots, f_m \rangle$ the *similarity type* of the schema. The arity of a basic operation will be $\geqslant 0$, while that of a basic relation will be $\geqslant 1$. A 0-ary operation is also called a constant.

Different classes of schemas have different expressive powers. Among the several ones that have been defined and studied in the literature, the two most familiar are the class of *flow-chart schemas* and the class of *recursive schemas*, which we shall respectively denote by $\mathscr{S}_{\text{fc}}$ and $\mathscr{S}_{\text{rec}}$. We shall need other classes of schemas in this

* Supported in part by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number N0014-70-A-0362-0006 and National Science Foundation Contract Number GJ004327.

paper, which will allow us to set up a hierarchy on all these classes relative to their expressive powers. One such class will represent, in some sense, an upper bound on the expressive power of programming languages; this will be the class of *effective schemas*, $\mathscr{S}_{\text{eff}}$, defined in Section 2. Another class will represent a lower bound on the expressive power of programming languages, but which must also possess a nontrivial computing power; this will be the class of *quasi-loop-free schemas*, $\mathscr{S}_{\text{qlf}}$. We shall consider also the class of *flow-chart schemas with counters*, $\mathscr{S}_{\text{fcc}}$, which are representative for the kinds of schemas widely studied in the literature.

Schemas compute relative to interpretations which fix the meanings of the basic operations and relations. Formally, a (total) *interpretation* $\mathfrak{A}$ for a schema S provides:

(i) a domain A,

(ii) for each k-ary basic relation symbol r in S, a total relation

$$r^A : A^k \to \{\text{true, false}\},$$

(iii) for each k-ary basic operation symbol f in S, a total operation

$$f^A : A^k \to A.$$

In the literature, it has been customary, in contrast to our definition, to have interpretations also fix the meanings of the variables. By leaving the assignment of the variables unspecified, as it is for us, an interpretation is none other than a (relational or algebraic) *structure*. A schema S under interpretation $\mathfrak{A}$, as given above, defines a *partial function* which we shall denote by $S^{\mathfrak{A}}$; for every assignment from the domain A to the variables of $S^{\mathfrak{A}}$, there corresponds a unique execution path (or computation) in $S^{\mathfrak{A}}$, which may either converge or diverge. For two partial functions U and V, we write $U \simeq V$ if, for all x, either both $U(x)$ and $V(x)$ are undefined or both are defined and $U(x) = V(x)$.

To compare the expressive powers of different classes of schemas, we have the concept of translatability (see for example [2, 5, 12, 16]): a class $\mathscr{S}_1$ of schemas is *translatable* into a class $\mathscr{S}_2$ of schemas, in symbols $\mathscr{S}_1 \subset \mathscr{S}_2$, if

$$(\forall S_1 \in \mathscr{S}_1)(\exists S_2 \in \mathscr{S}_2)(\forall \text{ interpretation } \mathfrak{A})[S_1^{\mathfrak{A}} \simeq S_2^{\mathfrak{A}}]. \qquad [*]$$

For example, the main result relating flow-chart to recursive schemas says that $\mathscr{S}_{\text{fc}} \subset \mathscr{S}_{\text{rec}}$ but $\mathscr{S}_{\text{rec}} \not\subset \mathscr{S}_{\text{fc}}$ [12, 16].

In this paper, we consider a notion of translatability relative to a subfamily of the family of all interpretations. That is, in the previous definition [*], we consider restricting the third quantifier to interpretations which satisfy certain *structural conditions* (to distinguish them from other kinds of conditions). If Γ is a set of structural conditions, we shall say that class $\mathscr{S}_1$ is *Γ-translatable* into class $\mathscr{S}_2$, in symbols $\mathscr{S}_1 \to_\Gamma \mathscr{S}_2$, if

$$(\forall \text{ interpretation } \mathfrak{A} \text{ satisfying } \Gamma)(\forall S_1 \in \mathscr{S}_1)(\exists S_2 \in \mathscr{S}_2)[S_1^{\mathfrak{A}} \simeq S_2^{\mathfrak{A}}]. \qquad [**]$$

Note that, in contrast to [*], we have put the quantification over interpretations in first position; so that, unlike the usual notion of translatability, Γ-translatability lets the translation depend on the interpretation. As a consequence, Γ-translatability does not reduce to the usual notion of translatability when Γ is the empty set; but, if $\mathscr{S}_1$ is translatable into $\mathscr{S}_2$ then, for any Γ, $\mathscr{S}_1$ is Γ-translatable into $\mathscr{S}_2$. (Of course, we could also keep the quantification over interpretations in third position in [**], which would give rise to a stronger notion of translatability over restricted interpretations. But this notion will not concern us here.)

The study of schemas over restricted interpretations has already been undertaken in the literature, although usually not in relation to the problem of translatability. Thus, in [11], Paterson considers equivalence of schemas over finite and recursive interpretations; and, in [3], Chandra and Manna consider decidability questions when interpretations have equality among their underlying relations. Sometimes, known results that are motivated by considerations different from ours can be restated to fit the setting of this paper; for example, Theorem 3.13 in [5] implies that, if Γ restricts operations of interpretations to be monadic, $\mathscr{S}_{\text{rec}}$ is Γ-translatable into $\mathscr{S}_{\text{fcc}}$.

Given two classes of schemas $\mathscr{S}_1$ and $\mathscr{S}_2$ (among those mentioned earlier in this introduction) such that $\mathscr{S}_1$ is not translatable into $\mathscr{S}_2$, the main results of this paper say how to restrict interpretations (via some Γ) so that $\mathscr{S}_1$ becomes Γ-translatable into $\mathscr{S}_2$. This is done in Section 3, where we define four structural conditions precisely, then state two theorems accordingly. Several easy consequences of these theorems are left to the reader's care; for example, on the basis of Theorem II, one can choose Γ so that $\mathscr{S}_{\text{rec}} \to_\Gamma \mathscr{S}_{\text{fcc}}$ (even without Γ restricting interpretations to have monadic operations only).

In Section 4, we give conditions Γ under which it is not possible to Γ-translate a class $\mathscr{S}_1$ into a class $\mathscr{S}_2$ (again, $\mathscr{S}_1$ and $\mathscr{S}_2$ are drawn from the classes mentioned earlier). The proofs of this section are constructions that clarify the mechanisms in the results of Section 3.

In Section 5, we relate the results of Section 3 to the notion of effective computability in algebraic structures. In particular, we specify the control structure required by a programming language to program any effective function over distributive lattices, of the rings, and fields.

All the proofs of our results, along with the required lemmas, are in the Appendix of the paper.

The author would like to express his gratitude to the referee who pointed out that Theorem II in this paper is a rewording of Theorem 1.6 in Ref. [4], and the proof is essentially the same in both. He also pointed out that the reformulation of condition (δ) at the end of Section 3 corresponds to Definition 1.23 of [4].

2. Effective Schemas

Our objective in this section is to give a tentative definition of a class of schemas which has as much expressive power as, or more than, any other class.

Given any flow-chart schema, we can "unwind" its loops to produce an equivalent countable binary tree. In fact, given any schema in one of the classes existing in the literature, we can write an equivalent countable binary tree. (To be correct, the binary tree describes only the "control structure" of the equivalent schema; to complete this equivalent schema, we also need to specify an instruction for each of the tree's nodes.) In the countable tree equivalent to a given schema, we can always enumerate effectively the set of finite paths (for the moment, we consider all finite paths, whether or not they are consistent). For a flow-chart schema, this set is regular; in a sense which can be made precise. (For this, observe that a flow-chart can be looked at as a finite automaton.) For a recursive schema, this set is not always regular; it is generally context-free. For other schemas, with more expressive power, this set becomes more complex; but it can always be enumerated effectively.

The above observations suggest our definition of the class of *effective schemas*, a candidate class for "universality," i.e., one with universal expressive power. First, we need some preliminary definitions.

An (uninterpreted) *instruction* is one of the following two:

(i) a *test* instruction, which is an expression of the form

$$r(x_{j_1}, ..., x_{j_k}),$$

where r is a k-ary relation symbol, $k \geqslant 1$.

(ii) an *assignment* instruction, which is an expression of the form

$$x_{j_1} := x_{j_2},$$

or

$$x_j := f(x_{j_1}, ..., x_{j_k}),$$

where f is a k-ary operation symbol, $k \geqslant 0$.

A *schema* is a countable binary tree, "growing downwards," where:

(1) the root node has one successor and is labeled by finitely many variables, called *input variables*;

(2) a node which is not the root and has one successor is labeled by an assignment instruction;

(3) a node which is not the root and has two successors is labeled by a test instruction; the two outgoing branches from a test node are labeled by $+$ and $-$;

(4) a terminal node, one with no successor, is labeled either by YES or by NO;

(5) along any path, a variable which appears in a test instruction or on the right of ":=" in an assignment instruction must have been mentioned at a preceding node on the left of ":=" in an assignment instruction, or else at the root node (and thus it is an input variable);

(6) the number of distinct symbols (other than variables) appearing in all the instructions of a schema is finite, i.e. the similarity type of a schema is finite; however, the number of distinct variables, which includes the input variables, is not necessarily finite.

The last point, (6), is compatible with our experience in computer programming, where any program is written in terms of finitely many relations and operations. This is not crucial. Under certain restrictions, we can extend the results we have to the case when the similarity type of a schema is countably infinite.

A schema as defined above may be called a *p-schema* ("*p*" for "predicate"). To define a *f-schema* ("*f*" for "function"), point (4) must be changed to (4′):

(4′) A terminal node, one with no successor, is labeled by an output variable, which is one of the variables appearing on the left of ":=" in one of the nodes preceding that terminal node or else must be one of the input variables.

In most of our results, we shall not need to make the distinction between the two kinds of schemas. We make the following convention. A statement where "schema" is mentioned without a prefix, *p*— or *f*—, is one which is true of all schemas of both kinds. Observe that the study of *f*-schemas is not a trivial variation on that of *p*-schemas, if we are not allowed to test equality between the assignments of two variables; i.e., if the equality relation is not in the similarity type, then the characteristic relation of a function definable by a *f*-schema need not be definable by a *p*-schema.

DEFINITION. An *effective schema* is a schema which can be effectively written down. A formal definition requires the arithmetization of the syntax: if the nodes of a schema are numbered with the nonnegative integers, from top donwnwards and from left to right (the root is at the top), then the node labels of an effective schema are generated as the recursively enumerable range of a total recursive function.

It is clear that the set of finite paths in an effective schema is recursively enumerable. The important point here is to observe that, under this paper's restriction to total interpretations, the other schemas in the literature can be expressed as effective schemas by unwinding loops.

Other investigators have also considered increasing the expressive power of programming languages up to a universal level. They have reached equivalent definitions via different motivations and different arguments. In [15], Strong has defined a programming language of "maximum power"; a simple proof shows that his definition is equivalent to ours (that of effective schemas). This is also the case for the definition of "generalized Turing algorithms" by Friedman in [4]. Finally, in the lattice of flow-diagrams defined by Scott in [13], what we might call the "computable elements" (i.e., flow-diagrams defined as limits of recursively enumerable sequences of finite approximations) are precisely the effective schemas. These facts give further evidence to the claim that the class of effective schemas has maximal power.

Chandra and Manna [3] state that "It turns out that the class of program schemas with equality is more powerful than the maximal classes suggested by other investigators." This seems to conflict with our claim as to the universality of effective schemas; and, indeed, there are effective procedures which are definable by flow-chart schemas with equality but *not* by effective schemas *without* equality. However, the differences between effective procedures (i.e., interpreted schemas) that schemas aim to reflect are differences at the level of the *control structure*, but not at the level of the initial computing capability (i.e., the basic relations and operations). We object therefore to Chandra and Manna's statement, because a proper comparison between classes of schemas can be made only if they have the same initial computing capability (the same similarity type with the same restrictions on it, if any). Thus, flow-charts with equality are incomparable with, not more powerful than, recursive schemas without equality.

We conclude this section by making a few remarks on notation and terminology. Schemas will be denoted by capital S's (subscripted, if need be). Interpretations (i.e., algebraic structures) will be denoted by German script letters, $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$,..., and their domains by the corresponding Latin letters, A, B, C,... . If a schema S has k input variables $x_1 ,..., x_k$, we shall sometimes exhibit them explicitly by writing: $S(x_1 ,..., x_k)$. Let the similarity type of S be $\langle r_0 ,..., r_l ; f_0 ,..., f_m \rangle$ and

$$\mathfrak{A} = \langle A; r_0^A, r_1^A, ...; f_0^A, f_1^A, ... \rangle$$

be an interpretation for S. To display the fact that the symbols $r_0 ,..., f_m$ in schema S have been interpreted as the relations and operations $r_0^A, ..., f_m^A$ on the domain A, we shall write: $S^{\mathfrak{A}}(x_1 ,..., x_k)$. By assigning input values $a_1 ,..., a_k \in A$ to $x_1 ,..., x_k$, a *computation* on S is thus specified, which will be denoted by $S^{\mathfrak{A}}(a_1 ,..., a_k)$.

$S^{\mathfrak{A}}(x_1 ,..., x_k)$ without input values from A assigned to $x_1 ,..., x_k$ (and we shall refer to this situation by saying: *S interpreted in the structure* $\mathfrak{A}$) can be viewed as a procedure over the domain A, written in terms of the underlying relations and operations of the algebraic structure $\mathfrak{A}$. If S is a $\cdots$ p-schema (f-schema), we shall call $S^{\mathfrak{A}}$ a $\cdots$ *p-procedure* (*f-procedure*) *of* $\mathfrak{A}$, where the three dots may be replaced by any of the following: flow-chart, flow-chart with counters, recursive, etc. (Whenever a schema S is interpreted in a structure $\mathfrak{A}$, it is assumed that the similarity type of S is contained in that of $\mathfrak{A}$.)

In one given context, all schemas will be in the same similarity type. For example, if we speak of the class of all flow-chart schemas, we only mean those with a similarity type equal to, or contained in, some fixed similarity type made clear by the context.

3. The Main Problem and Some Solutions to It

We consider the following classes of schemas

$$\begin{aligned}
\mathscr{S}_{\mathrm{eff}} &= \{\text{effective schemas}\},\\
\mathscr{S}_{\mathrm{rec}} &= \{\text{recursive schemas}\},\\
\mathscr{S}_{\mathrm{fcc}} &= \{\text{flow-chart schemas with counters}\},\\
\mathscr{S}_{\mathrm{fc}} &= \{\text{flow-chart schemas}\},\\
\mathscr{S}_{\mathrm{qlf}} &= \{\text{quasi-loop-free schemas}\}.
\end{aligned}$$

$\mathscr{S}_{\mathrm{rec}}$, $\mathscr{S}_{\mathrm{fcc}}$, and $\mathscr{S}_{\mathrm{fc}}$ have been defined elsewhere (see for example [12, 16]). A *quasi-loop-free schema* is a flow-chart schema where, from any test instruction t, the two outgoing branches do not go back to instructions preceding t (although they may go back to t itself, as self-loops). A quasi-loop-free schema may thus be called "a truth-table schema which may diverge," as opposed to a *loop-free schema* which may be viewed as "a truth-table schema which always converges."

Note that in a flow-chart schema S with counters, because there is no "communication" between the counters and the other variables of S, the arithmetical operations and relations which are applied to the assignments of counters do *not* appear in the similarity type of S. The point is that counters increase the power of the "control structure" of S, but not the power of the "initial computing capability" present in S (the basic relations and operations in S).

Let Σ denote the collection of all the forementioned classes,

$$\Sigma = \{\mathscr{S}_{\mathrm{qlf}}, \mathscr{S}_{\mathrm{fc}}, \mathscr{S}_{\mathrm{fcc}}, \mathscr{S}_{\mathrm{rec}}, \mathscr{S}_{\mathrm{eff}}\}.$$

Between these classes, we have the following translatabilities

$$\begin{tikzcd}
 & & \mathscr{S}_{\mathrm{rec}} \arrow[dr, phantom, sloped, "\subset"] & \\
\mathscr{S}_{\mathrm{qlf}} \arrow[r, phantom, "\subset"] & \mathscr{S}_{\mathrm{fc}} \arrow[ur, phantom, sloped, "\subset"] \arrow[dr, phantom, sloped, "\subset"] & & \mathscr{S}_{\mathrm{eff}} \\
 & & \mathscr{S}_{\mathrm{fcc}} \arrow[ur, phantom, sloped, "\subset"] &
\end{tikzcd}$$

and no other (unrestricted) translatability is possible ($\mathscr{S}_{\mathrm{fc}} \not\subset \mathscr{S}_{\mathrm{qlf}}$ is a consequence of Proposition III in Section 4, the other negated translatabilities follow from the work in [5, 12, 16], or can also be easily deduced from our other results in Section 4).

PROBLEM. *Let $\mathscr{S}_1$ and $\mathscr{S}_2 \in \Sigma$, such that $\mathscr{S}_1 \not\subset \mathscr{S}_2$ (otherwise the problem is trivial). Determine a set Γ of structural conditions such that $\mathscr{S}_1$ is Γ-translatable into $\mathscr{S}_2$.*

In determining Γ, we try to make it "as weak as possible"; for interpretations can be so restricted as to make $\mathscr{S}_1 \to_\Gamma \mathscr{S}_2$ always true, independently of $\mathscr{S}_1$ and $\mathscr{S}_2$ (for example, let Γ restrict interpretations to be finite). Further, since Γ-translatability between classes of schemas is a transitive relation, several answers to the above problem, that are not stated here, easily follow from the results given below.

Corresponding to the collection Σ, the solution to the problem stated above will depend on four structural conditions (called α, β, γ, and δ), which are stated in terms of the concepts defined next.

Let $\{f_0, ..., f_m\}$ be a finite set of operation symbols (all appearing in some fixed similarity type). We define the $(x_{i_1}, ..., x_{i_k})$-*terms*, for arbitrary nonnegative integers $i_1, ..., i_k$, inductively as follows (when there is no fear of confusion, we shall also call them $\mathbf{x}$-*terms*, abbreviating $(x_{i_1}, ..., x_{i_k})$ by $\mathbf{x}$):

(i) $x_{i_1}, ..., x_{i_k}$ are all $\mathbf{x}$-terms;

(ii) if $f \in \{f_0, ..., f_m\}$ is a j-ary operation symbol, $j \geqslant 0$, and $t_1, ..., t_j$ are $\mathbf{x}$-terms, then $f(t_1, ..., t_j)$ is a $\mathbf{x}$-term.

Given a $(x_{i_1}, ..., x_{i_k})$-term t, we shall sometimes exhibit explicitly the variables by writing: $t(x_{i_1}, ..., x_{i_k})$; and, if the similarity type of structure $\mathfrak{A}$ contains $\{f_0, ..., f_m\}$, the interpretation of t in $\mathfrak{A}$, denoted by t^A or $t^A(x_{i_1}, ..., x_{i_k})$, is defined in the obvious sense ($f_0, ..., f_m$ are interpreted as the underlying operations $f_0^A, ..., f_m^A$ of $\mathfrak{A}$, and t^A becomes "a polynomial on the domain A in the free variables $x_{i_1}, ..., x_{i_k}$").

Let $\mathfrak{A}$ be the structure $\langle A; r_0^A, ..., r_l^A; f_0^A, ..., f_m^A \rangle$. We say that an element $b \in A$ is *reachable* from elements $a_1, ..., a_k \in A$ if there is a $(x_1, ..., x_k)$-term $t(x_1, ..., x_k)$ such that $b = t^A(a_1, ..., a_k)$.

We say that structure $\mathfrak{A}$ is *n-generated* if there are n elements from the domain A, $a_1, ..., a_n \in A$, such that every $b \in A$ is reachable from $a_1, ..., a_n$.

The structural conditions satisfiable by a structure (or interpretation) $\mathfrak{A}$ which we shall consider are:

(α) For every integer $n \geqslant 1$, there is a finite cardinal w such that any n-generated substructure of $\mathcal{Q}$ is of cardinality $\leqslant w$. (A *substructure* $\mathfrak{B}$ of $\mathfrak{A}$ is determined by a subset B of A which is closed under the underlying operations of $\mathfrak{A}$.)

(β) One of the underlying operations of $\mathfrak{A}$, say f_0^A, is a constant. In addition, there is a 1-ary function F on the domain A, not necessarily total (i.e., F may be undefined for some $a \in A$), which is definable by a flow-chart f-procedure of $\mathfrak{A}$ and which generates from f_0^A the following chain

$$\overset{f_0^A}{\bullet} \longrightarrow \overset{F(f_0^A)}{\bullet} \longrightarrow \overset{F^2(f_0^A)}{\bullet} \longrightarrow \cdots \longrightarrow \overset{F^i(f_0^A)}{\bullet} \longrightarrow \cdots, \quad i \geqslant 1.$$

(γ) The equality relation on the entire domain A is definable by a total flow-chart p-procedure of $\mathfrak{A}$.

(δ) ($\forall$ integer $k \geqslant 1$) ($\exists$ integer $n \geqslant k$) ($\forall\, a_1 ,..., a_k , b \in A$) [$b$ is reachable from $a_1 ,..., a_k \Rightarrow$ there is a flow-chart f-procedure of $\mathfrak{A}$, with k input variables and using at most n variables, which outputs value b when given input values $a_1 ,..., a_k$].

We shall have use for the negations of the above conditions, the negation being appended to the name of a condition in its abbreviated form "$\neg$"; for example, the expression $\neg\gamma$ refers to: the equality relation on the entire domain A is *not* definable by a total flow-chart p-procedure of $\mathfrak{A}$.

THEOREM I. *Let $\mathcal{S}$ be any class of schemas in the collection Σ. $\mathcal{S}$ is $\{\alpha\}$-translatable into $\mathcal{S}_{\text{qlf}}$.*

THEOREM II. *$\mathcal{S}_{\text{eff}}$ is $\{\beta, \gamma, \delta\}$-translatable into $\mathcal{S}_{\text{fc}}$.*

There are other solutions to the Main Problem. An easy variation of Theorem II gives: $\mathcal{S}_{\text{eff}}$ is $\{\gamma, \delta\}$-translatable into $\mathcal{S}_{\text{fcc}}$ (verification is left to the reader). The universality of recursive schemas with counters, proved in [2 and 15], gives: $\mathcal{S}_{\text{eff}}$ is $\{\beta, \gamma\}$-translatable into $\mathcal{S}_{\text{rec}}$ (we can in fact weaken β and γ by substituting "recursive" for every occurrence of "flow-chart" and by not requiring that the p-procedure in condition γ be defined on elements of A outside the ω-chain described in condition β). Finally, one can readily see that $\mathcal{S}_{\text{fcc}}$ is $\{\beta, \gamma\}$-translatable into $\mathcal{S}_{\text{fc}}$ (here, also, we can weaken condition γ by not requiring that the p-procedure be defined on elements not in the ω-chain described in condition β).

We end this section by giving an alternative formulation for structural condition δ, which will be useful in the applications. We first need to define the *n-$\mathbf{x}$-terms*. To motivate this definition, we refer the reader to the proof of Theorem 1 in [12], where it is shown that a flow-chart schema with input variables $x_{i_1} ,.., x_{i_k}$ cannot "compute" all the $(x_{i_1} ,..., x_{i_k})$-terms (following the terminology of [16], we shall say that the uninterpreted $\mathbf{x}$-terms are *syntactic values*; this allows us to say that what schemas compute are syntactic values, as opposed to *semantic values*, or interpreted syntactic values, which are computed by procedures, or interpreted schemas). For arbitrary $x_{i_1} ,..., x_{i_k}$ and $n \geqslant k$, consider the following finite sequence: $\mathbf{y}_0 ,..., \mathbf{y}_u$, where

(i) each $\mathbf{y}_v$, $0 \leqslant v \leqslant u$, is an ordered sequence of length n;

(ii) $\mathbf{y}_0$ is $(x_{i_1} ,..., x_{i_k} , \underbrace{x_{i_1} ,..., x_{i_1}}_{n-k})$;

(iii) $\mathbf{y}_{v+1}$ is identical to $\mathbf{y}_v$, $0 \leqslant v < u$, except for one entry which is either $f(t_1 ,..., t_j)$, where $t_1 ,..., t_j$ are entries in $\mathbf{y}_v$ and f is a j-ary operation symbol in the similarity type under consideration, or else is some other entry of $\mathbf{y}_v$.

We define a n-$(x_{i_1}, ..., x_{i_k})$-term to be one of the entries in $\mathbf{y}_u$, for some sequence $\mathbf{y}_0, ..., \mathbf{y}_u$ as described above. For every n, one can verify that the n-$\mathbf{x}$-terms form a proper subset of the $\mathbf{x}$-terms; the n-$\mathbf{x}$-terms are exactly the $\mathbf{x}$-terms which are computed by flow-chart schemas with input variables $\mathbf{x}$ and using at most n variables. We are now in a position to reformulate structural condition δ:

(δ) For every $k \geqslant 1$, there is a $n \geqslant k$ such that, given any $\mathbf{x}$-term $t(x_1, ..., x_k)$ and any elements $a_1, ..., a_k$ from the domain A, there is a n-$\mathbf{x}$-term $\bar{t}(x_1, ..., x_k)$ such that:

$$t^A(a_1, ..., a_k) = \bar{t}^A(a_1, ..., a_k).$$

4. Additional Results

The results in this section are of the following form: Given two classes $\mathscr{S}_1$ and $\mathscr{S}_2$ in the collection Σ, we determine a set Γ of structural conditions for which $\mathscr{S}_1$ is *not* Γ-translatable into $\mathscr{S}_2$; that is, we determine a set Γ for which:

$$(\exists \text{ interpretation } \mathfrak{A} \text{ satisfying } \Gamma)\ (\exists S_1 \in \mathscr{S}_1)\ (\forall S_2 \in \mathscr{S}_2)\ [S_1{}^{\mathfrak{A}} \not\simeq S_2{}^{\mathfrak{A}}].$$

The proofs are constructions that clarify the mechanisms in the results of the previous section.

Proposition III. *Let $\mathscr{S}$ be a class of schemas in Σ and $\mathscr{S} \neq \mathscr{S}_{\mathrm{qlf}}$. $\mathscr{S}$ is not $\{\neg\alpha)$-translatable into $\mathscr{S}_{\mathrm{qlf}}$.*

Proposition IV. *$\mathscr{S}_{\mathrm{fcc}}$ is not $\{\neg\alpha\}$-translatable into $\mathscr{S}_{\mathrm{fc}}$.*

Proposition V. *$\mathscr{S}_{\mathrm{eff}}$ is not $\{\beta, \neg\gamma, \delta\}$-translatable into $\mathscr{S}_{\mathrm{fc}}$.*

There are other results that, being easy variations of the last two propositions, will not be proved here: (1) $\mathscr{S}_{\mathrm{fcc}}$ is not $\{\neg\alpha\}$-translatable into $\mathscr{S}_{\mathrm{rec}}$ (the technique of Lemma 3 in the Appendix still works when applied to a recursive, rather than flow-chart, schema (to see this, view a recursive schema as a flow-chart schema with a pushdown store) and the proof of Proposition IV, based on Lemma 3, can be used again with hardly any changes); (2) $\mathscr{S}_{\mathrm{eff}}$ is not $\{\beta, \neg\gamma, \delta\}$-translatable into $\mathscr{S}_{\mathrm{fcc}}$ (the proof of Proposition *V* still works when we allow flow-charts to have counters); (3) $\mathscr{S}_{\mathrm{rec}}$ is not $\{\beta, \neg\gamma, \delta\}$-translatable into $\mathscr{S}_{\mathrm{fcc}}$ and $\mathscr{S}_{\mathrm{fc}}$ (replace the effective schema defined in the proof of Proposition V by the recursive schema used in the second proof of Theorem 1 of [12]).

6. Effective Computability in Algebraic Structures

Logicians and computer scientists have recently studied the problem of what operations in an arbitrary algebraic structure can reasonably be considered "effective" in terms of the underlying operations and relations of the structure. Several approaches to this problem have been proposed; most notable among them are Kreisel's [8], Moschovakis' [9], and Abramson's [1]. Our contention is that schematology, via its various definitions of classes of schemas with universal expressive power, provides another approach. We propose the following thesis (also stated in [15]).

Thesis. *Given an algebraic structure (interpretation)* $\mathfrak{A}$, *a procedure on the domain* A *of* $\mathfrak{A}$, *effective relative to the underlying relations and operations of* $\mathfrak{A}$, *is an effective schema interpreted in* $\mathfrak{A}$.

We have investigated elsewhere some aspects of the notion of effectiveness in arbitrary algebraic structures (see [7]). One important aspect, which still remains unexplored, is to see how our schematology approach compares with the other forementioned approaches.

In this section we consider one specific problem, which we show can be handled with the theoretical tools developed earlier. We look at a given algebraic structure $\mathfrak{A}$ and attempt to find a class of schemas which will capture the notion of effectiveness in $\mathfrak{A}$. According to our Thesis, in any algebraic structure, the class of all effective schemas is adequate for this purpose. This is however a radical answer to the problem, which is indifferent to the "computing capability already present in $\mathfrak{A}$" (we do not make more precise this intuitive expression). For example, we would like to answer questions of the following nature: Is any effective procedure of $\mathfrak{A}$ equivalent to some flow-chart procedure of $\mathfrak{A}$? or to some recursive procedure of $\mathfrak{A}$? Let $\mathscr{S}$ be a class of schemas. We shall say that $\mathscr{S}$ *is sufficient in algebraic structure* $\mathfrak{A}$ if:

$$(\forall S_1 \in \mathscr{S}_{\mathrm{eff}})(\exists S_2 \in \mathscr{S})[S_1^{\mathfrak{A}} \simeq S_2^{\mathfrak{A}}];$$

that is, the expressive power of $\mathscr{S}$ is all we need to program any effective procedure on the domain A relative to the underlying operations and relations of $\mathfrak{A}$. The problem is then to find the "weakest" class of schemas which is sufficient in $\mathfrak{A}$. We can reverse our point of view and ask instead, for a given class of schemas $\mathscr{S}$, to characterize the algebraic structures $\mathfrak{A}$ where $\mathscr{S}$ is sufficient; i.e. to determine a set Γ of structural conditions such that

$$\mathfrak{A} \text{ satisfies } \Gamma \Leftrightarrow \mathscr{S} \text{ is sufficient in } \mathfrak{A}.$$

We now describe a few algebraic structures where $\mathscr{S}_{\mathrm{qlf}}$ or $\mathscr{S}_{\mathrm{fc}}$ suffices. (Observe that if Γ is a set of structural conditions satisfied by algebraic structure $\mathfrak{A}$ and $\mathscr{S}_{\mathrm{eff}}$ is Γ-translatable into class of schemas $\mathscr{S}$, then $\mathscr{S}$ is sufficient in $\mathfrak{A}$.)

A distributive lattice $\mathfrak{A}$ is an algebraic structure of the form

$$\mathfrak{A} = \langle A;\ \leqslant;\ \cap, \cup\rangle,$$

where $\leqslant$, $\cap$, and $\cup$ are given their usual meanings (all lattice-theoretic definitions can be found in [17]). An *expansion* $\mathfrak{B}$ of $\mathfrak{A}$ is of the form

$$\mathfrak{B} = \langle A;\ \leqslant;\ \cap, \cup, c_0, \dots, c_n\rangle,$$

where $c_0, \dots, c_n$ are arbitrary elements of A, considered as 0-ary operations on the domain A. (In a procedure of $\mathfrak{A}$, we can only use the relation $\leqslant$ and operations $\cap$, $\cup$; while in one of $\mathfrak{B}$, we can also use the constant operations $c_0, \dots, c_n$. In other words, we have a larger "initial computing capability" in $\mathfrak{B}$ than we have in $\mathfrak{A}$.)

COROLLARY VI. *$\mathscr{S}_{\mathrm{qlf}}$ is sufficient in a distributive lattice $\mathfrak{A}$, and in any expansion $\mathfrak{B}$ of $\mathfrak{A}$.*

An example of a distributive lattice is the following structure.

$$\mathfrak{N}^{+} = \langle N^{+};\ |;\ \mathrm{hcf}, \mathrm{lcm}\rangle,$$

where $N^{+} = \{1, 2, 3, \dots\}$, $x \mid y \Leftrightarrow x$ divides y, $\mathrm{hcf}(x, y) =$ highest common factor of x and y, and $\mathrm{lcm}(x, y) =$ least common multiple of x and y.

It is already well known that $\mathscr{S}_{\mathrm{fc}}$ suffices in the algebraic structure contemplated by recursive function theory, namely in $\langle N;\ =;\ \times, +, (\ )', 0\rangle$ (see [14]). $\mathscr{S}_{\mathrm{fc}}$ also suffices in other commonly encountered algebraic structures.

COROLLARY VII. *$\mathscr{S}_{\mathrm{fc}}$ is sufficient in rings and fields of characteristic zero which are respectively taken in the form*

$$\mathfrak{A} = \langle A;\ =;\ \times, +, -, 0, 1\rangle$$

and

$$\mathfrak{A} = \langle A;\ =;\ \times, +, -, (\ )^{-1}, 0, 1\rangle.$$

The usual way to translate recursive schemas (or, for that matter, any more powerful programming formalism) into flow-charts involves constructing flow-charts for a pairing function (a one-one function from A^2 to A) and its inverses in order to simulate a push-down store. According to Corollary VII, there are interpretations for which this is *not* necessary: It is sometimes possible to simulate recursive procedures by flow-charts without a coding and decoding capability (as it is, for example, over the field of algebraic numbers or over the real field).

7. Appendix

In this Appendix, we give the proofs of the results of the preceding sections.

Lemma 1. *Let $\mathfrak{A}$ be an arbitrary algebraic structure; and S_1, S_2 be arbitrary schemas with input variables drawn from $\{x_1, ..., x_k\}$ and in similarity types (possibly different) both contained in the similarity type of $\mathfrak{A}$. Then, $S_1^{\mathfrak{A}} \simeq S_2^{\mathfrak{A}} \Leftrightarrow$ for every k-generated substructure $\mathfrak{B}$ of $\mathfrak{A}$ we also have $S_1^{\mathfrak{B}} \simeq S_2^{\mathfrak{B}}$.*

Proof. The implication from left to right is immediate. To prove it from right to left, suppose that $S_1^{\mathfrak{A}} \not\simeq S_2^{\mathfrak{A}}$. There is therefore an input sequence $\mathbf{a} \in A^k$ from the domain of $\mathfrak{A}$ such that $S_1^{\mathfrak{A}}(\mathbf{a}) \neq S_2^{\mathfrak{A}}(\mathbf{a})$. If $\mathfrak{B}$ is the substructure of $\mathfrak{A}$ k-generated by the k components of $\mathbf{a}$, it is not difficult to verify that we also have $S_1^{\mathfrak{B}} \not\simeq S_2^{\mathfrak{B}}$.

Proof of Theorem I. It suffices to consider the case $\mathscr{S} = \mathscr{S}_{\text{eff}}$. We omit the easy proof of the following fact [*]: Given a family $\mathscr{K}$ of algebraic structures and an effective schema S, if all the interpretations for S from $\mathscr{K}$ use only finitely many terminating paths in S, then there exists a quasi-loop-free schema S_0 (with the same input variables as S) such that, for all interpretations $\mathfrak{A} \in \mathscr{K}$, $S^{\mathfrak{A}} \simeq S_0^{\mathfrak{A}}$. Let S be an arbitrary effective schema with n input variables. Let $\mathfrak{A}$ be an interpretation which satisfies structural condition α, and w be the finite cardinal which limits the size of all n-generated substructures of $\mathfrak{A}$. Notice that, given a finite similarity type (that of $\mathfrak{A}$ which contains that of S), there are finitely many nonisomorphic algebraic structures (say $\mathfrak{B}_1, ..., \mathfrak{B}_j$) which are of cardinality $\leqslant w$. Let $\mathscr{K}_i$ be the family of all interpretations which are isomorphic to $\mathfrak{B}_i$, $1 \leqslant i \leqslant j$. Since $\mathfrak{B}_i$ is finite, it is easy to see that all the interpretations from $\mathscr{K}_i$ for S can only use finitely many terminating paths in S. Hence, the same is also true of the family $\mathscr{K} = \mathscr{K}_1 \cup \cdots \cup \mathscr{K}_j$. By [*] above, there is a quasi-loop-free schema S_0 such that, for all $\mathfrak{B} \in \mathscr{K}$, $S^{\mathfrak{B}} \simeq S_0^{\mathfrak{B}}$. Noting that any n-generated substructure of $\mathfrak{A}$ belongs to $\mathscr{K}$, we also have that $S^{\mathfrak{A}} \simeq S_0^{\mathfrak{A}}$, by Lemma 1.

Lemma 2. (i) *Let $\mathfrak{A}$ be an interpretation which satisfies structural conditions β and γ, and identify the subset $\{F^i(f_0^A) | i \in N\}$ of its domain A with the set N of natural numbers. If φ is a partial recursive function (on N), there exists a partial function θ (on A), which is definable by a flow-chart f-procedure of $\mathfrak{A}$ such that $\varphi = \theta \upharpoonright N$.*

(ii) *A function on N is partial recursive if and only if it is definable by some flow-chart f-procedure of the algebraic structure*

$$\langle N; T; (\ )', '(\ ), 0\rangle,$$

where $T(x) \Leftrightarrow x$ is assigned value 0, and $(\ )'$ and $'(\ )$ are the successor and predecessor operations.

Proof. (i) This follows from the fact that we can define the partial recursive functions with flow-charts in the similarity type $\langle =;\ (\ )',\ 0\rangle$. Details are omitted.

(ii) The proof here is similar to that of (i). In fact, this is an old result established by Shepherdson and Sturgis [14].

Proof of Theorem II. We consider a particular case of an algebraic structure $\mathfrak{A}$ which satisfies structural conditions β, γ, and δ; namely, in

$$\mathfrak{A} = \langle A;\ r_0^A,\dots,\ r_l^A;\ f_0^A,\dots,\ f_m^A\rangle,$$

r_0^A is the equality relation $=$, and f_1^A is a 1-ary operation which generates from the constant f_0^A the chain drawn in the definition of structural condition β. It will be obvious how to generalize the proof to an arbitrary algebraic structure $\mathfrak{A}$ which satisfies the forementioned structural conditions. For convenience, we identify f_0^A with 0 and f_1^A with the successor operation $(\ )'$ when restricted to the set

$$N \simeq \{f_0^A, f_1^A(f_0^A), f_1^A f_1^A(f_0^A),\dots\} \subseteq A.$$

Without loss of generality, we restrict our attention to p-schemas. We show, given any effective p-schema S, how to construct (noneffectively in general, because of the noneffectiveness of structural condition δ) a flow-chart p-schema S_0 such that $S^{\mathfrak{A}} \simeq S_0^{\mathfrak{A}}$. Let $\mathbf{x} = (x_1,\dots, x_k)$ be the input variables of S. For every terminating path π through the tree corresponding to S, it is easy to write a finite conjunction $R(\mathbf{x})$ of atomic and negative atomic formulas, in the similarity type of S, such that:

$$S^{\mathfrak{A}} \text{ on input } \mathbf{a} \in A^k \text{ follows path } \pi \Leftrightarrow \mathfrak{A} \text{ satisfies } R(\mathbf{a})$$

(we assume the reader to be familiar with the concept of "satisfiability of a sentence in a structure," sentence $R(\mathbf{a})$ is obtained from formula $R(\mathbf{x})$ by substituting $\mathbf{a}$ for $\mathbf{x}$). S being effective, we can recursively enumerate its terminating paths, some of them possibly inconsistent, as $\langle \pi_i \mid i \in N\rangle$. Corresponding to these paths, there is a recursively enumerable sequence of formulas $\langle R_i(\mathbf{x}) \mid i \in N\rangle$. Rather than draw its binary tree, it should be clear that we can designate S by a recursively enumerable sequence of the form

$$R_0(\mathbf{x}) \wedge \lambda_0,\dots,\ R_i(\mathbf{x}) \wedge \lambda_i,\dots,\ i \in N,$$

where $\lambda_i \in \{\text{YES, NO}\}$ is the label of the last node of terminating path π_i. For later reference, we call the above designation of an effective schema a *R-sequence.* It is clear that we can also write a R-sequence for any other kind of schema, in particular for a flow-chart schema (whose R-sequence can be generated as a regular set, in a sense which can be made rigorous).

Given the R-sequence of an effective p-schema S, we arithmetize the syntax (starting with the "alphabet" $\{r_0, ..., r_l, f_0, ..., f_m\} \cup \{\text{YES, NO}\} \cup \{(,), \neg, \wedge\} \cup (x_1, ..., x_k\}$; then going to the $\mathbf{x}$-terms, and the well-formed atomic and negative atomic formulas; and finally to the finite conjunctions of the form $R(\mathbf{x}) \wedge \lambda$). We next generate the R-sequence of S as a recursively enumerable set of Gödel-numbers $\langle e_i \mid i \in N\rangle$, where $e_i \in N$ is the Gödel-number of "$R_i(x) \wedge \lambda_i$". More specifically, using Lemma 2, $\langle e_i \mid i \in N\rangle$ will be the consecutive output values of a flow-chart f-procedure F_S of $\mathfrak{A}$ when we feed in succession the input values $0, 1, ..., i, ..., i \in N \subseteq A$ (from Recursion Theory: an r.e. set is the range of a total recursive function).

The flow-chart p-procedure $S_0^{\mathfrak{A}}$ which defines the same predicate as $S^{\mathfrak{A}}$ over the domain A is shown in Fig. 1. $y_0, y_1, y_2, ...$ are variables which do not appear among

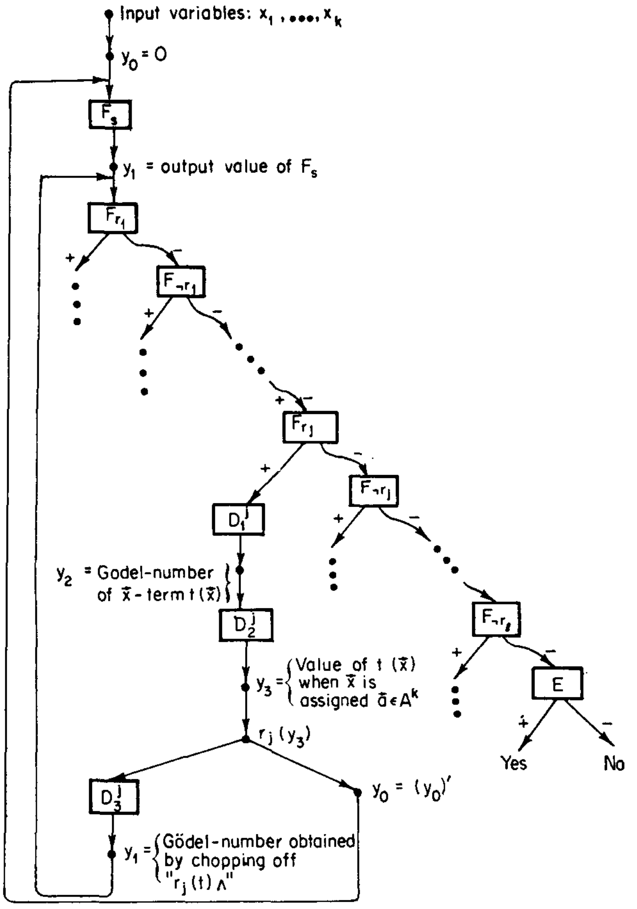

Input variables: $x_1, \dots, x_k$
$y_0 = 0$
F_s
y_1 = output value of F_s
F_{r_1}
$F_{\neg r_1}$
F_{r_j}
$F_{\neg r_j}$
$F_{\neg r_l}$
D_1^j
y_2 = Godel-number of $\vec{x}$-term $t(\vec{x})$
D_2^j
y_3 = Value of $t(\vec{x})$ when $\vec{x}$ is assigned $\vec{a} \in A^k$
$r_j(y_3)$
D_3^j
y_1 = Gödel-number obtained by chopping off "$r_j(t) \wedge$"
$y_0 = (y_0)'$
E
Yes
No
+
−

FIG. 1 (the x and a with arrows are equivalent to the bold face x and a in the text).

$\{x_1, ..., x_k\}$. All the boxes are flow-charts. We explain what each one of them does in the simulation of $S^{\mathfrak{A}}$;

F_S generates the Gödel-numbers of the R-sequence of S;

F_{r_j} (respec. $F_{\neg r_j}$), $0 \leqslant j \leqslant l$, checks whether a Gödel-number is that of an expression which begins with $r_j(t_1, ..., t_w)$ (respectively, $\neg r_j(t_1, ..., t_w)$), where w is the arity of r_j and $t_1, ..., t_w$ are arbitrary $\mathbf{x}$-terms;

D_1^j computes the Gödel-numbers of the w $\mathbf{x}$-terms appearing as arguments of r_j(respectively, $\neg r_j$), from the latter's Gödel-number;

D_2^j computes the values of $t_1, ..., t_w$ at any assignment of input values $a_1, ..., a_k$ from A to $x_1, ..., x_k$;

D_3^j computes the Gödel-number of the expression obtained by chopping off an initial segment of the form "$r_j(t_1, ..., t_w)\wedge$" (respectively, "$\neg r_j(t_1, ..., t_w)\wedge$");

E checks whether a Gödel-number is that of YES or that of NO, and outputs an answer accordingly.

In view of Lemma 2, it is not difficult to see that all the forementioned boxes, except possibly for D_2^j, are flow-chartable. We verify next that D_2^j also can be written as a flow-chart. (So far, without D_2^j, equality $=$ is only needed over $N \subseteq A$. Equality is needed, however, over the entire domain A for the definition of D_2^j, as shown next, thus explaining why totality in structural condition γ has been required.) Before describing the action of D_2^j, observe the following fact [*] (we shall use condition δ in the form given at the end of Section 3): The evaluation of any n-$\mathbf{x}$-term $\bar{t}(\mathbf{x})$ at $a_1, ..., a_k$ can be done by some *fixed* flow-chart f-procedure of $\mathfrak{A}$, say H, with $k + 1$ input variables and using $n +$ a few more other variables; if the input values to H are $a_1, ..., a_k$, and e, where e is the Gödel-number of n-$\mathbf{x}$-term $\bar{t}(\mathbf{x})$, then the output value of H is $\bar{t}^A(a_1, ..., a_k)$. We omit the details of [*]. Note, in passing, that H too needs the equality relation over $N \subseteq A$ only.

With no loss of generality, assume that r_j has just one argument $t(x_1, ..., x_k)$, i.e., r_j is a 1-ary relation symbol. D_2^j is to find the value of $\mathbf{x}$-term $t(\mathbf{x})$ at input values $a_1, ..., a_k \in A$. It proceeds as follows. From the Gödel-number of $t(\mathbf{x})$, it finds the Gödel-number of a sequence of $\mathbf{x}$-terms: $t_1(\mathbf{x}), t_2(\mathbf{x}), ..., t_u(\mathbf{x}) = t(\mathbf{x})$, where each $t_i(\mathbf{x})$, $1 \leqslant i \leqslant u$, is either a variable from $\{x_1, ..., x_k\}$ or $f(t_{i_1}, ..., t_{i_v})$ for some $f \in \{f_0, ..., f_m\}$ and $t_{i_1}, ..., t_{i_v}$ all appear before t_i in the sequence. Next, D_2^j begins enumerating Gödel-numbers of sequences of n-$\mathbf{x}$-terms of length u, of the form: $\bar{t}_1(\mathbf{x}), \bar{t}_2(\mathbf{x}), ..., \bar{t}_u(\mathbf{x})$. Suppose D_2^j stops this enumeration when it finds such a sequence of n-$\mathbf{x}$-terms satisfying:

$$t_1^A(\mathbf{a}) = \bar{t}_1^A(\mathbf{a}), ..., t_u^A(\mathbf{a}) = \bar{t}_u^A(\mathbf{a}). \qquad [**]$$

Observe that, since $\mathfrak{A}$ satisfies structural condition δ, there exists a sequence $\bar{t}_1, ..., \bar{t}_u$ of n-$\mathbf{x}$-terms satisfying [**]. Then, to find the value of $\mathbf{x}$-term $t(\mathbf{x})$ at $\mathbf{a} \in A^k$, D_2^j has only to compute the value of n-$\mathbf{x}$-term $\bar{t}_u(\mathbf{x})$ at $\mathbf{a}$. The latter computation is feasible on a fixed flow-chart H, by fact [*] above.

To complete the proof, it remains to show that D_2^j (a flow-chart) can indeed check whether condition [**] hold, for arbitrary $\mathbf{a} \in A^k$ and arbitrary $u \in N$. Suppose that all the operation symbols $f_0, \ldots, f_m$ are of arity $\leqslant q$. D_2^j will have $(q + 1)$ copies of flow-chart H as subroutines. To check whether $t_i^A(\mathbf{a}) = \bar{t}_i^A(\mathbf{a})$, $1 \leqslant i \leqslant u$, D_2^j performs the following steps

(a) Find which operation symbol is used in going from $t_1, \ldots, t_{i-1}$ to t_i, say it is $f \in \{f_0, \ldots, f_m\}$; that is, t_i is of the form $f(t_{i_1}, \ldots, t_{i_v})$ where $v \leqslant q$ and $i_1, \ldots, i_v$ are all in $\{1, \ldots, i-1\}$;

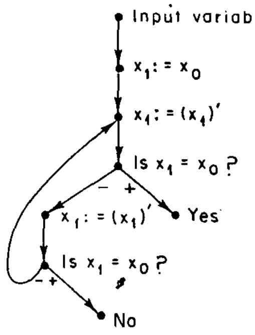
(b) Evaluate $\bar{t}_{i_1}^A(\mathbf{a}), \ldots, \bar{t}_{i_v}^A(\mathbf{a})$, as well as $\bar{t}_i^A(\mathbf{a})$ on $(v + 1)$ copies of H (out of $(q + 1) \geqslant (v + 1)$ available copies of H);

(c) Check whether $f^A(\bar{t}_{i_1}^A(\mathbf{a}), \ldots, \bar{t}_{i_v}^A(\mathbf{a})) = \bar{t}_i^A(\mathbf{a})$, which is possible because $\mathfrak{A}$ satisfies γ; in case of a positive answer, $t_i^A(\mathbf{a})$ is indeed the same as $\bar{t}_i^A(\mathbf{a})$ (but note that at no time does D_2^j have to evaluate $t_i^A(\mathbf{a})$ itself!) and D_2^j goes on to check whether $t_{i+1}^A(\mathbf{a}) = \bar{t}_{i+1}^A(\mathbf{a})$ in a similar way; in case of a negative answer, D_2^j computes the Gödel-number of a new sequence of n-$\mathbf{x}$-terms of length u.

Proof of Proposition III. Define for each integer $i \in N$ a finite structure $\mathfrak{A}_i$ of cardinality $(i + 1)$ such that

$$\begin{aligned}
\mathfrak{A}_i &= \langle A_i ; = ; (\,)' \rangle, \\
A_i &= \{a_{i0}, \ldots, a_{ii}\}, \\
(a_{ij})' &= \begin{cases} a_{i(j+1)}, & \text{if } j < i, \\ a_{i0}, & \text{if } j = i. \end{cases}
\end{aligned}$$

Let the structure $\mathfrak{A}$ be the union $\cup\{\mathfrak{A}_i \mid i \in N\}$. It is clear that $\mathfrak{A}$ satisfies structural condition $\neg\alpha$. Define now S_1 to be the following flow-chart schema



For all inputs $b \in A$, $S_1^{\mathfrak{A}}(b) = \text{YES} \Leftrightarrow$ substructure of $\mathfrak{A}$ generated by b has an odd cardinality; and $S_1^{\mathfrak{A}}(b) = \text{NO} \Leftrightarrow$ substructure of $\mathfrak{A}$ generated by b has an even cardinality. It is not difficult to verify that there is no quasi-loop-free p-procedure of $\mathfrak{A}$, $S_2^{\mathfrak{A}}$, such that $S_1^{\mathfrak{A}} \simeq S_2^{\mathfrak{A}}$.

LEMMA 3. *Let $\mathfrak{A}$ be a finite interpretation, and S a flow-chart schema with k input variables. Given any input value $\mathbf{a} \in A^k$, it is decidable whether $S^{\mathfrak{A}}(\mathbf{a})$ terminates.*

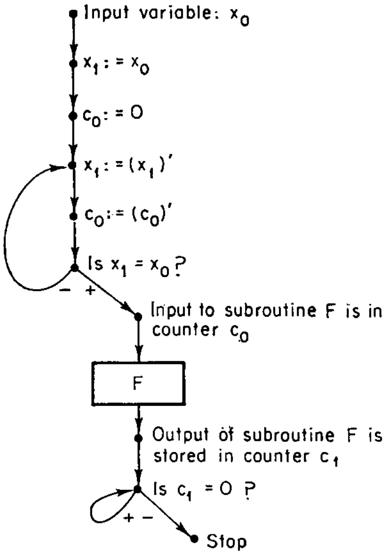
Proof. This is extracted from the second proof of Theorem 1 in [12]. Let the finite cardinality of the domain A be n. Suppose S has t instructions and uses $u \geqslant k$ variables, where t and u are positive integers. Define a *state* of $S^{\mathfrak{A}}$ to be an instruction of $S^{\mathfrak{A}}$ together with an entry from the set $A \cup \{*\}$ for each of the u variables, where the symbol $*$ abbreviates the expression "not yet assigned a value from A." Thus, $S^{\mathfrak{A}}$ has at most $t \cdot (n + 1)^u$ states. If, during the computation of $S^{\mathfrak{A}}$ on input $\mathbf{a} \in A^k$, a state is repeated then $S^{\mathfrak{A}}(\mathbf{a})$ is doomed to loop; i.e., if $S^{\mathfrak{A}}(\mathbf{a})$ is converging, then it must be in $\leqslant t \cdot (n + 1)^u$ steps.

Proof of Proposition IV. We define a flow-chart schema $\bar{S}$ with counters and an interpretation $\mathfrak{A}$ for $\bar{S}$ which satisfies structural condition $\neg\alpha$, such that for no flow-chart schema S (without counters) is it the case that $\bar{S}^{\mathfrak{A}} \simeq S^{\mathfrak{A}}$. The desired interpretation $\mathfrak{A}$ will be the one defined in the proof of Proposition III; it is clear that it satisfies the forementioned structural condition.

As for the definition of $\bar{S}$, note first that there is an effective enumeration of the class of flow-chart schemas (without counters) in the similarity type $\langle =; (\ )' \rangle$ (that of $\mathfrak{A}$). Choose such an enumeration: $S_0, ..., S_i, ..., i \in N$, and each $S_i \in \mathscr{S}_{\mathrm{fc}}$. Using Lemma 3, we can define a total recursive function F such that, for each $i \in N$:

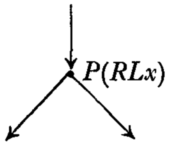
$$F(i) = \begin{cases} 0, & \text{if } S_i^{\mathfrak{A}_i} \text{ is total,} \\ 1, & \text{otherwise,} \end{cases}$$

where $\mathfrak{A}_i$ is the substructure of $\mathfrak{A}$ of size $(i + 1)$. Schema $\bar{S}$ is:

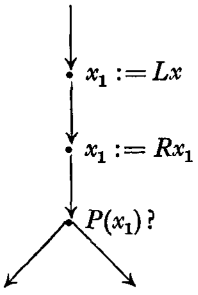


The counters of $\bar{S}$ are denoted by c_0, c_1,.... By Lemma 2 (ii), we can program F on the counters of $\bar{S}$, with c_0 and c_1 as the input and output counters of (the subroutine defining) F. To complete the proof, we have to show that: For all $S_i \in \mathscr{S}_{tc}$, there is an input $b \in A$ (in fact, for all $b \in A_i \subseteq A$), such that $\bar{S}^{\mathfrak{A}}(b)$ converges $\Leftrightarrow$ $S_i^{\mathfrak{A}}(b)$ diverges. We omit the easy proof of this last fact.

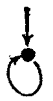
Proof of Proposition V. We first define an effective schema S with one input variable x and in the similarity type $\langle P; L, R, C\rangle$, where P is a 1-ary relation symbol, L and R are 1-ary operation symbols, and C is a constant symbol (which in fact does not appear in S, but will appear in the interpretations for S). For the sake of clarity, we omit the assignment instructions in the binary tree of schema S. The arguments of the test instructions, which are not strictly speaking in their legal form, show which assignment instructions have been omitted; for example, the following test



stands for the following sequence of instructions



We abbreviate the nonterminating paths in S, which are taken to be infinite sequences of the same assignment instruction, by the following



Schema S is shown in Fig. 2. Its finite paths cannot be described by a context-free language (we shall not make this assertion more precise) so that S is not a recursive schema. The important point is that S is drawn effectively, and the set of its finite paths can therefore be described by a type 0 language. It is not difficult to see that the binary tree of S corresponds to no flow-chart schema which has been "unwound". (But note that, by Theorem II, there are interpretations over which S is equivalent to some flow-chart.)

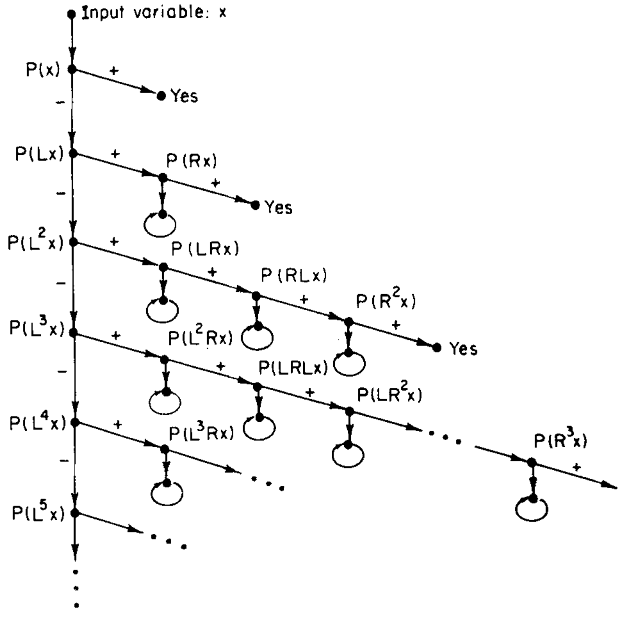


FIGURE 2

We now choose an algebraic structure

$$\mathfrak{A} = \langle A; P^A; L^A, R^A, C^A \rangle$$

which satisfies structural conditions β, $\neg\gamma$, and δ, such that the procedure $S^{\mathfrak{A}}$ is equivalent to no flow-chart procedure of $\mathfrak{A}$. The domain A is the set

$$A = \bigcup_{i \in N} \left\{ \bigcup_{0 \leqslant j \leqslant 2^i} \{a_{ij}, b_{ij}\}^* \right\},$$

where $\{a_{ij}, b_{ij}\}^* = \{\lambda_{ij}, a_{ij}, b_{ij}, (a_{ij})^2, a_{ij}b_{ij}, b_{ij}a_{ij}, (b_{ij})^2, \ldots\}$, i.e., the set of finite words over $\{a_{ij}, b_{ij}\}$ (note that the empty word λ_{ij} is also indexed by i and j). Let $| w |$ denote the number of symbols in the word $w \in A$, and $\| w \|$ denote the natural number obtained by replacing a_{ij} and b_{ij} in w by 0 and 1, respectively ($\| w \|$ is in binary notation). We also agree that $\| \lambda_{ij} \| = 0$. Given $w \in A$, i.e., $w \in \{a_{ij}, b_{ij}\}^*$ for some fixed i and j, relation P^A and operations L^A and R^A are defined as follows:

$$P^A(w) = \begin{cases} \text{true}, & \text{if } \quad | w | = i \quad \text{and} \quad \| w \| \neq j, \\ \text{false}, & \text{if } \quad | w | = i \quad \text{and} \quad \| w \| = j, \\ \text{false, otherwise,} \end{cases}$$

$$L^A(w) = a_{ij}w,$$

$$R^A(w) = b_{ij}w.$$

We choose arbitrarily C^A to be λ_{00} (this is simply to satisfy structural condition β).

To complete the proof, we use the method of the second proof of Theorem 1 in Ref. [12]. We have to show that $S^{\mathfrak{A}}$ is not equivalent to any flow-chart procedure of $\mathfrak{A}$.

Given a flow-chart $\bar{S}$, with t instructions and r variables, a *state* of procedure $\bar{S}^{\mathfrak{A}}$ is specified by an instruction of $\bar{S}^{\mathfrak{A}}$ together with an entry from the set $A \cup \{*\}$ for each of the r variables, where the symbol $*$ stands for the statement "not yet assigned a value from A." Two states of $\bar{S}^{\mathfrak{A}}$, ϕ_1 and ϕ_2, are *equivalent* if the sequences of instructions for the computations continuing from ϕ_1, ϕ_2 are the same. A moment of thought will show that, if we restrict the inputs to $\bar{S}^{\mathfrak{A}}$ to be from $\{a_{i2^i}, b_{i2^i}\}^*$, $\bar{S}^{\mathfrak{A}}$ has at most $t \cdot (i + 3)^r$ equivalence classes, for any $i \in N$. While, if we restrict the inputs to $\{a_{ij}, b_{ij}\}^*$ with $j < 2^i$, $\bar{S}^{\mathfrak{A}}$ has at most $t \cdot (2i + 2)^r$ equivalence classes. (To see this, draw the complete binary tree corresponding to $\{a_{ij}, b_{ij}\}^*$. λ_{ij} is at the root, every left-going arc corresponds to an application of L^A, and every right-going arc corresponds to an application of R^A. Single out every $w \in \{a_{ij}, b_{ij}\}^*$ for which $P^A(w)$ is true; such w is on the ith level of the binary tree. Consider how two values w_1 and w_2 must be located on the binary tree so that changing the assignment of a variable from w_1 to w_2, or vice versa, does not affect the outcome of the computation by $\bar{S}^{\mathfrak{A}}$.)

Let flow-chart $\bar{S}$ have one input variable. If, during a computation by $\bar{S}^{\mathfrak{A}}$, an equivalence class is repeated then the computation is doomed to loop. Hence, if $\bar{S}^{\mathfrak{A}}(\lambda_{ij})$ is supposed to converge, it must be in less than $t \cdot (2i + 3)^r$ steps, and at most that many values are tested during the computation. Now, for i such that $2^i > t \cdot (2i + 3)^r$, it is possible to choose $k < 2^i$ such that the flow-chart procedure $\bar{S}^{\mathfrak{A}}$ cannot distinguish between λ_{i2^i} and λ_{ik}, even though $S^{\mathfrak{A}}(\lambda_{i2^i})$ is YES but $S^{\mathfrak{A}}(\lambda_{ik})$ diverges (the choice of k is determined by the values tested by $\bar{S}^{\mathfrak{A}}$ in computing on input λ_{i2^i}). Hence, $S^{\mathfrak{A}}$ is not equivalent to $\bar{S}^{\mathfrak{A}}$.

Proof of Corollary VI. We consider only the case of $\mathfrak{A}$, from which will immediately follow the case of an expansion $\mathfrak{B}$ of $\mathfrak{A}$. We have the following facts: (i) a sublattice of a distributive lattice is distributive; (ii) the free distributive lattice generated by n elements is finite. The proof of (i) is easy. As for that of (ii), it follows from [17, Theorem 7, Chapter III]. (In [17], Birkhoff proves (ii) for the case when top I and bottom 0 are adjoined, but the proof there can be easily amended for the case without

I and 0.) From (i) and (ii), we readily get: Given any $n \in N$, the n-generated sublattices of a distributive lattice $\mathfrak{A}$ are all of cardinality $\leqslant w$, for some finite w. Hence, using Theorem I, any effective procedure of a distributive lattice $\mathfrak{A}$ (effective relative to $\leqslant$, $\cap$, and $\cup$) can in fact be written as a quasi-loop-free procedure.

Proof of Corollary VII. Rings and fields of characteristic zero are algebraic structures which satisfy structural conditions: β (their domains contain an infinite chain of order type ω), γ (one of their underlying relations is equality $=$), δ (because their operations are either constant or unary or binary which are associative, use condition δ in the form given at the end of Section 3). The desired results are now immediate consequences of Theorem II.

References

1. Abramson, Effective computation over the real numbers, *in* "Proc. of 12th SWAT Symposium," IEEE, 1971.
2. Brown, Gries, and Szymanski, Program schemes with pushdown stores, *SIAM J. Comput.* 1 (1972), 242–268.
3. Chandra and Manna, Program schemas with equality, *in* "Proc. of 4th ACM Symposium on Theory of Computing", 1972.
4. Friedman, Algorithmic procedures, generalized Turing algorithms, and elementary recursion theory, *in* "Logic '69," North-Holland, Amsterdam, 1970.
5. Garland and Luckham, Program schemes, recursion schemes, and formal languages, *J. Comput. Sys. Sci.* 7 (1973).
6. Garland and Luckham, On the equivalence of schemes, *in* "Proc. of 4th ACM Symposium on Theory of Computing," 1972.
7. Kfoury, Comparing algebraic structures up to algorithmic equivalence, *in* "Proc. of IRIA Symposium on Automata, Languages, and Programming, July '72," North-Holland, Amsterdam, 1973.
8. Kreisel, Model-theoretic invariants, *in* "Theory of Models" (Addison, Henkin, and Tarski, Eds.), North-Holland, Amsterdam, 1965.
9. Moschovakis, Abstract computability, *Trans. Am. Math. Society* 138 (1969), 427–504.
10. Paterson, Program schemata, *in* "Machine Intelligence 3," (Michie, Ed.), Edinburgh University, 1968.
11. Paterson, *Equivalence problems in a model of computation*, Ph.D. thesis, Cambridge Univ. 1967 or MIT A.I. Laboratory Technical Memo. no. 1, 1970.
12. Paterson and Hewitt, Comparative schematology, *in* "Project MAC Conf. on Concurrent Systems and Parallel Computation," 1970 or MIT A.I. Laboratory Technical Memo. No. 201.
13. Scott, The lattice of flow-diagrams, *in* "Symposium on Semantics of Algorithmic Languages" (Engeler, Ed.), Springer, Berlin, 1971.
14. Shepherdson and Sturgis, The computability of partial recursive functions, *J. Assoc. Comp. Mach.* 10 (1963), 217–255.
15. Strong, High level languages of maximum power, *in* "Proc. of 12th SWAT Symposium," IEEE, 1971.
16. Strong, Translating recursion equations into flow-charts, *J. Comput. Sys. Sci.* 5 (1971), 254–285.
17. Birkhoff, "Lattice Theory," 3rd ed., Vol. 25, Amer. Math. Soc. Colloquium Pub., 1967.