INSTYTUT MATEMATYCZNY POLSKIEJ AKADEMII NAUK

DISSERTATIONES MATHEMATICAE

(ROZPRAWY MATEMATYCZNE)

LXXXV

ANDRZEJ BLIKLE

Algorithmically definable functions
(A contribution towards the semantics of programming languages)

WARSZAWA 1971
PAŃSTWOWE WYDAWNICTWO NAUKOWE

PRINTED IN POLAND

WROCŁAWSKA DRUKARNIA NAUKOWA

CONTENTS

LIST OF THE MOSTLY USED SOMBOLS

Q	— the set of all formulas (Sec. 2),
I	— the set of all instructions (Sec. 2),
I^*	— the set of all finite sequences (lists) of instructions (Sec. 2),
$F^n(X)$	— the set of all n-argument partial functions with the domain and range in X (Sec. 3),
$F(X)$	— the set of all many-argument partial functions with the domain and range in X (Sec. 3),
$R^n(X)$	— the set of all n-argument relations in X (Sec. 3),
$R(X)$	— the set of all many-argument relations in X (Sec. 3),
M	— a model (Sec.3),
V_M	— the set of all valuations in M (Sec. 3),
T_M	— the set of all transvaluations in M (Sec. 3),
$\mathfrak{A}$	— a flow-algorithm (Sec. 4),
$F_{\mathfrak{A}}$	— the output functions of $\mathfrak{A}$ (Sec. 4),
$\mathrm{ADF}(M)$	— the set of all algorithmically definable functions in M (Sec. 5),
$\sigma(\mathfrak{A}_1, \mathfrak{A}_2)$	— the separation of $\mathfrak{A}_1$ with $\mathfrak{A}_2$ (Sec. 6),
$\mathfrak{A}_1 \circ \mathfrak{A}_2$	— the simple composition of $\mathfrak{A}_1$ with $\mathfrak{A}_2$ (Sec. 6),
$\mathfrak{A}^*$	— the iteration of $\mathfrak{A}$ (Sec. 6),
$\mathfrak{A}_1 * \mathfrak{A}_2$	— the complex composition of $\mathfrak{A}_1$ with $\mathfrak{A}_2$ (Sec. 6),
$\mathfrak{A}_1 \uparrow \mathfrak{A}_2$	— the recursive composition of $\mathfrak{A}_1$ with $\mathfrak{A}_2$ (Sec. 6),
$\mathfrak{A}_1 \to (\mathfrak{A}_2, \mathfrak{A}_3)$	— the conditional composition of $\mathfrak{A}_1$, $\mathfrak{A}_2$ and $\mathfrak{A}_3$ (Sec. 6),
$\overline{\overline{=}}_N$	— the natural equivalence (Sec. 7),
S_n^m	— the operation of substitution (Sec. 7),
R_n^m	— the operation of immediate recursion (Sec. 7),
M	— the operation of minimum (Sec. 7),
$\mathrm{ADR}(M)$	— the set of all algorithmically definable relations in M (Sec. 8),
C_n^m	— the operation of conditional definition (Sec. 8),
$\mathrm{GRF}(M)$	— the class of generalized recursive functions over M (Sec. 9),
$\mathrm{GRR}(M)$	— the set of all generalized recursive relations over M (Sec. 9),
CA	— the class of canonical algorithms (Sec. 10),
$\mathrm{CDF}(M)$	— the set of all canonically definable functions in M (Sec. 10),
$\mathrm{CDR}(M)$	— the set of all canonically definable relations in M (Sec. 10),
A	— the general algebra of algorithms (Sec. 11),
$\sim$	— the equivalence relation in A,
SPL	— simple programming language (Sec. 12),
$\mathrm{PDF}(M)$	— the set of all SPL-definable functions in M (Sec. 12).

1. INTRODUCTION AND SUMMARY*

In this paper we study the semantics of machine-independent programming languages. In particular we are concerned with answering the following three questions:

1. What do we mean — or rather what we intend or should mean — when saying that a given program π computes (defines) a given function f?

2. What can be said about the set of all functions computable (definable) by means of programs in a reasonable class of programming languages?

3. How to prove that a given program π computes a given function f (the problem of adequacy of a program)? In particular, what structures of programming languages are desirable in order to make these proofs sufficiently easy and applicable?

In order to resolve the three problems in a possible formal way, the general intuitive notion of a program should be replaced by some abstract mathematical concept. To this end we introduce the notion of a flow-algorithm (or simply algorithm) which is similar to the well known notion of a *flow-diagram* or *graph-schemata* (cf. [5] or [6]). A *flow-algorithm* is thus a finite, directed, binary ramificated graph (loops are clearly admitted) with exactly one input and exactly one output (see Sec. 3). With every vertex of this graph there is associated a pair $(\alpha; \varrho)$ where α is a finite string of expressions of the form

$$a_i := F_m^n(a_{j_1}, \ldots, a_{j_n}),$$

called *instructions* and ϱ is a single expression of the form

$$R_m^n(a_{j_1}, \ldots, a_{j_n}),$$

called *formula*. Instructions and formulas are elements of a *basic language*, being a language over the union of three infinitely denumerable alphabetes: of *variables* a_i, of *functional symbols* F_m^n and of *predicational sym-*

---

* This paper has been prepared during the stay (October–December, 1968) of the author at Istituto di Elaborazione della Informazione (CNR), Pisa, Italy.

bols R_m^n — and of one finite alphabet of *auxiliary symbols*. By a *model* we mean a pair $M = (X, J)$, where X is an arbitrary nonempty set and J is a function associating partial functions in X with functional symbols, and relations in X with predicational symbols. Now we introduce the concept of a *valuation* to be a total function defined in the set of all variables and with values in X. This notion is similar to the McCarthy's *state-vector* (cf. [7] and [8]), however valuations seem to be more efficient in formulating theorems about programs (flow-algorithms). Now, with every algorithm $\mathfrak{A}$ and every valuation v in a given model M we associate, in a natural way, a sequence $(g_0, v_0), (g_1, v_1), \ldots$ called the *trace* of v in $\mathfrak{A}$, where g_i's are vertices of $\mathfrak{A}$ and v_i's are successive valuations (successive in the course of performance the algorithm $\mathfrak{A}$ with the input (data) valuation v). Clearly, the trace may be finite or infinite. If it is finite and ends with a pair of the form (g_i, v_i), where g_i is the output of $\mathfrak{A}$, then v_i is said to be the value (for v) of the *output function* $F_{\mathfrak{A}}$ of $\mathfrak{A}$ in M. I.e. in this case $F_{\mathfrak{A}}(v) = v_i$. The notion of the output function permits to define the notion of a *function defined by a given algorithm* $\mathfrak{A}$ *in a given model* M (Def. 5.1 in Sec. 5). By ADF(M) we denote the set of all functions algorithmically definable in M.

In this way the question number 1 has been answered. Now we come to the second question, i.e. to the problem concerning the properties of ADF(M). Not much can be said about ADF(M) in the general case, however, we can prove that ADF(M) is never empty and that it is closed under the operation of substitution. On the other hand the models of many programming languages like ALGOL–60, ALGOL–68, FORTRAN, PL–1, etc. have some common properties which can be assumed now for our abstract models to handle with more particular and material cases. To this end we introduce the notion of a *normal model* (Def. 7.1, Sec. 7) to be a model $M = (X, J)$, where $N \subseteq X$ (N is the set of all natural numbers) and where the *successor* in N, the function equalling zero for all x in X, and the *identity relation* in N, belong to the range of J. As it turns out, if M is normal, then ADF(M) has very regular properties. Namely, for every normal model M, the set ADF(M) is the least set of partial many-argument functions in X, which contains a set of elementary functions (elementary in the sense of M) and is closed under three schemes of effective operations (see Sec. 7 and Theorem 9.1 with Definition 9.1 in Sec. 9). It is to be emphasized here that this theorem — which answers the question number 2 — has a general character, since in normal models the set X may contain, besides natural numbers, arbitrary elements like real or complex numbers, matrices, functions, functionals, sets etc. and the range of J may contain, besides the three required objects, any finite or infinite number of arbitrary functions and relations in X.

The answer to the third question is more complicated and requires few preliminary comments. Note first that, given an algorithm $\mathfrak{A}$, we can consider all possible traces in $\mathfrak{A}$, find the output function $F_{\mathfrak{A}}$ and then prove that $\mathfrak{A}$ defines, or not (in the sense of the Definition 5.1), a given function f. This way, although theoretically possible and used in this paper in proving general theorems, has no value for the programming practice, since it leads always to long and complicated proofs. In order to solve the problem of adequacy in a more efficient way, we suggest the following technique: We define two binary operations $\circ$ and $\uparrow$ in the set of all flow-algorithms (Sec. 6) and we prove theorems of the following form: *if* F_1 *and* F_2 *are the output functions of* $\mathfrak{A}_1$ *and* $\mathfrak{A}_2$ *respectively, then the output function of* $\mathfrak{A}_1 \circ \mathfrak{A}_2$ (*or* $\mathfrak{A}_1 \uparrow \mathfrak{A}_2$) *is so and so* (see Lemmas 6.2 and 6.5 in Sec. 6). Now we consider the least set of algorithms which contains the set of all two-vertex, loop-free algorithms (called *elementary*) and is closed under the operations of $\circ$ and $\uparrow$. The elements of this set are called *canonical algorithms* (Sec. 10). Clearly, for every canonical algorithm $\mathfrak{A}$ the problem of adequacy is relatively simple, since if $\mathfrak{A}$ is elementary, then the problem of finding $F_{\mathfrak{A}}$ is very easy, and if $\mathfrak{A}$ is not elementary, then it can be decomposed by means of the operations $\circ$ and $\uparrow$ and appropriate theorems can be applied in finding $F_{\mathfrak{A}}$. The problem is, that not every flowalgorithm is canonical. However, we can prove the following theorem: *for every normal model* M *and every function* f *algorithmically definable in* M *there exists a canonical algorithm which defines* f *in* M. This theorem gives particular suggestions concerning the structure a programming language should have in order to guarantee relatively simple proofs of the adequacy of programs. In Section 12 we describe a simple abstract programming language SPL which follows these suggestions. Now, if the language is specified, we go further in the adequacy-proving technique. Namely we use the fact (the answer to the second question) that for every normal model M the set $\mathrm{ADF}(M)$ is an algebra with three schemes (classes) of operation and we formulate theorems of the following from: *if* f_i *is defined by* $\mathfrak{A}_i$ *for* $i = 1, \ldots, n$ *and* $f = O_j(f_1, \ldots, f_n)$, *then the algorithm which defines* f *is so and so* (cf. Theorems 12.6, 12.7, 12.8 and 12.9). Clearly, these theorems permit to construct algorithms (programs) which — by this construction — are proved to be adequate. And this is the answer to the question number three.

The problem of semantics for programming languages has been investigated by many authors in a number of papers. A very large and complete reference of this subject can be found in a survey paper of J. W. de Bakker [1].

Some main results of the present paper have been published partly in [2], [3] and [4].

2. BASIC LANGUAGE

Let us recall briefly some concepts of the theory of formal languages to be used in this section. By an *alphabet* we shall mean any finite or infinite but denumerable set of symbols. If A is an alphabet, then A^* denotes the set of all finite strings of elements in A (the empty string ε included) called *words* over A. By a *language* over A we mean any subset L of A. If L_1 and L_2 are languages, then

$$L_1 L_2 = \{xy\colon\ x \in L_1 \ \&\ y \in L_2\},$$

where xy is the *concatenation* of x and y,

$$L_1^0 = \{\varepsilon\}, \qquad L_1^n = L_1 L_1^{n-1}, \qquad L_1^* = \bigcup_{i=0}^{\infty} L_1^i.$$

Consider now four alphabet as follows (first three infinite):

$A_V = \{a_0, a_1, \ldots\}$ — *variables*,

$A_F = \{F_m^n\colon\ n, m = 1, 2, \ldots\}$ — *functional symbols*,

$A_R = \{R_m^n\colon\ n, m = 1, 2, \ldots\}$ — *predicational symbols*,

$A_0 = \{:=, (,), '\}$ — *auxiliary symbols*.

Let $A = A_V \cup A_F \cup A_R \cup A_0$. Two kinds of expressions over A will be distinguished:

By a *formula* we mean any word over A of the form

$$R_m^n(a_{j_1}, \ldots, a_{j_n}).$$

Q will denote the set of all formulas.

By an *instruction* we shall mean any word of one of the following two forms:

$$a_i := a_j \quad \text{or} \quad a_i := F_m^n(a_{j_1}, \ldots, a_{j_n}).$$

I will denote the set of all instructions. By the *basic language* we mean the set $L = Q \cup I$.

In the sequel we shall deal frequently with finite sequences, called *lists*, of instructions. According to our notation I^* will denote the set of all lists of instructions.

The language L may seem to be very restricted since the right-hand-side expressions of instructions are always simple terms and formulas are always simple (atomic) formulas. This restriction is however immaterial for our investigations since the alphabets A_F and A_R have been assumed to be infinite.

3. SEMANTICS OF THE BASIC LANGUAGE

Consider an arbitrary set Y. By an *n-argument relation in* Y we mean any subset r of Y^n. The fact that $(y_1, \ldots, y_n) \in r$ may be denoted also by $r(y_1, \ldots, y_n)$. By an *n-argument partial function in* Y we mean any $(n+1)$-argument relation f in Y which satisfies an appropriate "functional" condition. The fact that $(y_1, \ldots, y_n, y_{n+1}) \in f$ is denoted by $f(y_1, \ldots, y_n) = y_{n+1}$ and we say that f *is defined for* $(y_1, \ldots, y_n)$. By $f\colon X \to Y$ we mean that X contains the domain and Y contains the range of f, i.e. that f is a partial function with the domain in X and range in Y. If the domain of f is equal to X, then f is said to be a *total function in* X.

Let now f and g be two arbitrary 1-argument partial functions in X. By a *composition of* f *with* g, in symbols

$$f \circ g,$$

we mean a function h with the following properties:

1. for any x in X, $h(x)$ is defined iff $f(x)$ and $g(f(x))$ are defined,
2. for any x in X, if $h(x)$ is defined, then $h(x) = g(f(x))$.

As can easily be seen, for any f and g there is exactly one h to satisfy properties 1 and 2. We denote this fact by $h = f \circ g$.

The following notation will be used in the sequel:

$\boldsymbol{F}^n(X)$ — the set of all n-argument partial functions with the domain and range in X,

$\boldsymbol{R}^n(X)$ — the set of all n-argument relations in X,

$$\boldsymbol{F}(X) = \bigcup_{n=1}^{\infty} \boldsymbol{F}^n(X), \qquad \boldsymbol{R}(X) = \bigcup_{n=1}^{\infty} \boldsymbol{R}^n(X).$$

DEFINITION 3.1. By a *model* we mean any ordered pair

$$M = (X, J)$$

where X is an arbitrary nonempty set and J is a total function, $J\colon A_F \cup A_R \to \boldsymbol{F}(X) \cup \boldsymbol{R}(X)$, with the following properties:

1° $J(F_m^n) \in \boldsymbol{F}^n(X)$ for any $n, m = 1, 2, \ldots$,

2° $J(R_m^n) \in \boldsymbol{R}^n(X)$ for any $n, m = 1, 2, \ldots$

It is to be emphasized here, that J is not assumed to be reversible. As it follows therefore we can consider, in particular, models with only finite number of nonempty functions and (or) relations in $J(A_F \cup A_R)$. This property has an essential importance for the generality of ours investigations.

Consider now an arbitrary model $M = (X, J)$. By a *valuation* in M we mean any total function

$$v\colon A_V \to X.$$

$\boldsymbol{V}_M$ will denote the set of all valuations in M.

By a *transvaluation* in M we mean any partial function

$$t\colon V_M \to V_M.$$

By T_M we denote the set of all transvaluations in M.

Now with every list of instructions we associate a transvaluation. To this effect we consider a function

$$S_M\colon I^* \to T_M$$

defined in the following way:

1) $S_M(\varepsilon)$ = the *identity transvaluation.*

2) Let $\alpha \in I$. Two cases are to be considered:

(i) If α is of the form $a_i := a_j$ then $S_M(\alpha)$ is a total transvaluation and for every valuation v_1 in V_M if $[S_M(\alpha)](v_1) = v_2$, then for every a_k in A_V

$$v_2(a_k) = \begin{cases} v_1(a_k) & \text{for} \quad k \neq i, \\ v_1(a_j) & \text{for} \quad k = i. \end{cases}$$

(ii) If α is of the form $a_i := F_m^n(a_{j_1}, \ldots, a_{j_n})$, then for any valuation v_1, $[S_M(\alpha)](v_1)$ is defined iff $[J(F_m^n)]\big(v_1(a_{j_1}), \ldots, v_1(a_{j_n})\big)$ is defined, and if $[S_M(\alpha)](v_1)$ is defined, then $[S_M(\alpha)](v_1) = v_2$ where for every a_k in A_V

$$v_2(a_k) = \begin{cases} v_1(a_k) & \text{for} \quad k \neq i, \\ [J(F_m^n)]\big(v_1(a_{j_1}), \ldots, v_1(a_{j_n})\big) & \text{for} \quad k = i. \end{cases}$$

3) For every list $a_1, \ldots, a_n$ in I^*

$$S_M(a_1 \ldots a_n) = S_M(a_1) \circ \ldots \circ S_M(a_n).$$

The notions of the valuation and transvaluation have a clear interpretation in the computer science. In fact, variables can be interpreted as addresses of a hypothetical memory and valuations as memory states. Now with every list of instructions there is associated a function which changes an actual state of the memory to a new one.

Transvaluations describe the meaning of instructions. Now, in order to describe the meaning of a formula, we introduce a new function

$$\Sigma_M\colon Q \to 2^{V_M}$$

to be defined as follows: for every $R_m^n(a_{j_1}, \ldots, a_{j_n})$ in Q

$$\Sigma_M\big(R_m^n(a_{j_1}, \ldots, a_{j_n})\big) = \{v\colon \big(v(a_{j_1}), \ldots, v(a_{j_n})\big) \in J(R_m^n)\}.$$

4. FLOW-ALGORITHMS

As it has been mentioned in the introduction, flow-algorithm is a graph (flow-diagram) where with every vertex there is associated a pair of expressions in the basic language.

By a *flow-diagram* we shall mean a system $\mathfrak{G} = (G, p, q, \Omega)$, where:

1) G is a finite set of natural numbers and the elements of G are called *vertices* of $\mathfrak{G}$,

2) p and q are two different elements in G and are called respectively the *input* and the *output* of $\mathfrak{G}$,

3) Ω is a total function in $G-\{q\}$ with values in $[G-\{p\}]^2$. If for some g_1, g_2, g_3 in G, $\Omega(g_1) = (g_2, g_3)$, then g_2 and g_3 are said to be the *successors* of g_1.

4) For every g in $G-\{p\}$ there exists a *path* from p to g in G, i.e. there exists a string $g_1, \ldots, g_n$ in G with $g_1 = p$, $g_n = g$ and g_{i+1} is a successor of g_i for $i = 1, \ldots, n-1$.

Two flow-diagrams $\mathfrak{G}_1 = (G_1, p_1, q_1, \Omega_1)$ and $\mathfrak{G}_2 = (G_2, p_2, q_2, \Omega_2)$ are said to be *isomorphic* if there exists a 1-1 mapping $\varphi\colon G_1 \to G_2$ such that for every g in $G_1-\{q\}$ if $\Omega_1(g) = (g_1, g_2)$, then $\Omega_2(\varphi(g)) = (\varphi(g_1), \varphi(g_2))$. Clearly, if φ is an isomorphism, then $\varphi(p_1) = p_2$, $\varphi(q_1) = q_2$ and φ^{-1} is an isomorphism too.

DEFINITION 4.1. By a *flow-algorithm* (or simply by an *algorithm*) we mean a system $\mathfrak{A} = (G, p, q, \Omega, \xi)$, where (G, p, q, Ω) is a flow-diagram and ξ is a total function on G with values in $I^* \times Q$. The vertices p and q are called, respectively, the *input* and the *output* of $\mathfrak{A}$. If $\xi(q) = (a; \varrho)$ for some ϱ in Q, then ϱ is called the *output formula* of $\mathfrak{A}$.

The abstract notion of a flow-algorithm replaces the intuitive general concept of a program. Now we shall introduce a new notion to describe the process of computation associated with a given program for given data.

Consider an arbitrary model $M = (X, J)$ and an arbitrary flow-algorithm $\mathfrak{A} = (G, p, q, \Omega, \xi)$. For every valuation v in V_M we associate with $\mathfrak{A}$ a finite or infinite sequence of pairs in $G \times V_M$ called the *trace* of v in $\mathfrak{A}$ (for the given M). The trace

$$(g_0, v_0), (g_1, v_1), \ldots, (g_i, v_i), \ldots$$

is defined by recursion in the following way:

1) Let $\xi(p) = (a; \varrho)$ where $a \in I^*$, $\varrho \in Q$. If $[S_M(a)](v)$ is not defined, then (g_0, v_0) is not defined and the trace of v in $\mathfrak{A}$ is empty. In the opposite case

$$g_0 = p \quad \text{and} \quad v_0 = [S_M(a)](v).$$

2) Let (g_i, v_i) be defined for some $i \geqslant 0$. If $g_i = q$ then (g_{i+1}, v_{i+1}) is not defined and (g_i, v_i) is the last element of the trace. If $g_i \neq q$, then let

$$\Omega(g_i) = (g', g''), \qquad \xi(g_i) = (a; \varrho),$$

$$g^{i+1} = \begin{cases} g' & \text{if} \quad v_i \in \Sigma_M(\varrho), \\ g'' & \text{if} \quad v_i \notin \Sigma_M(\varrho), \end{cases}$$

$$\xi(g^{i+1}) = (\beta, \eta).$$

Now, if $[S_M(\beta)](v_i)$ is not defined, then (g_{i+1}, v_{i+1}) is not defined and (g_i, v_i) is the last element of the trace. In the opposite case

$$g_{i+1} = g^{i+1} \quad \text{and} \quad v_{i+1} = [S_M(\beta)](v_i).$$

Intuitively (g_i, v_i) means that the control of a hypothetical computer performing $\mathfrak{A}$ leaves g_i in the state v_i.

In the sequel we shall frequently deal with traces, i.e. with finite or infinite sequences. To this effect we introduce here the following notation: $\{x_i\}_{i=k}$ is an abbreviation of $x_k, x_{k+1}, \ldots$ (finite or infinite) and $\{x_i\}_{i=k}^{m}$ is an abbreviation of $x_k, x_{k+1}, \ldots, x_m$. In particular $\{(g_i, v_i)\}_{i=0}$ is an abbreviation of $(g_0, v_0), (g_1, v_1), \ldots$

The notion of the trace will play a fundamental role in this paper. Now it permits to introduce a new important notion.

Let $\mathfrak{A}$ be an arbitrary algorithm and M an arbitrary model. By the *output function* of $\mathfrak{A}$ in M we mean a transvaluation $F_{\mathfrak{A}}$ in T_M defined in the following way:

For any v_1, v_2 in V_M, $F_{\mathfrak{A}}(v_1) = v_2$ iff the trace of v_1 in $\mathfrak{A}$ is finite and terminates with (q, v_2), where q is the output of $\mathfrak{A}$. Clearly, if the trace of v_1 in A is either infinite or terminates with a pair (g_i, v_i) where $g_i \neq q$, then $F_{\mathfrak{A}}(v_1)$ is not defined.

Now we shall introduce some auxiliary notions to be used in the sequel. Let B be an arbitrary subset of the set A_V of all variables. Two valuations v_1 and v_2 are said to be *equivalent modulo* B, in symbols

$$v_1 = v_2 \bmod B,$$

if $v_1(a) = v_2(a_i)$ for every a_i in B.

Consider an algorithm $\mathfrak{A}$, a model M, and let $F_{\mathfrak{A}}$ be the output function of $\mathfrak{A}$ in M. A variable a_i is said to be *external* in $\mathfrak{A}$ (for the model M) if for every valuation v, with $F_{\mathfrak{A}}(v)$ defined,

$$[F_A(v)](a_i) = v(a_i).$$

Note that every variable which does not appear in $\mathfrak{A}$ is external in $\mathfrak{A}$ but not conversely.

5. ALGORITHMICALLY DEFINABLE FUNCTIONS

DEFINITION 5.1. Let us consider an arbitrary model $M = (X, J)$, a partial function $f: X^n \to X$ and an algorithm $\mathfrak{A}$. Let $F_{\mathfrak{A}}$ be the output function of $\mathfrak{A}$ in M.

The function f is said to be *defined by* $\mathfrak{A}$ *in* M if there exists a sequence of variables $a_{i_1}, \ldots, a_{i_n}, a_{i_{n+1}}$, where $a_{i_1}, \ldots, a_{i_n}$ is a sequence without repetitions, with the following two properties:

1° For every valuation v in V_M, $F_{\mathfrak{A}}(v)$ is defined iff $f(v(a_{i_1}), \ldots, v(a_{i_n}))$ is defined.

2° For every valuation v in V_M if $F_{\mathfrak{A}}(v)$ is defined then

$$f(v(a_{i_1}), \ldots, v(a_{i_n})) = [F_{\mathfrak{A}}(v)](a_{i_{n+1}}).$$

The sequence $a_{i_1}, \ldots, a_{i_n}, a_{i_{n+1}}$ — written in the sequel as $a_{i_1}, \ldots, a_{i_n}; a_{i_{n+1}}$ — will be called a *matrix* of f in $\mathfrak{A}$. The variables $a_{i_1}, \ldots, a_{i_n}$ will be called *input variables* of the matrix, the variable $a_{i_{n+1}}$ — the *output variable* of the matrix, and a matrix with all input variables being external — an *external matrix*.

Note that in the general case a given function f defined by a given algorithm $\mathfrak{A}$ can have more than one matrix in $\mathfrak{A}$.

EXAMPLE 5.1. Consider an algorithm

$$\mathfrak{A} = (\{p, q\}, p, q, \Omega, \xi),$$

where

$$\Omega(p) = (q, q),$$
$$\xi(p) = (a_1 := F_1^2(a_2, a_3), a_4 := F_1^2(a_5, a_6); R_1^1(a_1)),$$
$$\xi(q) = (\varepsilon; R_1^1(a_1)).$$

Let now $M_1 = (R, J_1)$ where R is the set of all real numbers and $[J_1(F_1^2)](x, y) = x + y$ for $x, y \in R$. Clearly A defines in M_1 the function $f(x, y) = x + y$ with four matrices: $a_2, a_3; a_1, a_3, a_2; a_1, a_5, a_6; a_4$ and $a_6, a_5; a_4$.

Consider now another model $M_2 = (R, J_2)$ where $[J_2(F_1^2)](x, y) = x/y$ for $x, y \in R$ and $y \neq 0$. The algorithm $\mathfrak{A}$ defines in M_2 the following three-argument function:

$$g(x, y, z) = \begin{cases} x/y & \text{if} \quad yz \neq 0, \\ \text{not defined} & \text{if} \quad yz = 0. \end{cases}$$

This function is defined by $\mathfrak{A}$ with two matrices: $a_2, a_3, a_6; a_1$ and $a_5, a_6, a_3; a_4$. If we cancel in $\mathfrak{A}$ the instruction $a_4 := F_1^2(a_5, a_6)$, then the new algorithm, say $\mathfrak{A}_1$, defines in M_2 the quotient function with one matrix $a_2, a_3; a_1$.

Consider an arbitrary model M and a function f in M. This function is said to be *algorithmically definable in* M if there exists an algorithm which defines f in M. By $\mathrm{ADF}(M)$ we denote the set of all functions algorithmically definable in M.

LEMMA 5.1. *For every model* M *and every algorithm* $\mathfrak{A}$ *there exists a function* f *defined by* $\mathfrak{A}$ *in* M, *i.e. for every model* M, $\mathrm{ADF}(M) \neq \emptyset$.

Proof. Consider a model $M = (X, J)$, an algorithm $\mathfrak{A}$ and let $F_{\mathfrak{A}}$ be the output function of $\mathfrak{A}$ in M. Let $a_{i_1}, \ldots, a_{i_n}$ be the only variables in $\mathfrak{A}$. Evidently, for any two valuations v_1 and v_2 with

$$v_1 = v_2 \bmod \{a_{i_1}, \ldots, a_{i_n}\},$$

either both $F_{\mathfrak{A}}(v_1)$ and $F_{\mathfrak{A}}(v_2)$ are not defined or both are defined and

$$F_{\mathfrak{A}}(v_1) = F_{\mathfrak{A}}(v_2) \bmod \{a_{i_1}, \ldots, a_{i_n}\}.$$

Therefore, in particular, if $F_{\mathfrak{A}}(v_1)$ and $F_{\mathfrak{A}}(v_2)$ are defined then

$$[F_{\mathfrak{A}}(v_1)](a_{i_1}) = [F_{\mathfrak{A}}(v_2)](a_{i_1}).$$

It is now easy to see that the function

$$f(x_1, \ldots, x_n) = \begin{cases} [F_{\mathfrak{A}}(v)](a)_{i_1} & \text{if} \quad F_{\mathfrak{A}}(v) \text{ defined}, \\ \text{not defined} & \text{if} \quad F_{\mathfrak{A}}(v) \text{ not defined}, \end{cases}$$

where v is an arbitrary valuation with $v(a_{i_j}) = x_j$ for $j = 1, \ldots, n$, is defined by $\mathfrak{A}$ in M with the matrix $a_{i_1}, \ldots, a_{i_n}; a_{i_1}$, q.e.d.

LEMMA 5.2. *Consider two arbitrary finite sets of variables* B_1 *and* B_2 *with the same number of elements and let* $h\colon B_1 \to B_2$ *be a* 1-1 *mapping. For every algorithm* $\mathfrak{A}_1$ *with the set of variables* B_1 *there exists an algorithm* $\mathfrak{A}_2$ *with the set of variables* B_2 *such that for every model* M *and every function* f *defined by* $\mathfrak{A}_1$ *in* M *with a matrix* $a_{j_1}, \ldots, a_{j_n}; a_{j_{n+1}}$, *the function* f *is defined by* $\mathfrak{A}_2$ *in* M *with the matrix* $h(a_{j_1}), \ldots, h(a_{j_n}); h(a_{j_{n+1}})$.

A simple proof of this lemma will be omitted here.

LEMMA 5.3. *Let* M *be an arbitrary model. For every* n*-argument function* f *in* $\mathrm{ADF}(M)$ *and every string of variables* $a_{j_1}, \ldots, a_{j_n}; a_{j_{n+1}}$, *where* $a_{j_1}, \ldots, a_{j_n}$ *are distinct, there exists an algorithm* $\mathfrak{A}$ *defining* f *in* M *with the matrix* $a_{j_1}, \ldots, a_{j_n}; a_{j_{n+1}}$ *being external in* $\mathfrak{A}$.

Proof. Let $M = (X, J)$ be an arbitrary model and let f be an arbitrary function in $\mathrm{ADF}(M)$. Suppose f is defined in M by $\mathfrak{A} = (G, p, q, \Omega, \xi)$ with a matrix $a_{i_1}, \ldots, a_{i_n}; a_{i_{n+1}}$. Consider now an arbitrary string of variables $a_{j_1}, \ldots, a_{j_n}; a_{j_{n+1}}$ where $a_{j_1}, \ldots, a_{j_n}$ are all distinct. In virtue of Lemma 5.2 the variables $a_{j_1}, \ldots, a_{j_n}$ can be assumed, without loss of generality, as not appearing in $\mathfrak{A}$. Consider now an algorithm

$$\mathfrak{A}_1 = (G, p, q, \Omega, \xi_1),$$

where:

$\xi_1(g) = \xi(g)$ for any g in $G - \{p, q\}$,

$\xi_1(p) = (a_{i_1} := a_{j_n}, \ldots, a_{i_n} := a_{j_n}, \alpha; \varrho)$, where $\xi(p) = (\alpha; \varrho)$,

$\xi_1(q) = (\beta, a_{j_{n+1}} := a_{j_{n+1}}; \omega)$, where $\xi(q) = (\beta; \omega)$.

It is clear now that $a_{j_1}, \ldots, a_{j_n}$ are external in $\mathfrak{A}_1$ and that $\mathfrak{A}_1$ defines f in M with the matrix $a_{j_1}, \ldots, a_{j_n}; a_{j_{n+1}}$ q.e.d.

6. OPERATIONS ON ALGORITHMS

In this section we introduce several operations on algorithms to be used later in proving the theorems concerning ADF(M). First some auxiliary concepts.

Consider two arbitrary algorithms $\mathfrak{A}_i = (G_i, p_i, q_i, \Omega_i, \xi_i)$, where $i = 1, 2$.

$\mathfrak{A}_1$ and $\mathfrak{A}_2$ are said to be *disjoint* if $G_1 \cap G_2 = \emptyset$.

$\mathfrak{A}_1$ is said to be a *subalgorithm* of $\mathfrak{A}_2$ if:

1° $G_1 \subseteq G_2$,

2° $(\forall g \in G_1 - \{q_1\})(\Omega_1(g) = \Omega_2(g))$,

3° $(\forall g \in G_1)(\xi_1(g) = \xi_2(g))$.

Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be two subalgorithms of some algorithm $\mathfrak{A} = (G, p, q, \Omega, \xi)$. $\mathfrak{A}_1$ will be said to *precede* $\mathfrak{A}_2$ in $\mathfrak{A}$, in symbols

$$\mathfrak{A}_1 \underset{\mathfrak{A}}{\prec} \mathfrak{A}_2,$$

if there exists a path (see the definition of a flow-diagram in Sec. 4) in G from q_1 to p_2 but does not exist a path from p_2 to q_1. It follows therefore that if $\mathfrak{A}_1 \underset{\mathfrak{A}}{\prec} \mathfrak{A}_2$, then $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are disjoint.

$\mathfrak{A}_1$ and $\mathfrak{A}_2$ are said to be *isomorphic* if the corresponding flow-diagrams $(G_1, p_1, q_1, \Omega_1)$ and $(G_2, p_2, q_2, \Omega_2)$ are isomorphic and

$$\xi_1(g) = \xi_2(\varphi(g)) \quad \text{for every } g \text{ in } G_1,$$

where $\varphi\colon G_1 \to G_2$ is the corresponding isomorphism.

As it is easy to see, if $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are isomorphic, then the output functions of $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are identical.

By a *separation* of two arbitrary algorithms $A_1 = (G_1, p_1, q_1, \Omega_1, \xi_1)$ and $\mathfrak{A}_2 = (G_2, p_2, q_2, \Omega_2, \xi_2)$, in symbols $\sigma(\mathfrak{A}_1, \mathfrak{A}_2)$, we mean an algorithm

$$\mathfrak{A}_3 = (G_3, p_3, q_3, \Omega_3, \xi_3),$$

where:

$k = \max G_1 + 1$,

$G_3 = \{g + k\colon g \in G_2\}$,

$p_3 = p_2 + k$, $q_3 = q_2 + k$;

for every g in $G_2 - \{q_2\}$ if $\Omega_2(g) = (g_1, g_2)$, then
$\Omega_3(g+k) = (g_1+k, g_2+k)$,
$\xi_3(g+k) = \xi_2(g)$ for every g in G_2.

As it is easy to see, for every $\mathfrak{A}_1$ and $\mathfrak{A}_2$ the algorithms $\mathfrak{A}_1$ and $\sigma(\mathfrak{A}_1, \mathfrak{A}_2)$ are disjoint and $\sigma(\mathfrak{A}_1, \mathfrak{A}_2)$ is isomorphic with $\mathfrak{A}_2$.

Consider two arbitrary algorithms $\mathfrak{A}_1$ and $\mathfrak{A}_2$ and let $\mathfrak{A}_1 = (G_1, p_1, q_1, \Omega_1, \xi_1)$ and $\sigma(\mathfrak{A}_1, \mathfrak{A}_2) = (G_2, p_2, q_2, \Omega_2, \xi_2)$. By a *simple composition* of $\mathfrak{A}_1$ with $\mathfrak{A}_2$, in symbols $\mathfrak{A}_1 \circ \mathfrak{A}_2$, we mean an algorithm

$$A = (G_1 \cup G_2, p_1, q_2, \Omega, \xi),$$

where

$(\forall g \in G_i - \{q_i\})[\Omega(g) = \Omega_i(g)]$ for $i = 1, 2$,
$\Omega(q_1) = (p_2, p_2)$,
$(\forall g \in G_i)[\xi(g) = \xi_i(g)]$ for $i = 1, 2$.

LEMMA 6.1. *The operation of simple composition is associative.*

Proof. Consider three arbitrary algorithms $\mathfrak{A}_i = (G_i, p_i, q_i, \Omega_i, \xi_i)$ for $i = 1, 2, 3$. We shall show that

$$\mathfrak{A}_1 \circ (\mathfrak{A}_2 \circ \mathfrak{A}_3) = (\mathfrak{A}_1 \circ \mathfrak{A}_2) \circ \mathfrak{A}_3.$$

Consider first $\mathfrak{A}_1 \circ (\mathfrak{A}_2 \circ \mathfrak{A}_3)$. This algorithm can be decomposed into three disjoint and successively preceding algorithms

$$\mathfrak{A}_1 \prec \mathfrak{B}_2 \prec \mathfrak{B}_3,$$

where $\mathfrak{B}_2$ and $\mathfrak{B}_3$ are isomorphic with $\mathfrak{A}_2$ and $\mathfrak{A}_3$ respectively. Similarly $(\mathfrak{A}_1 \circ \mathfrak{A}_2) \circ \mathfrak{A}_3$ can be decomposed into

$$\mathfrak{A}_1 \prec \mathfrak{C}_2 \prec \mathfrak{C}_3,$$

where $\mathfrak{C}_2$ and $\mathfrak{C}_3$ are also isomorphic with $\mathfrak{A}_2$ and $\mathfrak{A}_3$ respectively. We shall show now that $\mathfrak{B}_i = \mathfrak{C}_i$ for $i = 2, 3$. Let

φ_i be the isomorphism of $\mathfrak{A}_i$ onto $\mathfrak{B}_i$ for $i = 2, 3$,
ψ_i be the isomorphism of $\mathfrak{A}_i$ onto $\mathfrak{C}_i$ for $i = 2, 3$,
γ_1 be the isomorphism of $\mathfrak{A}_3$ onto $\sigma(\mathfrak{A}_2, \mathfrak{A}_3)$,
γ_2 be the isomorphism of $\mathfrak{A}_2 \circ \mathfrak{A}_3$ onto $\sigma(\mathfrak{A}_1, \mathfrak{A}_2 \circ \mathfrak{A}_3)$,
η_1 be the isomorphism of $\mathfrak{A}_2$ onto $\sigma(\mathfrak{A}_1, \mathfrak{A}_2)$,
η_2 be the isomorphism of $\mathfrak{A}_3$ onto $\sigma(\mathfrak{A}_1 \circ \mathfrak{A}_2, \mathfrak{A}_3)$.

Let now $k_i = \max G_i + 1$. Therefore
$\gamma_1(g) = g + k_2$ for $g \in G_3$,
$\gamma_2(g) = g + k_1$ for g in the set of nodes of $\mathfrak{A}_2 \circ \mathfrak{A}_3$,
$\eta_1(g) = g + k_1$ for $g \in G_1$,
$\eta_2(g) = g + k_1 + k_2$ for $g \in G_3$.

On the other hand

$\varphi_2(g) = \gamma_2(g) = g + k_1$ for $g \in G_2$,

$\varphi_3(g) = [\gamma_1 \circ \gamma_2](g) = \gamma_2(\gamma_1(g)) = g + k_1 + k_2$ for $g \in G_3$,

$\psi_2(g) = \eta_1(g) = g + k_1$ for $g \in G_2$,

$\psi_3(g) = \eta_2(g) = g + k_1 + k_2$ for $g \in G_3$.

Thus $\varphi_2 = \psi_2$ and $\varphi_3 = \psi_3$, q.e.d.

LEMMA 6.2. *Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be arbitrary algorithms and M — an arbitrary model. If F_1, F_2 and F are the output functions in M of $\mathfrak{A}_1$, $\mathfrak{A}_2$ and $\mathfrak{A}_1 \circ \mathfrak{A}_2$ respectively, then $F = F_1 \circ F_2$.*

Proof. Let $\mathfrak{A}_2^0 = \sigma(\mathfrak{A}_1, \mathfrak{A}_2)$ and let p_1, q_1 and p_2, q_2 be the input and the output of $\mathfrak{A}_1$ and $\mathfrak{A}_2^0$ respectively. Consider $\mathfrak{A} = \mathfrak{A}_1 \circ \mathfrak{A}_2$ and let $\{(g_i, v_i)\}_{i=0}$ be the trace of v in $\mathfrak{A}$ for some valuation v. Evidently $\mathfrak{A}_1 \prec_{\mathfrak{A}} \mathfrak{A}_2^0$ thus $\{(g_i, v_i)\}_{i=0}$ has the following property:

either $\{(g_i, v_i)\}_{i=0}$ is the trace of v in $\mathfrak{A}_1$ or there exists a natural number $j > 1$ such that $\{(g_i, v_i)\}_{i=0}^{j}$ is the trace of v in $\mathfrak{A}_1$, $g_j = q_1$, (g_{j+1}, v_{j+1}) is defined and $\{(g_{j+k}, v_{j+k})\}_{k=1}$ is the trace of v_j in $\mathfrak{A}_2^0$.

Thus, if $F_1(v)$ and $F_2(F_1(v))$ are defined, then $F(v)$ is defined and $F(v) = F_2(F_1(v))$. (Clearly the output function of $\sigma(\mathfrak{A}_1, \mathfrak{A}_2)$ is F_2). Conversely, if $F(v)$ is defined then the j mentioned above exists and therefore both $v_j = F_1(v)$ and $F_2(v_j)$ are defined with $F(v) = F_2(v_j) = F_2(F_1(v))$, q.e.d.

Let $\mathfrak{A} = (G, p, q, \Omega, \xi)$ be an arbitrary algorithm. By an *iteration* of $\mathfrak{A}$, in symbols $\mathfrak{A}^*$, we shall mean an algorithm

$$A^* = (G \cup \{p^*, q^*\}, p^*, q^*, \Omega^*, \xi^*),$$

where

$p^* = \max G + 1$,

$q^* = \max G + 2$,

$(\forall g \in G - \{q\})[\Omega^*(g) = \Omega(g)]$,

$\Omega^*(p^*) = (p, p)$,

$\Omega^*(q) = (q^*, p)$,

$(\forall g \in G)[\xi^*(g) = \xi(g)]$,

$\xi^*(p^*) = (\varepsilon; R_1^1(a_i))$, where i is the least index of a variable in $\mathfrak{A}$, $\xi^*(q^*) = (\varepsilon; \varrho)$, where ϱ is the output formula of $\mathfrak{A}$ (see Fig. 6.1).

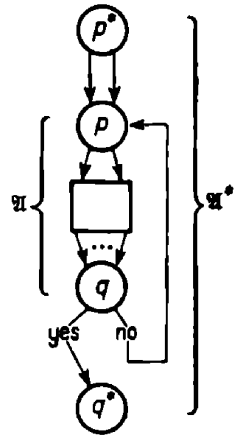

Fig 6.1

LEMMA 6.3. *Let $\mathfrak{A}$ be an arbitrary algorithm with an output formula ϱ. Let M be an arbitrary model and F, F^* — the output functions in M of $\mathfrak{A}$ and $\mathfrak{A}^*$ respectively.*

For every valuation v, $F^*(v)$ is defined iff there exists an $i \geqslant 1$ with $F^1(v), F^2(v), \ldots, F^i(v)$ defined and with $F^i(v) \in \Sigma_M(\varrho)$ (F^i stands for $F \circ \ldots \circ F$

i-times), *and if* $F^*(v)$ *is defined, then* $F^*(v) = F^j(v)$, *where* j *is the least* i *with* $F^i(v) \in \Sigma_M(\varrho)$.

Proof. Let v be an arbitrary valuation and let $F^*(v)$ be defined. Suppose $\{(g_i, v_i)\}_{i=0}^n$ is the trace of v in $\mathfrak{A}$. Evidently there exists a sequence of indices $i_1 < i_2 < \ldots < i_k$ with the four following properties:

1) $g_{i_j} = q$, $g_{i_j+1} = p$ for $j = 1, \ldots, k-1$,
$g_{i_k} = q$, $g_{i_k+1} = q$, $i_k + 1 = n$;

2) the sequences

$$(g_2, v_2), \ldots, (g_{i_1}, v_{i_1}),$$
$$(g_{i_1+1}, v_{i_1+1}), \ldots, (g_{i_2}, v_{i_2}),$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$(g_{i_{k-1}+1}, v_{i_{k-1}+1}), \ldots, (g_{i_k}, v_{i_k})$$

are traces in $\mathfrak{A}$ of $v, v_{i_1}, \ldots, v_{i_{k-1}}$ respectively;

3) $v_{i_1} \notin \Sigma_M(\varrho), \ldots, v_{i_{k-1}} \notin \Sigma_M(\varrho)$ but $v_{i_k} \in \Sigma_M(\varrho)$;

4) $v_{i_1} = F(v)$, $v_{i_{j+1}} = F(v_{i_j})$ for $j = 1, \ldots, k-1$.

Therefore $v_{i_k} = F^k(v)$ and k is the least j with $F^j(v) \in \Sigma_M(\varrho)$. But $v_{i_k} = v_n = F^*(v)$.

It is easy to prove now, that for every valuation v if $F^i(v)$ are defined for $i = 1, \ldots, k$ and $F^i(v) \notin \Sigma_M(\varrho)$ for $i = 1, \ldots, k-1$, but $F^k(v) \in \Sigma_M(\varrho)$, then $F^*(v)$ is defined and $F^*(v) = F^k(v)$ q.e.d.

Consider two arbitrary algorithms $\mathfrak{A}_1$ and $\mathfrak{A}_2$ and let $\mathfrak{A}_1 = (G_1, p_1, q_1, \Omega_1, \xi_1)$ and $\sigma(\mathfrak{A}_1, \mathfrak{A}_2) = (G_2, p_2, q_2, \Omega_2, \xi_2)$. By a *complex composition* of $\mathfrak{A}_1$ and $\mathfrak{A}_2$, in symbols $\mathfrak{A}_1 * \mathfrak{A}_2$, we shall mean an algorithm

$$\mathfrak{A} = (G_1 \cup G_2 \cup \{q\}, p_1, q, \Omega, \xi),$$

where

$q = \max(G_1 \cup G_2) + 1$,
$(\forall g \in G_i - \{q_i\})[\Omega(g) = \Omega_i(g)]$ for $i = 1, 2$,
$\Omega(q_1) = (q, p_2)$,
$\Omega(q_2) = (q, q)$,
$(\forall g \in G_i)[\xi(g) = \xi_i(g)]$ for $i = 1, 2$,
$\xi(q) = (\varepsilon; \varrho_2)$, where ϱ_2 is the output formula of $\mathfrak{A}_2$ (see Fig. 6.2).

Evidently the operation of complex composition is not associative.

LEMMA 6.4 *Let* $\mathfrak{A}_1$ *and* $\mathfrak{A}_2$ *be two arbitrary algorithms. Let* ϱ *be the output formula of* $\mathfrak{A}_1$ *and let* M *be an arbitrary model. If* F_1, F_2 *and* F *are the output functions in* M *of* $\mathfrak{A}_1, \mathfrak{A}_2$ *and* $\mathfrak{A}_1 * \mathfrak{A}_2$ *respectively, then for every valuation* v, $F(v)$ *is defined iff either* $F_1(v)$ *is defined and*

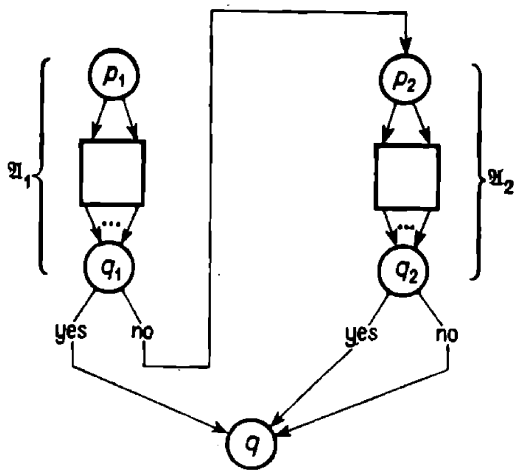

Fig. 6.2

$F_1(v) \in \Sigma_M(\varrho)$ or $[F_1 \circ F_2](v)$ is defined and $F_1(v) \notin \Sigma_M(\varrho)$, and if $F(v)$ is defined, then

$$F(v) = \begin{cases} F_1(v) & \text{if} \quad F_1(v) \in \Sigma_M(\varrho), \\ [F_1 \circ F_2](v) & \text{if} \quad F_1(v) \notin \Sigma_M(\varrho). \end{cases}$$

Proof. Let $\mathfrak{A}_2^0 = \sigma(\mathfrak{A}_1, \mathfrak{A}_2)$. The output function of $\mathfrak{A}_2^0$ is evidently equal to F_2. Consider an arbitrary valuation v and let $F(v)$ be defined. Suppose $\{(g_i, v_i)\}_{i=0}^n$ is the trace of v in $\mathfrak{A} = \mathfrak{A}_1 * \mathfrak{A}_2$. Since $\mathfrak{A}_1 \underset{\mathfrak{A}}{\prec} \mathfrak{A}_2^0$, there exists an $m < n$ with $g_m = q_1$. Thus $\{(g_i, v_i)\}_{i=0}^m$ is the trace of v in $\mathfrak{A}_1$ and $F_1(v) = v_m$ is defined. Now, if $v_m \in \Sigma_M(\varrho)$, then $n = m+1$ and $g_{m+1} = q$, thus $F(v) = F_1(v)$. If however $v_m \notin \Sigma_M(\varrho)$, then $m+1 < n$ and $\{(g_{m+k}, v_{m+k})\}_{k=1}^{n-m-1}$ is the trace of v_m in $\mathfrak{A}_2$. In this case $F(v) = v_n = v_{n-1} = F_2(F_1(v))$.

Now one can easily prove that for every valuation v, if either $F_1(v)$ is defined and $F_1(v) \in \Sigma_M(\varrho)$ or $F_1(v)$ is defined, $F_1(v) \notin \Sigma_M(\varrho)$ and $F_2(F_1(v))$ is defined, then $F(v)$ is defined and satisfies the required equality, q.e.d.

By a *recursive composition* of two algorithms $\mathfrak{A}_1$ and $\mathfrak{A}_2$, in symbols $\mathfrak{A}_1 \uparrow \mathfrak{A}_2$, we mean the algorithm

$$\mathfrak{A}_1 \uparrow \mathfrak{A}_2 = \mathfrak{A}_1 * \mathfrak{A}_2^*.$$

Clearly we assume here that the iteration is stronger than the complex composition, i.e. that $\mathfrak{A}_1 \uparrow \mathfrak{A}_2$ is the complex composition of $\mathfrak{A}_1$ and $\mathfrak{A}_2^*$. Evidently, the recursive composition is not associative.

In virtue of Lemmas 6.3 and 6.4 we conclude immediately the following:

LEMMA 6.5. *Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be two arbitrary algorithms and let ϱ_1 and ϱ_2 be the output formulas of $\mathfrak{A}_1$ and $\mathfrak{A}_2$ respectively. Let M be an arbitrary model and let F_1, F_2 and F be the output functions of $\mathfrak{A}_1$, $\mathfrak{A}_2$ and $\mathfrak{A}_1 \uparrow \mathfrak{A}_2$ respectively.*

For every valuation v, $F(v)$ is defined iff either $F_1(v)$ is defined and $F_1(v) \in \Sigma_M(\varrho_1)$ or $F_1(v)$ is defined, $F_1(v) \notin \Sigma_M(\varrho_1)$ and there exists an i with $[F_1 \circ F_2^1](v), \ldots, [F_1 \circ F_2^i](v)$ defined and with $[F_1 \circ F_2^i](v) \in \Sigma_M(\varrho_2)$.

For every valuation v, if $F(v)$ is defined then

$$F(v) = \begin{cases} F_1(v) & \text{if} \quad F_1(v) \in \Sigma_M(\varrho_1), \\ [F_1 \circ F_2^j](v) & \text{if} \quad F_1(v) \notin \Sigma_M(\varrho_1), \end{cases}$$

where j is the least i with $[F_1 \circ F_2^i](v) \in \Sigma_M(\varrho_2)$. (Clearly F_2^i means $F_2 \circ \ldots \circ F_2$ i-times).

Consider three arbitrary algorithms $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{A}_3$ and let

$$\mathfrak{A}_1 = (G_1, p_1, q_1, \Omega_1, \xi_1),$$
$$\sigma(\mathfrak{A}_1, \mathfrak{A}_2) = (G_2, p_2, q_2, \Omega_2, \xi_2) = \mathfrak{A}_2',$$
$$\sigma(\mathfrak{A}_1 \circ \mathfrak{A}_2, \mathfrak{A}_3) = (G_3, p_3, q_3, \Omega_3, \xi_3) = \mathfrak{A}_3'.$$

Clearly $\mathfrak{A}_1$, $\mathfrak{A}_2'$ and $\mathfrak{A}_3'$ are mutually disjoint and $\mathfrak{A}_2'$, $\mathfrak{A}_3'$ are isomorphic with $\mathfrak{A}_2$, $\mathfrak{A}_3$ respectively.

By a *conditional composition* of $\mathfrak{A}_1$, $\mathfrak{A}_2$ and $\mathfrak{A}_3$, in symbols $\mathfrak{A}_1 \to (\mathfrak{A}_2, \mathfrak{A}_3)$, we mean an algorithm

$$\mathfrak{A} = (G_1 \cup G_2 \cup G_3 \cup \{q\}, p_1, q, \Omega, \xi),$$

where

$q = \max(G_1 \cup G_2 \cup G_3) + 1$,
$(\forall g \in G_i - \{q_i\})[\Omega(g) = \Omega_i(g)]$ for $i = 1, 2, 3$,
$\Omega(q_1) = (p_2, p_3)$,
$\Omega(q_2) = (q, q)$,
$\Omega(q_3) = (q, q)$,
$(\forall g \in G_i)[\xi(g) = \xi_i(g)]$ for $i = 1, 2, 3$,
$\xi(q) = (\varepsilon; \varrho_3)$, where ϱ_3 is the output formula of $\mathfrak{A}_3$.

LEMMA 6.6. *Let $\mathfrak{A}_1, \mathfrak{A}_2$ and $\mathfrak{A}_3$ be three arbitrary algorithms and let ϱ_1 be the output formula of $\mathfrak{A}_1$. Consider an arbitrary model M and let F_1, F_2, F_3 and F be the output functions of $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{A}_3$ and $\mathfrak{A}_1 \to (\mathfrak{A}_1, \mathfrak{A}_2)$ respectively.*

For every valuation v, $F(v)$ is defined iff $F_1(v)$ is defined and either $F_2(F_1(v))$ is defined and $F_1(v) \in \Sigma_M(\varrho_1)$ or $F_3(F_1(v))$ is defined and $F_1(v) \notin \Sigma_M(\varrho_1)$ and if $F(v)$ is defined, then

$$F(v) = \begin{cases} F_2(F_1(v)) & \text{if} \quad F_1(v) \in \Sigma_M(\varrho_1), \\ F_3(F_1(v)) & \text{if} \quad F_1(v) \notin \Sigma_M(\varrho_1). \end{cases}$$

We omit the simple proof of this lemma.

7. NORMAL MODELS

Not much can be said about ADF(M) in the general case. As we have proved (see Lemma 5.1) ADF(M) is never empty, and as we shall prove in this section, ADF(M) is always closed under the operation of substitution and contains all the basic functions, i.e. all functions of the set $J(A_F)$. It seems that no more interesting results can be expected in the general case.

Now we shall restrict our attention to a particular class of models, however large enough to cover models of at least the numerical-purpose programming languages. E.g. the models of ALGOL or FORTRAN will be contained in this class.

DEFINITION 7.1. A model $M = (X, J)$ is said to be *normal* if the following conditions are satisfied:

1° $N \subseteq X$, where N is the set of all natural numbers,

2° the functions $0(x) = 0$ for all x in X and $S(x) = x+1$ for all x in N are in $J(A_F)$,

3° there exists in $J(A_R)$ a two-argument relation $x \underset{N}{=} y$ with the following property:

$$(\forall x, y \in N)[x \underset{N}{=} y \Leftrightarrow x = y],$$

where $=$ is the identity relation in X.

Consider now an arbitrary normal model $M = (X, J)$. In the set $F(X) = \bigcup_{n=0}^{\infty} F^n(X)$ (see Sec. 3) of all partial many-argument functions in X we define three schemes (classes) of operations. For the sake of simplicity $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \ldots$ will denote vectors in X.

1) *The operation of substitution.* Let $h \in F^m(X)$, $g_1, \ldots, g_m \in F^n(X)$ for some $m, n \geqslant 1$. By

$$f = \mathbf{S}_n^m(h, g_1, \ldots, g_m)$$

we shall mean a function in $F^n(X)$ with the following properties: for any $\boldsymbol{x}$ in X^n, $f(\boldsymbol{x})$ is defined iff $g_1(\boldsymbol{x}), \ldots, g_m(\boldsymbol{x})$, $h\big(g_1(\boldsymbol{x}), \ldots, g_m(\boldsymbol{x})\big)$ are all defined and if $f(\boldsymbol{x})$ is defined, then $f(\boldsymbol{x}) = h\big(g_1(\boldsymbol{x}), \ldots, g_m(\boldsymbol{x})\big)$.

2) *The operation of immediate recursion.* Let $g_1, \ldots, g_m \in F^n(X)$, $h_1, \ldots, h_m \in F^{n+m+1}(X)$ for some $n, m \geqslant 1$. By

$$(f_1, \ldots, f_m) = \mathbf{R}_n^m(g_1, \ldots, g_m, h_1, \ldots, h_m)$$

we shall mean a string of functions in $F^{n+1}(X)$ which satisfies the following conditions:

(i) for any x not natural and any $\boldsymbol{x}$ in X^n, $f_i(x, \boldsymbol{x})$ is not defined for $i = 1, \ldots, m$,

(ii) for any $\boldsymbol{x}$ in X^n, $f_i(0, \boldsymbol{x})$ is defined iff $g_i(\boldsymbol{x})$ is defined, and if $_i(0, \boldsymbol{x})$ is defined, then $f_i(0, \boldsymbol{x}) = g_i(\boldsymbol{x})$ for $i = 1, \ldots, m$,

(iii) for any $\boldsymbol{x}$ in X^n and any $k \geqslant 1$ natural, $f_i(k+1, \boldsymbol{x})$ is defined iff $f_1(k, \boldsymbol{x}), \ldots, f_m(k, \boldsymbol{x})$ and $h_i\big(k, \boldsymbol{x}, f_1(k, \boldsymbol{x}), \ldots, f_m(k, \boldsymbol{x})\big)$ are defined, and if $f_i(k+1, \boldsymbol{x})$ is defined, then

$$f_i(k+1, \boldsymbol{x}) = h_i\big(k, \boldsymbol{x}, f_1(k, \boldsymbol{x}), \ldots, f_m(k, \boldsymbol{x})\big) \quad \text{for } i = 1, \ldots, m.$$

3) *The operation of minimum.* Let $h \in \boldsymbol{F}^{n+1}(X)$ for some $n \geqslant 1$. By

$$f = \boldsymbol{M}(h)$$

we shall mean a function in X^n with the following properties:

(i) for any $\boldsymbol{x}$ is X^n, $f(\boldsymbol{x})$ is defined iff there exists an $i \geqslant 0$ natural with $h(0, \boldsymbol{x}), \ldots, h(i, \boldsymbol{x})$ defined and $h(i, \boldsymbol{x}) = 0$,

(ii) for any $\boldsymbol{x}$ in X^n, if $f(\boldsymbol{x})$ is defined, then $f(\boldsymbol{x})$ is the least i with $h(i, \boldsymbol{x}) = 0$.

It is easy to prove that all the three classes of operations are well defined. This proof will be omitted here.

Before investigating normal models we shall prove two theorems concerning the general case.

THEOREM 7.1. *For every model* $M = (X, J)$, $J(A_F) \subseteq \mathrm{ADF}(M)$.

Proof. Let $F_j^n \in A_F$. The algorithm defining $J(F_j^n)$ is the following:

$$\mathfrak{A} = (\{p, q\}, p, q, \Omega, \xi),$$

where

$\Omega(p) = (q, q)$,

$\xi(p) = \big(a_1 := F_j^m(a_2, \ldots, a_{n+1}); R_1^1(a_1)\big)$,

$\xi(q) = \big(\varepsilon; R_1^1(a_1)\big)$, q.e.d.

THEOREM 7.2. *For every model* M *the set* $\mathrm{ADF}(M)$ *is closed under the operations of substitution.*

Proof. Consider a model $M = (X, J)$ and let $h, g_1, \ldots, g_m \in \mathrm{ADF}(M)$, where $h \in \boldsymbol{F}^m(X)$ and $g_i \in \boldsymbol{F}^n(X)$ for $i = 1, \ldots, m$. We shall show that

$$f = \boldsymbol{S}_n^m(h, g_1, \ldots, g_m) \in \mathrm{ADF}(M).$$

Let, for every $i = 1, \ldots, m$,

$$\mathfrak{A}_i = (G_i, p_i, q_i, \Omega_i, \xi_i)$$

be an algorithm defining g_i. Let

$$\mathfrak{A}_{m+1} = (G_{m+1}, p_{m+1}, q_{m+1}, \Omega_{m+1}, \xi_{m+1})$$

be an algorithm defining h. Without any loss of generality (cf. Lemma 5.2 and Lemma 5.3) we can assume the following:

1) there exist variables $a_{k_1}, \ldots, a_{k_m}$ such that, for every $j = 1, \ldots, m$, g_j is defined by $\mathfrak{A}_j$ with an external matrix $a_1, \ldots, a_n$; a_{k_j} and a_{k_j} is external in $\mathfrak{A}_{j+1}, \ldots, \mathfrak{A}_m$.

2) h is defined by $\mathfrak{A}_{m+1}$ with a matrix $a_{k_1}, \ldots, a_{k_m}$; a_s for some variable a_s.

Consider now the algorithm

$$\mathfrak{A} = \mathfrak{A}_1 \circ \mathfrak{A}_2 \circ \ldots \circ \mathfrak{A}_m \circ \mathfrak{A}_{m+1}.$$

We shall show that $\mathfrak{A}$ defines f with a matrix $a_1, \ldots, a_n$; a_s. Let $F_1, \ldots, F_{m+1}$ and F be the output functions of $\mathfrak{A}_1, \ldots, \mathfrak{A}_{m+1}$ and $\mathfrak{A}$ respectively. By Lemma 6.2

$$F = F_1 \circ \ldots \circ F_{m+1}.$$

We shall first show that the condition 2° of Definition 5.1 is satisfied for f. Let for some valuation v, $F(v)$ be defined. Therefore $[F_1 \circ \ldots \circ F_i](v)$ is defined for every $i = 1, \ldots, m+1$. By virtue of the assumption 1) we can easily prove the following: for every $i = 1, \ldots, m$ and every $j \leqslant i$

$$\big[[F_1 \circ \ldots \circ F_i](v)\big](a_{k_j}) = g_j\big(v(a_1), \ldots, v(a_n)\big).$$

Thus, in particular, for every $j = 1, \ldots, m$

$$\big[[F_1 \circ \ldots \circ F_m](v)\big](a_{k_j}) = g_j\big(v(a_1), \ldots, v(a_n)\big).$$

Therefore, by the assumption 2),

$$\begin{aligned}[F(v)](a_s) &= h\big(g_1(v(a_1), \ldots, v(a_n)), \ldots, g_m(v(a_1), \ldots, v(a_n))\big) \\ &= f\big(v(a_1), \ldots, v(a_n)\big).\end{aligned}$$

It is easy to prove now, that 1° of Definition 5.1 is also satisfied, q.e.d.

THEOREM 7.3. *For every normal model M the set* $\mathrm{ADF}(M)$ *is closed under every of the operations* $\boldsymbol{R}_n^m$.

Proof. Consider an arbitrary normal model $M = (X, J)$. To simplify the notation let $S, 0, \overline{\overline{N}}$ denote in the basic language the successor, the zero-function and an arbitrary relation satisfying the condition 3° in Definition 7.1. Let now for some $n, m \geqslant 1$

$$(f_1, \ldots, f_m) = \boldsymbol{R}_n^m(g_1, \ldots, g_m, h_1, \ldots, h_m),$$

where $g_1, \ldots, g_m, h_1, \ldots, h_m \in \mathrm{ADF}(M)$. We shall show, that

$$f_1, \ldots, f_m \in \mathrm{ADF}(M).$$

Let for every $i = 1, \ldots, m$

$$\mathfrak{A}_i^1 = (G_i^1, p_i^1, q_i^1, \Omega_i^1, \xi_i^1)$$

be an algorithm which defines g_i in M and let for every $j = 1, \ldots, m$

$$\mathfrak{A}_j^2 = (G_j^2, p_j^2, q_j^2, \Omega_j^2, \xi_j^2)$$

be an algorithm which defines h_j in M. Without any loss of generality (cf. Lemma 5.2 and Lemma 5.3) we can assume the following:

1) For every $i = 1, \ldots, m$, g_i is defined by $\mathfrak{A}_i^1$ with an external matrix $a_1, \ldots, a_n, a_{k_i}$, where a_{k_i} does not appear in any of the algorithms $\mathfrak{A}_{i+1}^1, \ldots, \mathfrak{A}_m^1$. (It follows therefore, that $a_{k_i} \neq a_{k_j}$ for $i \neq j$.)

2) For every $j = 1, \ldots, m$, h_j is defined by $\mathfrak{A}_j^2$ with an external matrix $a_p, a_1, \ldots, a_n, a_{k_1}, \ldots, a_{k_m}; a_{s_j}$ where a_{s_j} does not appear in any of the algorithms $\mathfrak{A}_{j+1}^2, \ldots, A_m^2$.

Let us consider now two auxiliary algorithms:

$$\mathfrak{A}_1 = (\{p_1, q_1\}, p_1, q_1, \Omega_1, \xi_1),$$

where

$\Omega_1(p_1) = (q_1, q_1)$,

$\xi_1(p_1) = \big(a_p := 0(a_p); R_1^1(a_p)\big)$,

$\xi_1(q_1) = (\varepsilon; a_s \underset{N}{\equiv} a_p)$, where a_s does not appear in any of $\mathfrak{A}_1^1, \ldots, \mathfrak{A}_m^1$, $\mathfrak{A}_1^2, \ldots, \mathfrak{A}_m^2$, and the other algorithm

$$\mathfrak{A}_2 = (\{p_2, q_2\}, p_2, q_2, \Omega_2, \xi_2),$$

where

$\Omega_2(p_2) = (q_2, q_2)$,

$\xi_2(p_2) = \big(a_{k_1} := a_{s_1}, \ldots, a_{k_m} := a_{s_m}; R_1^1(a_1)\big)$,

$\xi_2(q_2) = (a_p := S(a_p); a_s \underset{N}{\equiv} a_p)$.

Now we are ready to construct the algorithm defining $(f_1, \ldots, f_m)$. It is following (cf. also Fig. 7.1):

$$\mathfrak{A} = (\mathfrak{A}_1^1 \circ \ldots \circ \mathfrak{A}_m^1 \circ \mathfrak{A}_1) \uparrow (\mathfrak{A}_1^2 \circ \ldots \circ \mathfrak{A}_m^2 \circ \mathfrak{A}_2).$$

We shall prove, that for every $i = 1, \ldots, m$, $\mathfrak{A}$ defines f_i with a matrix $a_s, a_1, \ldots, a_n; a_{k_i}$. Let us introduce the following notation:

F_j^i is the output function of $\mathfrak{A}_j^i$ for $i = 1, 2$, $j = 1, \ldots, m$,

F_i is the output function of $\mathfrak{A}_i$ for $i = 1, 2$,

F^* is the output function of the algorithm $(\mathfrak{A}_1^2 \circ \ldots \circ \mathfrak{A}_m^2 \circ \mathfrak{A}_2)^*$,

F is the output function of $\mathfrak{A}$.

Consider now an arbitrary valuation v and suppose $F(v)$ is defined. Then by Lemma 6.5

$$F(v) = \begin{cases} [F_1^1 \circ \ldots \circ F_m^1 \circ F_1](v) \text{ if } \big[[F_1^1 \circ \ldots \circ F_m^1 \circ F_1](v)\big](a_s) \\ \qquad\qquad = \big[[F_1 \circ \ldots \circ F_m^1 \circ F_1](v)\big](a_p) \\ [F_1^1 \circ \ldots \circ F_m^1 \circ F_1 \circ F^*](v) \quad \text{in the converse case.} \end{cases}$$

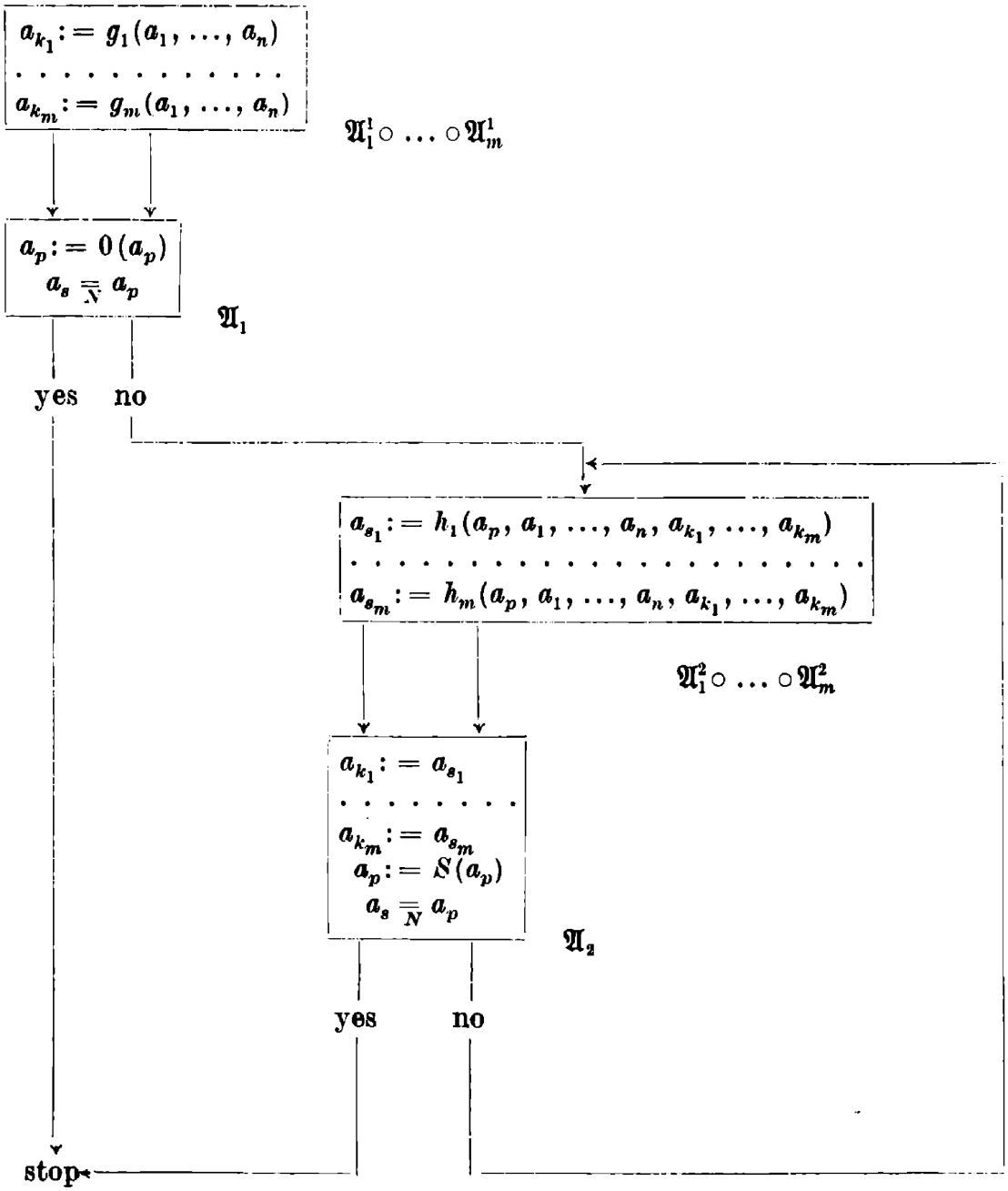

Fig. 7.1

We have assumed a_s as not appearing in any of $\mathfrak{A}_1^1, \ldots, \mathfrak{A}_m^1$, thus

$$\big[[F_1^1 \circ \ldots \circ F_m^1 \circ F_1](v)\big](a_s) = v(a_s).$$

On the other hand

$$\big[[F_1^1 \circ \ldots \circ F_m^1 \circ F_1](v)\big](a_p) = 0.$$

Thus

$$F(v) = \begin{cases} [F_1^1 \circ \ldots \circ F_m^1 \circ F_1](v) & \text{if} \quad v(a_s) = 0, \\ [F_1^1 \circ \ldots \circ F_m^1 \circ F_1 \circ F^*](v) & \text{if} \quad v(a_s) \neq 0. \end{cases}$$

Hence, if $v(a_s) = 0$, then by the assumption 1)

$$(7.1)\qquad \big[[F_1^1 \circ \ldots \circ F_m^1 \circ F_1](v)\big](a_{k_i}) = [F_i^1(v)](a_{k_i}) = g_i\big(v(a_1), \ldots, v(a_s)\big)$$

for every $i = 1, \ldots, m$.

Suppose now $v(a_s) \neq 0$ and let

$$[F_1^1 \circ \ldots \circ F_m^1 \circ F_1](v) = v_1.$$

Hence $F(v) = F^*(v_1)$. Denote

$$F_3 = F_1^2 \circ \ldots \circ F_m^2 \circ F_2.$$

Then, by Lemma 6.3,

$$F^*(v_1) = [F_3]^j(v_1),$$

where j is the least i with

$$\big[[F_3]^i(v_1)\big](a_s) = \big[[F_3]^i(v_1)\big](a_p),$$

thus, by the assumption 2), with

$$\big[[F_2]^i(v_1)\big](a_s) = \big[[F_2]^i(v_1)\big](a_p),$$

i.e. with $v_1(a_s) = i$. Since we have assumed a_s as not appearing in any of $\mathfrak{A}_1^1, \ldots, \mathfrak{A}_m^1$, we have $v_1(a_s) = v(a_s)$, hence

$$F^*(v_1) = [F_3]^{v(a_s)}(v_1).$$

Evidently, for every $i \geqslant 1$,

$$[F_3]^{i+1}(v_1) = F_3\big([F_3]^i(v_1)\big).$$

Thus it is easy to prove by induction on i that for every $i \leqslant v(a_s)$ and every $j \leqslant m$

$$\big[[F_3]^{i+1}(v_1)\big](a_{k_j}) = h_j\big(i, v(a_1), \ldots, v(a_n), v_i(a_{k_1}), \ldots, v_i(a_{k_m})\big)$$

where $v_i = [F_3]^i(v_1)$. Now, using again the scheme of induction, we can prove that for every $1 \leqslant i \leqslant v(a_s)$ and every $j = 1, \ldots, m$

$$\big[[F_3]^i(v_1)\big](a_{k_j}) = f_j\big(i, v(a_1), \ldots, v(a_n)\big),$$

thus in particular

$$[F^*(v_1)](a_{k_j}) = f_j\big(v(a_s), v(a_1), \ldots, v(a_n)\big)$$

but, if $v(a_s) \neq 0$, then $F(v) = F^*(v_1)$, hence, finally, by virtue of (7.1)

$$[F(v)](a_{k_j}) = f_j\big(v(a_s), v(a_1), \ldots, v(a_n)\big)$$

for any natural $v(a_s) \geqslant 0$ and any $j = 1, \ldots, m$. Thus 2° and the left-to-right implication in 1° of Definition 5.1 is proved. By an analogous argument we can prove now the remaining right-to-left implication in 1°, q.e.d.

THEOREM 7.4. *For every normal model M the set* ADF(M) *is closed under the operation of minimum.*

Proof. Consider a normal model M and an arbitrary n-argument function h in ADF(M). Let h be defined in M by

$$\mathfrak{A} = (G, p, q, \Omega, \xi)$$

with an external matrix $a_1, \ldots, a_n; a_k$. Consider now two auxiliary algorithms (the symbols S, 0, and $\underset{N}{=}$ have the same meaning as in the proof of Theorem 7.3):

$$\mathfrak{A}_1 = (G, p, q, \Omega, \xi_1),$$

where

$\xi_1(g) = \xi(g)$ for every g in $G-\{p, q\}$,
$\xi_1(p) = (a_1 := 0(a_1), \alpha; \varrho)$ for $\xi(p) = (\alpha; \varrho)$,
$\xi_1(q) = (\beta; a_k \underset{N}{=} a_1)$ for $\xi(q) = (\beta; \gamma)$,

and the second algorithm

$$\mathfrak{A}_2 = (G, p, q, \Omega, \xi_2)$$

where

$\xi_2(g) = \xi(g)$ for every g in $G-\{p, q\}$,
$\xi_2(p) = (a_1 := S(a_1), \alpha; \varrho)$ for $\xi(p) = (\alpha; \varrho)$,
$\xi_2(q) = (\beta, a_0 := 0(a_0); a_k \underset{N}{=} a_0)$ for $\xi(q) = (\beta; \gamma)$.

Now we shall prove that the algorithm

$$\mathfrak{A}_3 = \mathfrak{A}_1 \uparrow \mathfrak{A}_2$$

defines $f = M(h)$ with the matrix $a_2, \ldots, a_n; a_1$ (cf. Fig. 7.2).

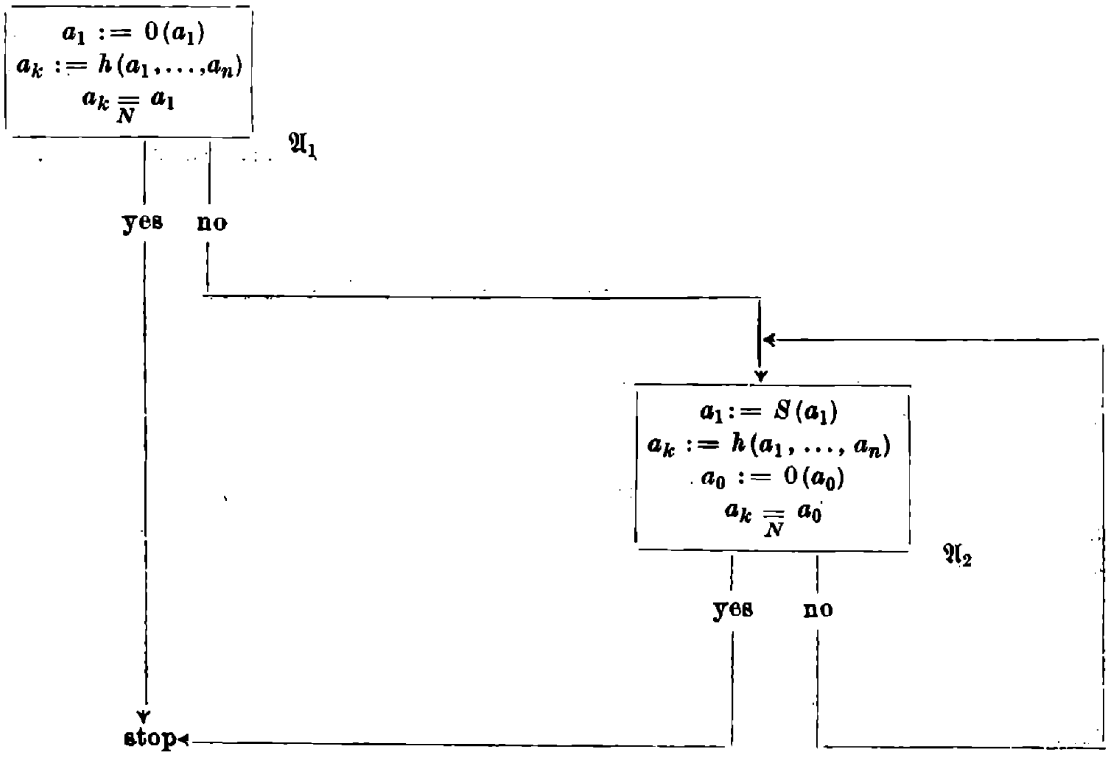

Fig. 7.2

Let F_1, F_2 and F_3 be the output functions of $\mathfrak{A}_1$, $\mathfrak{A}_2$ and $\mathfrak{A}_3$ respectively. Clearly, by Lemma 6.3, if $F_3(v)$ is defined, then

$$F_3(v) = \begin{cases} F_1(v) & \text{if} \quad h\big(0, v(a_2), \ldots, v(a_n)\big) = 0, \\ [F_1 \circ F_2^*](v) & \text{if} \quad h\big(0, v(a_2), \ldots, v(a_n)\big) \neq 0. \end{cases}$$

If $h\big(0, v(a_2), \ldots, v(a_n)\big) = 0$, then clearly

$$[F_3(v)](a_1) = [F_1(v)](a_1) = 0.$$

Suppose thus $h\big(0, v(a_2), \ldots, v(a_n)\big) \neq 0$. Then $[F_1 \circ F_2^*](v)$ is defined and it is easy to prove that $\big[[F_1 \circ F_2^*](v)\big](a_1)$ is the least natural j with $h\big(j, v(a_1), \ldots, v(a_n)\big) = 0$. Thus in fact

$$F_3(v) = f\big(v(a_2), \ldots, v(a_n)\big).$$

Now one can easily prove that the right-to-left implication in 1° of Definition 5.1 is also satisfied, q.e.d.

As it is easy to see, for every model M — not necessarily normal one — every many-argument identity function, i.e. every function

$$I_j^n(x_1, \ldots, x_n) = x_j, \quad j \leqslant n,$$

is in ADF(M). On the other hand, by the definition of natural models and by Theorem 7.1 the successor and the zero-function are in ADF(M) for every normal M. Hence, Theorems 7.2, 7.3 and 7.4 imply immediately the following:

COROLLARY 7.1. *For every normal model M the set* ADF(M) *contains all partial recursive functions.*

8. RELATIONS DEFINABLE IN NORMAL MODELS

In this section we shall deal with the concept of an algorithmically definable relation. This notion is not of a particular interest, however we shall need it in proving other theorems concerning ADF(M).

Consider an arbitrary normal model $M = (X, J)$ and let r be an arbitrary n-argument relation in X. By a *characteristic function of* r we shall mean a n-argument total function f in X with the following property:

$$(\forall x \in X^n)[r(x) \Rightarrow f(x) = 1 \,\&\, \neg r(x) \Rightarrow f(x) = 0].$$

A relation r in X is said to be *algorithmically definable in* M if its characteristic function is algorithmically definable in M. ADR(M) will denote the set of all algorithmically definable relations in M.

THEOREM 8.1. *For every normal model* M, $J(A_R) \subseteq \mathrm{ADR}(M)$.

Proof. Consider an arbitrary normal model $M = (X, J)$ and let $r = J(R_j^n)$ for some R_j^n in A_R. Consider an algorithm

$$\mathfrak{A} = (\{p, g, q\}, p, q, \Omega, \xi),$$

where

$\Omega(p) = (q, g)$,
$\Omega(g) = (q, g)$,
$\xi(p) = \left(a_k := 0(a_k),\ a_k := S(a_k);\ R_m^n(a_1, \ldots, a_n)\right)$, $k > n$,
$\xi(g) = \left(a_k := 0(a_k);\ a_k \underset{N}{=} a_k\right)$,
$\xi(q) = (\varepsilon;\ a_k \underset{N}{=} a_k)$,

where S, 0 and $\underset{N}{=}$ denote respectively the successor, the zero-function and the "natural equivalence" (see Def. 7.1).

Clearly, $\mathfrak{A}$ defines the characteristic function of r with a matrix $a_1, \ldots, a_n; a_k$, q.e.d.

THEOREM 8.2. *For every normal model* M *and every natural* $n \geqslant 1$ *the set* $\mathrm{ADR}^n(M)$ *of all n-argument algorithmically definable relations in* M *contains the empty and the full relations and is closed under the operations of union, intersection and complement (alternative, conjunction and negation).*

Proof. Consider an arbitrary normal model $M = (X, J)$. It is evident, that the empty and the full relations are in $\mathrm{ADR}^n(M)$. Suppose now $r_1, r_2 \in \mathrm{ADR}^n(M)$ and let f_1, f_2 be the characteristic functions of r_1 and r_2 respectively. By Corollary 7.1 the following natural functions are in $\mathrm{ADF}(M)$:

1) the *sum* and the *product* in N;

2) $\mathrm{sg}(x) = \begin{cases} 0 & \text{if} \quad x = 0, \\ 1 & \text{if} \quad x \neq 0; \end{cases}$

3) $\overline{\mathrm{sg}}(x) = \begin{cases} 1 & \text{if} \quad x = 0, \\ 0 & \text{if} \quad x \neq 0. \end{cases}$

It is easy to see now, that $\mathrm{sg}\left(f_1(x)+f_2(x)\right)$, $f_1(x)\cdot f_2(x)$ and $\overline{\mathrm{sg}}\left(f_1(x)\right)$ are the characteristic functions of $r_1 \cup r_2$, $r_1 \cap r_2$ and $X^n - r_1$ respectively. By Theorem 7.2 all these functions are in $\mathrm{ADF}(M)$, q.e.d.

Consider an arbitrary normal model $M = (X, J)$. A two-argument function p in X is said to be a *product-like function*, if for every x in X, $p(x, 0)$ and $p(x, 1)$ are defined and $p(x, 0) = 0$, $p(x, 1) = x$.

A two-argument function s in X is said to be a *sum-like function* if for every x in X, $s(x, 0)$ and $s(0, x)$ are defined and $s(x, 0) = s(0, x) = x$.

LEMMA 8.1. *For every normal model* M *the set* $\mathrm{ADF}(M)$ *contains at least one product-like function and at least one sum-like function.*

Proof. Consider the following recursively defined function

$$p(x, 0) = 0,$$
$$p(x, y+1) = x.$$

Clearly, $p(x, y)$ is a product-like function and by Theorem 7.3 p is in $\mathrm{ADF}(M)$.

Consider now the following function

$$s(x, y) = \begin{cases} x & \text{for } y = 0 \text{ and any } x, \\ y & \text{for } x = 0 \text{ and any } y, \\ \text{not defined} & \text{if } x \neq 0 \text{ and } y \neq 0. \end{cases}$$

Clearly, s is a sum-like function. We shall show that $s \in \mathrm{ADF}(M)$.

Consider the algorithm

$$\mathfrak{A} = (\{p, g, q\}, p, q, \Omega, \xi),$$

where

$\xi(p) = (a_3 := a_1, a_4 := 0(a_4); a_3 \underset{N}{=} a_4)$,
$\Omega(p) = (q, g)$,
$\xi(g) = (a_3 := a_2; a_1 \underset{N}{=} a_4)$,
$\Omega(g) = (q, g)$,
$\xi(q) = (\varepsilon; a_1 \underset{N}{=} a_1)$.

It is easy to see that A defines s with the matrix $a_1, a_2; a_3$, q.e.d.

In Section 7 we have defined three schemes (classes) of operations on functions. Now we shall define a new one:

4) *The operation of conditional definition.* Let $g_1, \ldots, g_m \in \boldsymbol{F}^n$ and let $r_1, \ldots, r_m \in \boldsymbol{R}^n$, where $r_i \cap r_j = \emptyset$ for $i \neq j$ and $i, j = 1, \ldots, m$. By

$$f = C_n^m(g_1, \ldots, g_m, r_1, \ldots, r_m)$$

we mean a function defined in the following way:

$$f(x) = \begin{cases} g_1(x) & \text{if} \quad x \in r_1, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ g_m(x) & \text{if} \quad x \in r_m, \\ \text{undefined} & \text{if} \quad x \notin r_1 \cup \ldots \cup r_m. \end{cases}$$

THEOREM 8.3. *For every normal model M and every $n \geqslant 1$, if $r_1, \ldots, r_m \in \mathrm{ADR}^n(M)$, where $r_i \cap r_j = \emptyset$ for $i \neq j$ and $i, j = 1, \ldots, m$, and $g_1, \ldots, g_m$ are n-argument function in $\mathrm{ADF}(M)$, then the function $f = C_n^m(g_1, \ldots, g_m, r_1, \ldots, r_m)$ is in $\mathrm{ADF}(M)$.*

Proof. Let p and s be functions defined as in the proof of Lemma 8.1. Consider the function

$$h(x_1, \ldots, x_m) = s\big(x_1, s\big(x_2, \ldots, s(x_{m-1}, x_m)\ldots\big)\big).$$

As it is easy to see, for every $i \leqslant m$, if $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m$ are all equale zero, then

$$h(x_1, \ldots, x_m) = x_i.$$

Clearly, by Theorem 7.2, $h \in \mathrm{ADF}(M)$. Let now $f_1, \ldots, f_m$ be characteristic functions of $r_1, \ldots, r_m$ respectively. Then f satisfies the following equality:

$$f(x) = h\big(p\big(g_1(x), f_1(x)\big), \ldots, p\big(g_m(x), f_m(x)\big)\big) \times \\ \times \mathrm{E}[1/h\big(p\big(g_1(x), f_1(x)\big), \ldots, p\big(g_m(x), f_m(x)\big)\big)],$$

where $\mathrm{E}(z)$ is the integral part of z. By Corollary 7.1 the quotient, the product and E are in $\mathrm{ADF}(M)$. Therefore, by Theorem 7.2, f is in $\mathrm{ADF}(M)$ too, q.e.d.

9. THE MAIN THEOREM ON NORMAL MODELS

Definition 9.1. Consider an arbitrary normal model $M = (X, J)$ and let s be the sum-like function as defined in Lemma 8.1.

By the *class of generalized recursive functions* over M, in symbols $\mathrm{GRF}(M)$, we mean the least class of many-argument partial functions in X containing

(i) all functions in $J(A_F)$,

(ii) characteristic functions of all relations in $J(A_R)$,

(iii) all many-argument identity functions,

(iv) the sum-like function s

and closed under the operations of

1° substitution,

2° immediate recursion,

3° minimum.

A relation r in X is said to be a *generalized recursive relation* over M, if the characteristic function of r is in $\mathrm{GRF}(M)$. By $\mathrm{GRR}(M)$ we denote the set of all generalized recursive relations over M.

Lemma 9.1 *For every normal model M the set* $\mathrm{GRF}(M)$ *contains all partial recursive functions, a product-like function and is closed under the operations C_n^m of conditional definition.*

Proof. Since M is normal, $\mathrm{GRF}(M)$ contains the successor and the zero-function, hence — all partial recursive functions. A product-like function is in $\mathrm{GRF}(M)$, since p, as defined in the proof of Lemma 8.1,

is clearly in $\mathrm{GRF}(M)$. Now, since $\mathrm{GRF}(M)$ contains the product-like function p and the sum-like function s, we can prove, by an argument analogous to that in the proof of Theorem 8.3, that $\mathrm{GRF}(M)$ is closed under the operation of conditional definition, q.e.d.

By analogy to the proof of Theorem 8.2 we can easily prove the following:

LEMMA 9.2. *For every normal model M and every $n \geqslant 1$ the set $\mathrm{GRR}^n(M)$, of all the n-argument generalized recursive relations over M, contains the empty and the full relation and is closed under the operation of union, intersection and the complement.*

THEOREM 9.1 (the main). *For every normal model M* $\mathrm{ADF}(M) = \mathrm{GRF}(M)$.

Proof. Consider an arbitrary normal model $M = (X, J)$. By Theorems 7.1, 8.1 and Lemma 8.1, the set $\mathrm{ADF}(M)$ contains the functions (i), (ii) and (iv) specified in Definition 9.1. On the other hand, it is clear that $\mathrm{ADF}(M)$ contains all identity functions, thus by Theorems 7.2, 7.3 and 7.4

$$\mathrm{GRF}(M) \subseteq \mathrm{ADF}(M).$$

We shall prove now the converse inclusion:

$$\mathrm{ADF}(M) \subseteq \mathrm{GRF}(M). \tag{9.1}$$

Consider an arbitrary algorithm $\mathfrak{A} = (G, p, q, \Omega, \xi)$ and let $a_1, \ldots, a_n$ be the only variables in $\mathfrak{A}$ (cf. Lemma 5.2). Suppose $\mathfrak{A}$ defines some function f with a matrix $a_1, \ldots, a_m; a_k$. We shall prove that $f \in \mathrm{GRF}(M)$.

Note first that for any two valuations v^1 and v^2 in V_M, if $v^1 = v^2 \bmod \{a_1, \ldots, a_n\}$ and $\tau^j = \{(g_i^j, v_i^j)\}_{i=0}$ is the trace of v^j in $\mathfrak{A}$ for $j = 1, 2$, then τ^1 and τ^2 are of the same length and

$$v_i^1 = v_i^2 \bmod \{a_1, \ldots, a_n\} \quad \text{for} \quad i = 0, 1, \ldots,$$
$$g_i^1 = g_i^2 \quad \text{for} \quad i = 0, 1, \ldots$$

As it follows therefore, instead of considering valuations, we can consider vectors in X^n and instead of considering transvaluations, we can consider functions in X^n. Thus with the algorithm $\mathfrak{A}$ we can associate $n+1$ parial functions:

$g\colon N \times X^n \to G$,

$w_i\colon N \times X^n \to X$ for $i = 1, \ldots, n$

defined in the following way:

Let $x_1, \ldots, x_n \in X$, let v be an arbitrary valuation with $v(a_i) = x_i$ for $i = 1, \ldots, n$ and let $\{(g_j, v_j)\}_{j=0}$ be the trace of v in $\mathfrak{A}$. Then for

every $j \geqslant 0$

$$(9.2) \qquad g(j, x_1, \ldots, x_n) = \begin{cases} g_j & \text{if} \quad (g_j, v_j) \text{ defined}, \\ \text{undefined} & \text{if} \quad (g_j, v_j) \text{ undefined}; \end{cases}$$

$$(9.3) \qquad w_j(j, x_1, \ldots, x_n) = \begin{cases} v_j(a_i) & \text{if} \quad (g_j, v_j) \text{ defined}, \\ \text{undefined} & \text{if} \quad (g_j, v_j) \text{ undefined} \end{cases}$$

for any $i = 1, \ldots, n$.

Let now

$$l(x_1, \ldots, x_n) = (\mu j)[g(j, x_1, \ldots, x_n) = q]$$

or, which is the same,

$$(9.4) \qquad l(x_1, \ldots, x_n) = (\mu j)[s(g(j, x_1, \ldots, x_n) \dot{-} q, q \dot{-} g(j, x_1, \ldots, x_n)) = 0],$$

where s is the sum-like function and $\dot{-}$ is the symmetrical difference, and let

$$(9.5) \qquad h(x_1, \ldots, x_n) = w_k(l(x_1, \ldots, x_n), x_1, \ldots, x_n).$$

(We recall that a_k is the output variable of the matrix of f in $\mathfrak{A}$.) Evidently, for any $x_1, \ldots, x_n$ in X and any v with $v(a_i) = x_i$ for $i = 1, \ldots, n$, the function $h(x_1, \ldots, x_n)$ is defined iff $F_{\mathfrak{A}}(v)$ is defined, and if $F_{\mathfrak{A}}(v)$ is defined, then

$$(9.6) \qquad [F_{\mathfrak{A}}(v)](a_k) = h(x_1, \ldots, x_n).$$

On the other hand, $\mathfrak{A}$ defines f with a matrix $a_1, \ldots, a_m; a_k$, thus by 1° and 2° of Definition 5.1 we have the following:

For any two vectors $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ in X^n, if $x_i = y_i$ for $i = 1, \ldots, m$, then $h(x_1, \ldots, x_n)$ is defined iff $h(y_1, \ldots, y_n)$ is defined, and if $h(x_1, \ldots, x_n)$ is defined, then $h(x_1, \ldots, x_n) = h(y_1, \ldots, y_n)$.

Therefore, by (9.6), it is now evident, that for every vector $(x_1, \ldots, x_m)$ in X^m the function $f(x_1, \ldots, x_m)$ is defined iff $h(x_1, \ldots, x_m, 0, \ldots, 0)$ is defined, and if $f(x_1, \ldots, x_m)$ is defined then

$$f(x_1, \ldots, x_m) = h(x_1, \ldots, x_m, 0, \ldots, 0).$$

Now, to prove that f is in $\mathrm{GRF}(M)$, it suffices to prove that h is in $\mathrm{GRF}(M)$, thus, by (9.4) and (9.2), that g and w_k are in $\mathrm{GRF}(M)$. We shall show, that $g, w_1, \ldots, w_n$ are all in $\mathrm{GRF}(M)$.

Before we come to this proof, let us consider two functions:

$$L\colon G - \{q\} \to G \quad \text{and} \quad R\colon G - \{q\} \to G$$

with $\Omega(g) = (L(g), R(g))$ for every g in $G - \{q\}$. Clearly, L and R are

partial recursive (since they are finite), thus $L, R \in \mathrm{GRF}(M)$. Consider now an arbitrary vertex g in G and let

$$\xi(g) = (a; \varrho).$$

Since for any two valuations v_1 and v_2 with $v_1 = v_2 \mathrm{mod}\{a_1, \ldots, a_n\}$ the following equalities:

$$\big[[S_M(a)](v_1)\big](a_i) = \big[[S_M(a)](v_2)\big](a_i) \quad \text{for} \quad i = 1, \ldots, n$$

are true and

$$[\Sigma_M(\varrho)](v_1) \Leftrightarrow [\Sigma_M(\varrho)](v_2),$$

we can consider a function s_g and a relation r_g defined as follows: for any $x_1, \ldots, x_n$ and every valuation v with $v(a_i) = x_i$ for $i = 1, \ldots, n$

$$s_g(i, x_1, \ldots, x_n) = \begin{cases} \big[[S_M(a)](v)\big](a_i) & \text{for} \quad i = 1, \ldots, n, \\ \text{undefined} & \text{for} \quad i > n \end{cases}$$

and

$$r_g(x_1, \ldots, x_n) \Leftrightarrow \Sigma_M(\varrho).$$

Clearly, s_g can be composed by means of the functions in $J(A_F)$ and operations of composition and conditional definition. Therefore and by Lemma 9.1

$$s_g \in \mathrm{GRF}(M).$$

By an analogous argument

$$r_g \in \mathrm{GRR}(M)$$

and this is evidently true for every g in G.

Now, since G is clearly in $\mathrm{GRR}(M)$, the function

$$s(g, i, x_1, \ldots, x_n) = \begin{cases} s_g(i, x_1, \ldots, x_n) & \text{if} \quad g \in G, \\ \text{undefined} & \text{if} \quad g \notin G \end{cases}$$

is, by Lemma 9.1, in $\mathrm{GRF}(M)$, and, by Lemma 9.2, the relation

$$r(g, x_1, \ldots, x_n) \Leftrightarrow [g = g_1 \Rightarrow r_{g_1}(x_1, \ldots, x_n)] \,\&\, \ldots \,\&\, [g = g_s \Rightarrow r_{g_s}(x_1, \ldots, x_n)],$$

where $\{g_1, \ldots, g_s\} = G$, is in $\mathrm{GRR}(M)$. Therefore the function

$$Q(g, x_1, \ldots, x_n) = \begin{cases} L(g) & \text{if} \quad g \in G - \{q\} \,\&\, r(g, x_1, \ldots, x_n), \\ R(g) & \text{if} \quad g \in G - \{q\} \,\&\, \neg r(g, x_1, \ldots, x_n), \\ \text{undefined} & \text{if} \quad g \in G - \{q\} \end{cases}$$

is also in GRF(M). Now, by the definition of the trace and (9.2) with (9.3), we have immediately the following:

$$g(0, x_1, \ldots, x_n) = p,$$

$$w_i(0, x_1, \ldots, x_n) = s(p, i, x_1, \ldots, x_n) \quad \text{for} \quad i = 1, \ldots, n$$

and

$$g(j+1, x_1, \ldots, x_n) = Q\big(g(j, x_1, \ldots, x_n), w_1(j, x_1, \ldots, x_n), \ldots \\ \ldots, w_n(j, x_1, \ldots, x_n)\big),$$

$$w_i(j+1, x_1, \ldots, x_n) = s\Big(Q\big(g(j, x_1, \ldots, x_n), w_1(j, x_1, \ldots, x_n), \ldots \\ \ldots, w_n(j, x_1, \ldots, x_n)\big), i, w_1(j, x_1, \ldots, x_n), \ldots, w_n(i, x_1, \ldots, x_n)\Big)$$

for $i = 1, \ldots, n$.

We can easily see now that $g, w_1, \ldots, w_n$ with Q and s satisfy conditions (i)–(iii) in the definition of the operation of immediate recursion. Thus $g, w_1, \ldots, w_n \in \mathrm{GRF}(M)$ and (9.1) is proved, q.e.d.

It may be interesting to note here, that all the results concerning normal models, with the main theorem in particular, remain true for a larger class of models to be defined as follows:

A model $M = (X, J)$ is said to be *semi-normal* if the following conditions are satisfied:

1° $N \subseteq X$,

2° the zero-function and the successor are in ADF(M),

3° there exists in ADR(M) a two-argument relation $\underset{N}{=}$ with the following property:

$$(\forall x, y \in N)[x \underset{N}{=} y \Leftrightarrow x = y]$$

where $=$ is the identity relation in X.

10. CANONICAL ALGORITHMS

In Section 6 five operations on algorithms have been defined, but in proving Theorems 7.2, 7.3 and 7.4 we have used only two of them, namely the simple and the recursive composition. As it follows therefore, these operations suffice to realize algorithmically the schemes of substitution, immediate recursion and minimum. In this section we shall deal more closely with this fact, which seems to be very important and helpful for proving the adequacy of algorithms in normal models. First we introduce two new notions:

By an *elementary algorithm* we mean any algorithm of the form $\mathfrak{A} = (\{0, 1\}, 0, 1, \Omega, \xi)$, where $\Omega(0) = (1, 1)$.

By the *class of canonical algorithms*, abbreviated CA, we mean the least class of algorithms which contains all elementary algorithms and is closed under the operations of simple and recursive composition. Every algorithm in CA will be said to be a *canonical algorithm*.

Let M be an arbitrary model. By $\mathrm{CDF}(M)$, to be read as *the set of all canonically definable functions in* M, we mean the set of all functions definable in M by means of canonical algorithms. If M is normal, then $\mathrm{CDR}(M)$ denotes analogously the set of all *canonically definable relations* in M.

It is now easy to see, that Lemmas 5.2 and 5.3 remain true for canonical algorithms. Hence the proofs of Theorems 7.1–7.4, 8.1 and 8.2 imply immediately the following theorems:

THEOREM 10.1. *For every normal model* $M = (X, J)$ *the set* $\mathrm{CDF}(M)$ *contains* $J(A_F)$ *and is closed under the operations of substitution, immediate recursion and minimum.*

THEOREM 10.2. *For every normal model* $M = (X, J)$ *the set* $\mathrm{CDR}(M)$ *contains* $J(A_R)$ *and for every* $n \geqslant 1$ *the set* $\mathrm{CDR}^n(M)$ *of all n-argument canonically definable relations in M contains the empty and the full relation and is closed under the operations of union, intersection and the complement.*

Now we can easily prove the following

THEOREM 10.3. *For every normal model M the set* $\mathrm{CDF}(M)$ *contains all partial recursive functions, at least one product-like and at least one sum-like function and is closed under the operation of conditional definition.*

Therefore, by an argument analogous to the one in the proof of the main Theorem 9.1, we can show, that every normal model M satisfies the equality $\mathrm{CDF}(M) = \mathrm{GRF}(M)$. In what follows

THEOREM 10.4. *For every normal model* M, $\mathrm{CDF}(M) = \mathrm{ADF}(M)$.

This theorem is of material importance for proving the adequacy of algorithms. In fact, if we have to prove a given function f to be defined by a given algorithm $\mathfrak{A}$, then, in the general case, we have to consider traces of all possible valuations in $\mathfrak{A}$ and by this means prove (or disprove) appropriate properties of the output function $F_{\mathfrak{A}}$. It is clear that this would lead to a long and complicated proof. On the other hand, if $\mathfrak{A}$ is canonical algorithm, then we can decompose it by means of the operations of simple and recursive composition and apply Lemmas 6.2 and 6.5 in proving properties of the output function $F_{\mathfrak{A}}$. The Theorem 10.4 guarantees that if M is normal, then every f in $\mathrm{ADF}(M)$ can be defined by an algorithm which is easy to be proved adequate. This idea is developed with more details in the Section 12.

11. THE ALGEBRA OF ALGORITHMS

Although the simple and the recursive composition have been proved to generate a class of algorithms semantically equivalent (in the sense of Theorem 10.4) with the class of all algorithms, other operations defined in the Section 6 can be interested too, because of the existence of non-normal models and even for normal models because of some applicational reasons (e.g. it is rather difficult to express $\rightarrow$ by $\circ$ and $\uparrow$ in a normal model). Now we shall consider some algebraic properties of these five operations.

Let $\mathscr{A}$ denotes the class of all algorithms. By the *general algebra of algorithms* we shall mean an algebra

$$\boldsymbol{A} = (\mathscr{A}, {}^*, \circ, *, \uparrow, \rightarrow, \sim),$$

where $^*, \circ, *, \uparrow, \rightarrow$ are operations as defined in Section 6 and $\sim$ is an equivalence relation in $\mathscr{A}$ defined in the following way:

Two algorithms $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are said to be *equivalent*, in symbols $\mathfrak{A}_1 \sim \mathfrak{A}_2$, iff

1) for every model M the output function of $\mathfrak{A}_1$ in M is identical with the output function of $\mathfrak{A}_2$ in M,

2) the output formula of $\mathfrak{A}_1$ is identical with the output formula of $\mathfrak{A}_2$.

THEOREM 11.1. *Consider an arbitrary model M and an arbitrary function f in* ADF(M). *For any two algorithms $\mathfrak{A}_1$ and $\mathfrak{A}_2$, if $\mathfrak{A}_1$ defines f in M with a matrix $a_{i_1}, \ldots, a_{i_n}; a_{i_{n+1}}$ and $\mathfrak{A}_1 \sim \mathfrak{A}_2$, then $\mathfrak{A}_2$ defines f in M with the same matrix. Moreover, if $a_{i_1}, \ldots, a_{i_n}$ are external in $\mathfrak{A}_1$, then they are external in $\mathfrak{A}_2$ too.*

The simple proof of this theorem is omitted here.

THEOREM 11.2. *The equivalence relation $\sim$ is a congruence relation in the algebra $\boldsymbol{A}$.*

Proof. Consider arbitrary algorithms $\mathfrak{A}_1, \ldots, \mathfrak{A}_6$ and let $\mathfrak{A}_1 \sim \mathfrak{A}_2$, $\mathfrak{A}_3 \sim \mathfrak{A}_4$, $\mathfrak{A}_5 \sim \mathfrak{A}_6$.

By Lemma 6.2

$$\mathfrak{A}_1 \circ \mathfrak{A}_3 \sim \mathfrak{A}_2 \circ \mathfrak{A}_3 \sim \mathfrak{A}_2 \circ \mathfrak{A}_4,$$

by Lemma 6.3

$$\mathfrak{A}_1^* \sim \mathfrak{A}_2^*,$$

by Lemma 6.4

$$\mathfrak{A}_1 * \mathfrak{A}_3 \sim \mathfrak{A}_2 * \mathfrak{A}_3 \sim \mathfrak{A}_2 * \mathfrak{A}_4,$$

and therefore

$$\mathfrak{A}_1 \uparrow \mathfrak{A}_3 \sim \mathfrak{A}_1 * \mathfrak{A}_3^* \sim \mathfrak{A}_2 * \mathfrak{A}_4^* \sim \mathfrak{A}_2 \uparrow \mathfrak{A}_4,$$

by Lemma 6.6

$$\mathfrak{A}_1 \rightarrow (\mathfrak{A}_3, \mathfrak{A}_5) \sim \mathfrak{A}_2 \rightarrow (\mathfrak{A}_3, \mathfrak{A}_5) \sim \mathfrak{A}_2 \rightarrow (\mathfrak{A}_4, \mathfrak{A}_6);$$ q.e.d.

As has been mentioned in the Section 6 the operation $*$ of complex composition is not associative in $\boldsymbol{A}$. We shall show now that $*$ is not associative in $\boldsymbol{A}/\sim$ as well. To this effect consider a model M, three algorithms $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{A}_3$, and let F_1, F_2, F_3, F^1 and F^2 denote the output functions in M of $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{A}_3$, $(\mathfrak{A}_1 * \mathfrak{A}_2) * \mathfrak{A}_3$ and $\mathfrak{A}_1 * (\mathfrak{A}_2 * \mathfrak{A}_3)$, respectively. Now a model M and a valuation v in V_M should be chosen in such a way, that $F_1(v)$ satisfies the output condition of $\mathfrak{A}_1$ but not of $\mathfrak{A}_2$. In this case

$$F^1(v) = [F_1 \circ F_3](v) \quad \text{and} \quad F^2(v) = F_1(v),$$

hence

$$(A_1 * A_2) * A_3 \nsim A_1 * (A_2 * A_3)$$

if only F_3 is not the identity transvaluation.

THEOREM 11.3. *For every four algorithms* $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{A}_3, \mathfrak{A}_4$ *the following equivalences are true*:

1. $(\mathfrak{A}_1^*)^* \sim \mathfrak{A}_1$,
2. $\mathfrak{A}_1 * \mathfrak{A}_1^* \sim \mathfrak{A}_1^*$,
3. $(\mathfrak{A}_1 * \mathfrak{A}_1) * \mathfrak{A}_2 \sim \mathfrak{A}_1 * (\mathfrak{A}_1 * \mathfrak{A}_2)$,
4. $(\mathfrak{A}_1 \circ \mathfrak{A}_2) * \mathfrak{A}_3 \sim \mathfrak{A}_1 \circ (\mathfrak{A}_2 * \mathfrak{A}_3)$,
5. $\mathfrak{A}_1 \circ [\mathfrak{A}_2 \to (\mathfrak{A}_3, \mathfrak{A}_4)] \sim \mathfrak{A}_1 \circ \mathfrak{A}_2 \to (\mathfrak{A}_3, \mathfrak{A}_4)$,
6. $[\mathfrak{A}_1 \to (\mathfrak{A}_2, \mathfrak{A}_3)] \circ \mathfrak{A}_4 \sim \mathfrak{A}_1 \to (\mathfrak{A}_2 \circ \mathfrak{A}_4, \mathfrak{A}_3 \circ \mathfrak{A}_4)$.

Proof. ad. 1. By the definition of iteration, $(\mathfrak{A}_1^*)^*$ has the same output formula as $\mathfrak{A}_1^*$ (this is the output formula of $\mathfrak{A}_1$). Thus, by Lemma 6.3, $(\mathfrak{A}_1^*)^* \sim \mathfrak{A}_1^*$.

ad. 2. By the definition of iteration and complex composition, the output formula of $\mathfrak{A}_1 * \mathfrak{A}_1^*$ is identical with the output formula of $\mathfrak{A}_1$, hence it is identical with the output formula of $\mathfrak{A}_1^*$. Consider now an arbitrary model M and let F_1, F_1^* and F_2 denote the output functions of $\mathfrak{A}_1, \mathfrak{A}_1^*$ and $\mathfrak{A}_1 * \mathfrak{A}_1^*$ respectively. Let moreover ϱ_1 be the output formula of $\mathfrak{A}_1$ and denote $r = \Sigma_M(\varrho_1)$. Suppose now $F_2(v)$ is defined for some valuation v. Then, by Lemma 6.4, two cases are possible:

1) $F_1(v)$ is defined and $F_1(v) \in r_1$; then $F_2(v) = F_1(v)$,

2) $[F_1 \circ F_1^*](v)$ is defined and $F_1(v) \notin r_1$; then $F_2(v) = [F_1 \circ F_1^*](v)$.

In the first case $F_1^*(v)$ is defined and $F_1^*(v) = F_1(v)$ (cf. Lemma 6.3), hence $F_2(v) = F_1^*(v)$. In the other case $F_1(v)$ and $F_1^*(v)$ are defined and $F_2(v) = F_1^*(v)$, because, if $F_1(v) \notin r_1$, then $F_1^*(F_1(v)) = F_1^*(v)$. It is now easy to prove, that if $F_1^*(v)$ is defined, then $F_2(v)$ is defined, and $F_2(v) = F_1^*(v)$.

ad. 3. Let for some arbitrary model M:

F_1, F_2 denote the output functions of $\mathfrak{A}_1, \mathfrak{A}_2$ respectively,

F_{11} denotes the output function of $\mathfrak{A}_1 * \mathfrak{A}_1$,

F_{12} denotes the output function of $\mathfrak{A}_1 * \mathfrak{A}_2$,

F_3 denotes the output function $(\mathfrak{A}_1 * \mathfrak{A}_1) * \mathfrak{A}_2$,

F_4 denotes the output function of $\mathfrak{A}_1 * (\mathfrak{A}_1 * \mathfrak{A}_2)$.

Let moreover $r_i = \Sigma_M(\varrho_i)$, where ϱ_i is the output formula of $\mathfrak{A}_i$ for $i = 1, 2$. It is easy to see now, that the output formula of both $(\mathfrak{A}_1 * \mathfrak{A}_1) * \mathfrak{A}_2$ and $\mathfrak{A}_1 * (\mathfrak{A}_1 * \mathfrak{A}_2)$ is ϱ_2, thus it remains only to prove the equality $F_3 = F_4$.

Consider an arbitrary valuation v. By Lemma 6.4, $F_3(v)$ is defined iff one of the following conditions is satisfied:

1° $F_{11}(v)$ is defined and $F_{11}(v) \in r_1$,

2° $F_{11}(v)$, $[F_{11} \circ F_2](v)$ are defined and $F_{11}(v) \notin r_1$.

On the other hand, $F_{11}(v)$ is defined iff one of the following cases holds:

(i) $F_1(v)$ is defined and $F_1(v) \in r_1$,

(ii) $F_1(v)$, $[F_1 \circ F_1](v)$ are defined, and $F_1(v) \notin r_1$.

Therefore $F_3(v)$ is defined iff: either 1° & (i) or 1° & (ii) or 2° & (i) or 2° & (ii). It is easy to see now, that 2° & (i) is never true, since in this case $F_{11}(v) = F_1(v)$ and by 2° $F_{11}(v) \notin r_1$, but by (i) $F_{11}(v) \in r_1$. Finally $F_3(v)$ is defined iff:

(11.1) either 1° & (i) or 1° & (ii) or 2° & (ii).

Similarly $F_4(v)$ is defined iff one of the following cases holds:

3° $F_1(v)$ is defined and $F_1(v) \in r_1$,

4° $F_1(v)$, $[F_1 \circ F_{12}](v)$ are defined and $F_1(v) \notin r_1$.

On the other hand, $[F_1 \circ F_{12}](v) = F_{12}(F_1(v))$ is defined iff one of the following cases holds:

(iii) $F_1(F_1(v))$ is defined and $F_1(F_1(v)) \in r_1$,

(iv) $F_1(F_1(v))$, $[F_1 \circ F_2](F_1(v))$ are defined and $F_1(F_1(v)) \notin r_1$.

Finally $F_4(v)$ is defined iff

(11.2) either 3° or 4° & (iii) or 4° & (iv).

Suppose now $F_3(v)$ is defined. We shall show $F_4(v)$ to be also defined and $F_3(v) = F_4(v)$. Following (11.1) there are three cases to be considered.

Let 1° & (i). Then 3°, thus $F_4(v)$ is defined. On the other hand, in this case,

$$F_3(v) = F_{11}(v) = F_1(v) \quad \text{and} \quad F_4(v) = F_1(v).$$

Let 1° & (ii). Then 4° & (iii) since in this case $F_{11}(v) = F_1(F_1(v))$. On the other hand, in this case,

$$F_3(v) = F_{11}(v) = F_1(F_1(v)) \quad \text{and} \quad F_4(v) = [F_1 \circ F_{12}](v) = F_1(F_1(v)).$$

Let 2° & (ii). In this case

$$F_{11}(v) = F_1(F_1(v)) \text{ and } [F_{11} \circ F_2](v) = F_2\big(F_1(F_1(v))\big) = [F_1 \circ F_2](F_1(v)),$$

thus (iv) is satisfied. Hence $F_{12}(F_1(v)) = [F_1 \circ F_{12}](v)$ is defined. Therefore, and by (ii), 4° is also satisfied. Thus $F_4(v)$ is defined. Now

$$F_3(v) = \big|[F_1 \circ F_1] \circ F_2\big|(v) = F_2\big(F_1(F_1(v))\big)$$

and

$$F_4(v) = \big|F_1 \circ [F_1 \circ F_2]\big|(v) = F_2\big(F_1(F_1(v))\big).$$

In this way the first part of this proof is finished. Now we have to prove that if $F_4(v)$ is defined, then $F_3(v)$ is also defined. Following (11.2) three cases are to be considered.

Let 3°. In this case $F_{11}(v)$ is defined and $F_{11}(v) = F_1(v)$, thus 1° & (i) is satisfied.

Let 4° & (iii). Then (ii) is true. Hence, $F_{11}(v)$ is defined and $F_{11}(v) = F_1(F_1(v))$. Therefore, by (ii), $F_{11}(v) \in r_1$, thus 1° is true.

Let 4° & (iv). Then $F_1(v)$, $F_1(F_1(v))$ are defined (hence (ii)) and $F_1(v) \notin r_1$, thus $F_{11}(v)$ is defined and $F_{11}(v) = F_1(F_1(v))$. Therefore, since $[F_1 \circ F_2](F_1(v))$ is defined,

$$[F_1 \circ F_2](F_1(v)) = F_2\big(F_1(F_1(v))\big) = [F_{11} \circ F_2](v).$$

Thus $[F_{11} \circ F_2](v)$ is also defined and 2° & (ii) is satisfied. And this is the end of this damned proof of 3.

ad. 4. Let for some arbitrary model M:

F_i denotes the output function $\mathfrak{A}_i$ for $i = 1, 2, 3$,

F_{12} denotes the output function of $\mathfrak{A}_1 \circ \mathfrak{A}_2$,

F_{23} denotes the output function of $\mathfrak{A}_2 * \mathfrak{A}_3$,

F_4 denotes the output function of $(\mathfrak{A}_1 \circ \mathfrak{A}_2) * \mathfrak{A}_3$,

F_5 denotes the output function of $\mathfrak{A}_1 \circ (\mathfrak{A}_2 * \mathfrak{A}_3)$.

Let moreover ϱ_i be the output formula of $\mathfrak{A}_i$ for $i = 1, 2, 3$ and let $r_i = \Sigma_M(\varrho_i)$ for $i = 1, 2, 3$. It is easy to see, that the output formulas of $(\mathfrak{A}_1 \circ \mathfrak{A}_2) * \mathfrak{A}_3$ and $\mathfrak{A}_1^* \circ (\mathfrak{A}_2 * \mathfrak{A}_3)$ are the same and are equal to ϱ_3. It remains to prove the equality $F_4 = F_5$.

Consider an arbitrary valuation v. By Lemma 6.4, $F_4(v)$ is defined iff one of the following conditions is satisfied:

1° $F_{12}(v) = F_2(F_1(v))$ is defined and $F_{12}(v) \in r_2$,

2° $F_{12}(v)$, $[F_{12} \circ F_3](v)$ are defined and $F_{12}(v) \notin r_2$.

Now, by Lemma 6.2, $F_{12}(v)$ is defined iff:

(i) $F_1(v)$, $F_2(F_1(v))$ are defined.

On the other hand $F_5(v)$ is defined, iff

3° $F_1(v)$, $F_{23}(F_1(v))$ are defined,

and $F_{23}(F_1(v))$ is defined iff one of the following cases holds:

(ii) $F_2(F_1(v))$ is defined and $F_2(F_1(v)) \in r_2$,

(iii) $F_2(F_1(v))$, $[F_2 \circ F_3](F(v))$ are defined and $F_2(F_1(v)) \notin r_2$.

Suppose now $F_4(v)$ is defined. Two cases are to be considered.

Let 1° be true. Then 3° & (ii), thus $F_5(v)$ is defined. On the other hand, in this case,

$$F_4(v) = F_2(F_1(v)) \quad \text{and} \quad F_5(v) = F_{23}(F_1(v)) = F_2(F_1(v)).$$

Let 2° be true. In this case $[F_{12} \circ F_3](v)$ is defined and

$$[F_{12} \circ F_3](v) = F_3(F_{12}(v)) = F_3(F_2(F_1(v))) = [F_2 \circ F_3](F_1(v)),$$

thus 3° & (iii) is satisfied and $F_4(v) = F_5(v)$.

Suppose now $F_5(v)$ is defined. It is easy to see that, if 3° & (ii) is true, then 1° is true and if 3° & (iii) is true, then 2° is true. Thus, if $F_5(v)$ is defined, then also $F_4(v)$ is defined. In this way 4 is proved.

The proofs of the equalities 5 and 6 are analogous; we omitt them, q.e.d.

The equalities listed in Theorem 11.3 permit to prove other ones, e.g.:

7. $\mathfrak{A}_1 \uparrow \mathfrak{A}_1 \sim \mathfrak{A}_1^*$
8. $(\mathfrak{A}_1 \uparrow \mathfrak{A}_1)^* \sim \mathfrak{A}_1^*$
9. $(\mathfrak{A}_1 \circ \mathfrak{A}_2) \uparrow \mathfrak{A}_3 \sim \mathfrak{A}_1 \circ (\mathfrak{A}_2 \uparrow \mathfrak{A}_3)$
10. $(\mathfrak{A}_1 * \mathfrak{A}_1) \uparrow \mathfrak{A}_2 \sim \mathfrak{A}_1 * (\mathfrak{A}_1 \uparrow \mathfrak{A}_2)$
11. $(\mathfrak{A}_1 \circ \mathfrak{A}_2) \uparrow \mathfrak{A}_2 \sim \mathfrak{A}_1 \circ \mathfrak{A}_2^*$
12. $(\mathfrak{A}_1 \circ (\mathfrak{A}_2 * \mathfrak{A}_2)) * \mathfrak{A}_3 \sim \mathfrak{A}_1 \circ (\mathfrak{A}_2 * (\mathfrak{A}_2 * \mathfrak{A}_3))$, etc.

12. AN ABSTRACT PROGRAMMING LANGUAGE

Many of the results discussed in the preceding sections can have, in our opinion, some importance in the programming practice. In this section we give some arguments towards this opinion. We shall construct here an abstract programming language, with the syntax and the semantics formally described, and we shall show how to prove the adequacy of programs in this language.

Consider the alphabets A_V, A_F and A_R as defined in Section 2 and let

$$A_0^1 = \{:=, (,), [,], ', ;, \text{REC}\},$$

$$A^1 = A_V \cup A_F \cup A_R \cup A_0^1.$$

By the *simple programming language* (SPL) we shall mean the least language L over A^1 with the following properties:

1) For every nonempty string $a_1, \ldots, a_n$ of instructions in I and every formula ϱ in Q, the expression $[a_1, \ldots, a_n; \varrho]$, which we call an *elementary program*, is in L.

2) For any π_1 and π_2 in L the expressions $\pi_1\pi_2$ and $[\pi_1 \mathrm{REC} \pi_2]$ are also in L.

The elements of SPL will be called *programs*. Now we introduce two binary operations in SPL defined as follows:

$$\pi_1 \circ \pi_2 = \pi_1\pi_2,$$

$$\pi_1 \uparrow \pi_2 = [\pi_1 \mathrm{REC} \pi_2],$$

for any π_1 and π_2 in SPL. Clearly, these operations are not the operations of the general algebra of algorithms (see Section 11), however, for the sake of simplicity, we use here the same symbols. On the other hand, the algebra (SPL, $\circ$, $\uparrow$) is evidently a homomorphic image of the algebra $(\mathscr{A}, \circ, \uparrow)$.

Now we shall describe a mapping Δ of SPL into the set of algorithms $\mathscr{A}$, called the *structural semantics of* SPL.

1) For every elementary program $\pi = [a_1, \ldots, a_n; \varrho]$ in SPL

$$\Delta(\pi) = (\{0, 1\}, 0, 1, \Omega, \xi),$$

where

$\Omega(0) = (1, 1)$,

$\xi(0) = (a_1, \ldots, a_n; R_1^1(a_i))$, where a_i is the variable on the left of the symbol $:=$ in a_1,

$\xi(1) = (\varepsilon; \varrho)$.

2) For any two programs π_1 and π_2 in SPL

$$\Delta(\pi_1 \circ \pi_2) = \Delta(\pi_1) \circ \Delta(\pi_2),$$

$$\Delta(\pi_1 \uparrow \pi_2) = \Delta(\pi_1) \uparrow \Delta(\pi_2).$$

Now we can consider the following equivalence relation in SPL:

Two programs π_1 and π_2 in SPL are said to be *equivalent*, in symbols $\pi_1 \sim \pi_2$, if $\Delta(\pi_1) \sim \Delta(\pi_2)$.

Consider now the algebra

$$\boldsymbol{A}_{\mathrm{SPL}} = (\mathrm{SPL}, \circ, \uparrow, \sim)$$

called the SPL-*algebra*. Clearly, Δ is an isomorphism of $\boldsymbol{A}_{\mathrm{SPL}}$ onto the algebra (CA, $\circ$, $\uparrow$, $\sim$). Therefore (see Theorem 11.2):

THEOREM 12.1. *The equivalence relation* $\sim$ *is a congruence relation in* $\boldsymbol{A}_{\mathrm{SPL}}$.

Consider an arbitrary model $M = (X, J)$, a function $f\colon X^n \to X$, and a program π in SPL. The program π is said to *define f in M with a matrix $a_{i_1}, \ldots, a_{i_n}; a_{i_{n+1}}$* if $\Delta(\pi)$ defines f in M with the same matrix.

A matrix (variable) is said to be *external* in π if it is external in $\Delta(\pi)$. A function $f\colon X^n \to X$ is said to be SPL-*definable in M*, if there exists a program π in SPL which defines f in M. By $\mathrm{PDF}(M)$ we denote the set of all SPL-definable functions in M.

Clearly, (see Section 10), for every model M,

$$\mathrm{PDF}(M) = \mathrm{CDF}(M) \subseteq \mathrm{ADF}(M).$$

Hence Theorem 10.4 implies

THEOREM 12.2. *For every normal model M,* $\mathrm{PDF}(M) = \mathrm{ADF}(M)$

In what follows, for normal models, SPL is rich enough to define all functions definable by means of programs in the conventional sense.

Now we shall deal with the problem of adequacy of programs in SPL, i.e. with the problem of proving that a given program in SPL defines a given function. We shall start with formulating there simple theorems (the proofs will be omitted).

THEOREM 12.3. *For every model M, every function f in* $\mathrm{PDF}(M)$, *and any two programs π_1 and π_2 in* SPL, *if π_1 defines f in M with a matrix $a_{i_1}, \ldots, a_{i_n}; a_{i_{n+1}}$ and $\pi_1 \sim \pi_2$, then π_2 defines f in M with the same matrix. Moreover, if $a_{i_1}, \ldots, a_{i_n}$ are external in π_1, then they are also external in π_2* (see Theorem 11.1).

THEOREM 12.4. *For any two elementary programs* $[\alpha_1, \ldots, \alpha_n; \varrho]$ *and* $[\beta_1, \ldots, \beta_m; \omega]$ *in* SPL,

$$[\alpha_1, \ldots, \alpha_n; \varrho] \circ [\beta_1, \ldots, \beta_m; \omega] \sim [\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m; \omega].$$

THEOREM 12.5. *For every program π and every elementary program* $\pi_1 = [a_i := a_i; \varrho]$,

$$\pi_1 \circ \pi \sim \pi.$$

As we have mentioned in Section 10, in proving the adequacy of canonical algorithms — thus also in proving the adequacy of SPL-programs — we use the technique of decomposition of algorithms (programs) and apply Lemmas 6.2 and 6.5. This method, although more efficient than that dealing with traces, is still not satisfactory, since it permits to prove only theorems of the form *F is the output function of π* but not of the form *f is defined by π*. Clearly, the knowledge of the output function helps in proving the adequacy of a program, but the step of the proof to be completed in this case may be rather lengthy. (cf. proofs of Theorems 7.2–7.4).

Compare now the Main Theorem 9.1 with Theorem 12.2. As it follows therefrom

$$\mathrm{GRF}(M) = \mathrm{PDF}(M)$$

for every normal model M, i.e. every function SPL-definable in M can be defined by means of elementary functions ((i)–(iv) in Def. 9.1) and the operations of substitution, immediate recursion and minimum. Hence, we can solve our problem by formulating theorems of the form: *if* π_i *defines* f_i *for* $i = 1, \ldots, n$ *and* $f = \boldsymbol{O}_j(f_1, \ldots, f_n)$, *then* π, *defining* f, *is so and so*, where $\boldsymbol{O}_j$ is one of the operations on functions. In fact, such theorems follow immediately from the proofs of Theorems 7.2–7.4.

THEOREM 12.6. *Consider an arbitrary model* $M = (X, J)$, $m+1$ *functions* $g_1, \ldots, g_m \in \boldsymbol{F}^n(X)$, $h \in \boldsymbol{F}^m(X)$ *and* $m+1$ *programs* $\pi_1, \ldots, \pi_{m+1} \in \mathrm{SPL}$.

If the following conditions are satisfied:

1) *For every* $j = 1, \ldots, m$ π_j *defines* g_j *with an external matrix* $a_{i_1}, \ldots, a_{i_n}; a_{k_j}$ *where* a_{k_j} *is external in* $\pi_{j+1}, \ldots, \pi_m$,

2) π_{m+1} *defines* h *with a matrix* $a_{k_1}, \ldots, a_{k_m}; a_s$ *and* $a_{i_1}, \ldots, a_{i_n}$ *are external in* π_{m+1},

then the program $\pi = \pi_1 \circ \ldots \circ \pi_m \circ \pi_{m+1}$ *defines the function* $f = \boldsymbol{S}_n^m(h, g_1, \ldots, g_n)$ *with an external matrix* $a_{i_1}, \ldots, a_{i_n}; a_s$.

THEOREM 12.7. *Consider an arbitrary normal model* $M = (X, J)$, $2m$ *functions* $g_1, \ldots, g_m \in \boldsymbol{F}^n(X)$ *and* $h_1, \ldots, h_m \in \boldsymbol{F}^{n+m+1}(X)$, *and* $2m+2$ *programs* $\pi_1^1, \ldots, \pi_m^1, \pi_1^2, \ldots, \pi_m^2, \pi_1, \pi_2 \in \mathrm{SPL}$.

If the following four conditions are satisfied:

1) π_j^1 *defines* g_j *with an external matrix* $a_{i_1}, \ldots, a_{i_n}; a_{k_j}$ *and* a_{k_j} *is external in* $\pi_{j+1}^1, \ldots, \pi_m^1$, *for* $j = 1, \ldots, m$,

2) π_j^2 *defines* h_j *with an external matrix* $a_p, a_{i_1}, \ldots, a_{i_n}, a_{k_1}, \ldots, a_{k_m}; a_{s_j}$ *and* a_{s_j} *is external in* $\pi_{j+1}^2, \ldots, \pi_m^2$, *for* $j = 1, \ldots, m$,

3) $\pi_1 = [a_p := 0(a_p); a_r \underset{N}{=} a_p]$,

$\pi_2 = [a_{k_1} := a_{s_1}, \ldots, a_{k_m} := a_{s_m}, a_p := S(a_p); a_r \underset{N}{=} a_p]$,

4) $(f_1, \ldots, f_n) = \boldsymbol{R}_n^m(g_1, \ldots, g_m, h_1, \ldots, h_m)$,

then for every $j = 1, \ldots, m$, *the program*

$$\pi = (\pi_1^1 \circ \ldots \circ \pi_m^1 \circ \pi_1) \uparrow (\pi_1^2 \circ \ldots \circ \pi_m^2 \circ \pi_2)$$

defines f_j *with the external matrix* $a_r, a_{i_1}, \ldots, a_{i_n}; a_{k_j}$.

This theorem deals with the general case of the immediate recursion. Since in practice we deal more frequently with the simple recursion ($m = 1$), it seems useful to formulate separately a theorem concerning this particular case.

THEOREM 12.8. *Consider an arbitrary normal model* $M = (X, J)$, *two functions* $g \in \boldsymbol{F}^n(X)$ *and* $h \in \boldsymbol{F}^{n+2}(X)$, *and four programs* $\pi^1, \pi^2, \pi_1, \pi_2 \in$ SPL.

If the following four conditions are satisfied:

1) π^1 *defines* g *with an external matrix* $a_{i_1}, \ldots, a_{i_n}; a_k$,

2) π^2 *defines* h *with a matrix* $a_p, a_{i_1}, \ldots, a_{i_n}; a_k$ *where* $a_p, a_{i_1}, \ldots, a_{i_n}$ *are external in* π^2,

3) $\pi_1 = [a_p := 0(a_p); a_r \overset{=}{N} a_p]$,
$\pi_2 = [a_p := S(a_p); a_r \overset{=}{N} a_p]$,

4) $f = \boldsymbol{R}_1^n(g, h)$,

then the program

$$\pi = (\pi^1 \circ \pi_1) \uparrow (\pi^2 \circ \pi_2),$$

defines f *with the external matrix* $a_r, a_{i_1}, \ldots, a_{i_n}; a_k$.

THEOREM 12.9. *Consider an arbitrary normal model* $M = (X, J)$, *a function* $h \in \boldsymbol{F}^n(X)$ *and five programs* $\pi^1, \pi_1, \pi_2, \pi_3, \pi_4$ *in* SPL.

If the following conditions are satisfied:

1) π^1 *defines* h *with an external matrix* $a_{i_1}, \ldots, a_{i_n}; a_k$,

2) $\pi_1 = [a_{i_1} := 0(a_{i_1}); R_1^1(a_{i_1})]$,
$\pi_2 = [a_k := a_k; a_k \overset{=}{N} a_{i_1}]$,
$\pi_3 = [a_{i_1} := S(a_{i_1}); R_1^1(a_{i_1})]$,
$\pi_4 = [a_p := 0(a_p); a_k \overset{=}{N} a_p]$, *where* $a_p \notin \{a_{i_1}, \ldots, a_{i_n}, a_k\}$,

then the program

$$\pi = (\pi_1 \circ \pi^1 \circ \pi_2) \uparrow (\pi_3 \circ \pi^1 \circ \pi_4)$$

defines the function $f = \boldsymbol{M}(h)$ *with the external matrix* $a_{i_2}, \ldots, a_{i_n}; a_{i_1}$.

Suppose now we have to write a program in *SPL* defining a given function f in a normal model M. First we define f by means of the elementary functions ((i)–(iv) in Def. 9.1) and the operations of substitution, recursion and minimum. (Clearly, if this turns out to be impossible, then, by Theorems 12.2 and 9.1, f is not in PDF(M), i.e. f can not be defined in SPL). Then, following Theorems 12.6–12.9, we construct a program π which defines f and which, by this construction, is proved adequate.

EXAMPLE 12.1. Suppose for simplicity that the basic language admits instructions of the form

$$a_i := A(a_{i_1}, \ldots, a_{i_n})$$

where $A(a_{i_1}, \ldots, a_{i_n})$ is a simple arithmetical expression in the sense of ALGOL. Consider a model $M = (R, J)$, where R is the set of all real numbers and the symbols $+, -, \times, /, =$ are interpreted (by the function J) as the sum, the difference, the product, the quotient and the

identity relation in R. Clearly, M is a normal model — since a_i+1 and 0 are arithmetical expressions in ALGOL — therefore Theorems 12.7–12.9 are applicable in this case.

Consider now the function

$$f(p, x) = \sum_{n=0}^{p} (2x)^n/n!$$

to be SPL-defined. First, we shall define it by means of the composition and recursion. Consider the function $t(p, x)$ defined as follows:

$$t(0, x) = 2x, \quad t(p+1, x) = t(p, x)\cdot[(2x)/(p+2)],$$

i.e.

$$t(0, x) = 2x, \quad t(p+1, x) = h_1\big(p, x, t(p, x)\big),$$

where $h_1(p, x, y) = y\cdot[2x/(p+2)]$. As it is easy to see, $t(p, x)$ $(2x)^{p+1}/(p+1)!$. Now, f can be defined as follows:

$$f(0, x) = 1, \quad f(p+1, x) = f(p, x)+t(p, x),$$

i.e.

$$f(0, x) = 1, \quad f(p+1, x) = h_2\big(p, x, f(p, x)\big),$$

where $h_2(p, x, z) = z+t(p, x)$.

Consider now the following programs:

$\pi^1 = [a_2 := 2\times a_1;\ a_1 = a_1]$,

$\pi^2 = [a_2 := a_3\times(2\times a_1/a_3+2);\ a_1 = a_1]$,

$\pi_1 = [a_3 := 0;\ a_4 = a_3]$,

$\pi_2 = [a_3 := a_3+1;\ a_4 = a_3]$.

Clearly,

π^1 defines $g_1(x) = 2x$ with the external matrix $a_1;\ a_2$,

π^2 defines $h_1(p, x, y)$ with the external matrix $a_3, a_1, a_2;\ a_2$.

Therefore, by Theorem 12.8, the program

$$\pi_{12} = (\pi^1\circ\pi_1)\uparrow(\pi^2\circ\pi_2) \tag{12.1}$$

defines the function $t(p, x)$ with the external matrix $a_4, a_1;\ a_2$.

Consider now another five programs

$\pi^3 = [a_5 := 1;\ a_5 = a_5]$,

$\pi_0^4 = [a_5 := a_5+a_2;\ a_5 = a_5]$,

$\pi_1^4 = [a_5 := a_5;\ a_5 = a_5]$,

$\pi_3 = [a_4 := 0;\ a_6 = a_4]$,

$\pi_4 = [a_4 := a_4+1;\ a_6 = a_4]$.

Clearly,

π^3 defines $g_2(x) = 1$ with the external matrix $a_1;\ a_5$,

π_0^4 defines $s(z, w) = z+w$ with the external matrix $a_5, a_2;\ a_5$,

π_1^4 defines $I(p, x, z) = z$ with the external matrix $a_4, a_1, a_5;\ a_5$.

On the other hand, π_{12} defines not only $t(p, x)$ with the matrix $a_4, a_1; a_2$, but also the function $t_1(p, x, z) = t(p, x)$, for all $z \in R$, with the matrix $a_4, a_1, a_5; a_2$. Therefore, by Theorem 12.6, the program

$$\pi^4 = \pi_1^4 \circ \pi_{12} \circ \pi_0^4 \tag{12.2}$$

defines $h_2(p, x, z) = z + t(p, x)$ with the external matrix $a_4, a_1, a_5; a_5$. Hence, by Theorem 12.8, the program

$$\pi_{34} = (\pi^3 \circ \pi_3) \uparrow (\pi^4 \circ \pi_4) \tag{12.3}$$

defines $f(p, x)$ with the external matrix $a_6, a_1; a_5$. Now, we can write this program explicitly and simplify it by means of Theorems 12.4 and 12.5. First,

$$\pi_{12} \sim \big[[a_2 := 2 \times a_1, a_3 := 0; a_4 = a_3]$$
$$\text{REC } [a_2 := a_2 \times (2 \times a_1/a_3 + 2), a_3 := a_3 + 1; a_4 = a_3]\big].$$

By Theorem 12.5 and (12.2),

$$\pi^4 \sim \pi_{12} \circ \pi_0^4.$$

Therefore, by (12.3) and Theorem 12.1,

$$\pi_{34} \sim (\pi^3 \circ \pi_3) \uparrow (\pi_{12} \circ \pi_0^4 \circ \pi_4)$$

and by Theorem 12.4

$$\pi_{34} \sim \big[[a_5 := 1, a_4 := 0; a_6 = a_4$$
$$\text{REC}\big[[a_2 := 2 \times a_1, a_3 := 0; a_4 = a_3]$$
$$\text{REC}\,[a_2 := a_2 \times (2 \times a_1/a_3 + 2), a_3 := a_3 + 1; a_4 = a_3]\big]$$
$$[a_5 := a_5 + a_2, a_4 := a_4 + 1; a_6 = a_4]\big] = \pi.$$

Theorem 12.3 implies now, that if π_{34}, then also π defines $f(p, x) = \sum_{n=0}^{p} (2x)^n/n!$ with the external matrix $a_6, a_1; a_5$.

13. FINAL REMARKS AND OPEN PROBLEMS

SPL has been thought as an example of a programming language rich enough to define every function definable in a normal model by means of conventional programs and with the property that given a non-algorithmic definition of a function, we now how to proceed effectively to the program producing jointly the proof of its adequacy (semantical correctness). Clearly, the syntactic structure of SPL is very restricted relatively to the structure of real programming languages (this may lead to long programs), but it is also clear that SPL can be easily extended — maintaining its properties — to a more efficient language by

adding new operations, e.g. the complex composition, the conditional composition, etc., or new facilities, e.g. the procedure facility. It should to be emphasized here that not all facilities can be added in this way, since e.g. adding the **go to** facility would essentially change the structure of the language increasing the difficulty of proving adequacy of programs. In general, new operations and facilities can be added to SPL as long as it stays an algebra generated by a set of simple loop-free programs. It may be interesting to note, that we can easily define a subset of ALGOL isomorphic with SPL (or with an extension of it) which is, by Theorem 12.2, rich enough to define all functions definable in ALGOL and, on the other hand, permits to prove easily adequacy of programs.

Another problem of interest concerning the semantics of programming languages is the problem of equivalence of programs. In this paper we have defined a relation $\sim$ of equivalence (Sec. 11) which holds between two programs (flow-algorithms) if they "do" the same in all possible models. This relation is clearly very strong (too strong for many applications) and in practice we would be probably more interested in the relation $\underset{M}{\sim}$ (*M-equivalence*) which holds between two programs if they "do" the same in the model M. Although $\underset{M}{\sim}$ is weaker than $\sim$, it is still too strong to consider the proper fact of interest to the effect that two programs π_1 and π_2 define the same function f. In fact, most frequently we deal with a situation where given a program π defining a function f we want to simplify or to speed up this program preserving the only property that it defines f. Clearly, the fact two programs π_1 and π_2 define the same (given) function f cannot be described by any reasonable equivalence relation. This means that the algebraic approach to the problems of optimalizing programs will probably lead to relatively weak results.

At the end of this section we shall point out some open problems of the theory of algorithmically definable functions, which seem to be of interest for the studies concerning the semantics of programming languages.

PROBLEM 1. Consider an arbitrary model M and a function f in ADF(M). Let $\mathfrak{A}$ defines f in M. $\mathfrak{A}$ is said to *define f unambiguously* if for any two matrices $a_{i_1}, \ldots, a_{i_n}; a_{i_{n+1}}$ and $a_{j_1}, \ldots, a_{j_n}; a_{j_{n+1}}$ of f in $\mathfrak{A}$, $a_{i_1}, \ldots, a_{i_n}$ is a permutation of $a_{j_1}, \ldots, a_{j_n}$ and $a_{i_{n+1}} = a_{j_{n+1}}$. (E.g. in the Example 5.1 the algorithm $\mathfrak{A}$ does not define $f(x, y)$ unambigously).

The problem is, *whether for every model M and every function f in* ADF(M) *there exists an algorithm $\mathfrak{A}$ defining unambiguously f in M?*

PROBLEM 2. Consider an arbitrary model M and a function f in ADF(M). In the class of all algorithms which define f in M we distinguish an algorithm with the least number of variables. Let $mv_M(f)$ denotes the number of variables of this algorithm and let

$$mo_M(f) = mv_M(f) - n,$$

where n is the number of arguments of f. The problem consists in answering the following two questions:

1) *Is it true that for every model M and every natural number m, there exists a function f in* $\mathrm{ADF}(M)$ *with* $mo_M(f) > m$?

2) *Does there exists, for every model M, a function φ with the property that, for every function f in* $\mathrm{ADF}(M)$, $mo_M(f) < \varphi(n)$, *where n is the number of arguments of f*?

PROBLEM 3. *Does there exist a model M with* $\mathrm{CDF}(M) \neq \mathrm{ADF}(M)$?

PROBLEM 4. *Is the elementary theory of the general algebra of algorithms $\mathbf{A}$ finitely axiomatizable, i.e. does there exist a finite set of equivalences in $\mathbf{A}$ which implies all other equivalences in $\mathbf{A}$*?

ACKNOWLEDGEMENTS

The author wishes to thank Professor Z. Pawlak for suggesting the problem and for constant encouragement and help during this work. He also thanks Dr A. Salwicki for suggesting the technique of investigations.

REFERENCES

[1] J. W. de Bakker, *Semantics of programming languages*, to appear.

[2] A. Blikle, *About the meaning of programs*, Bull. Acad. Polon. Sci., Sér. Sci. Math., Astronom. Phys. 18 (1970).

[3] — *Functions definable by means of programs*, ibid.

[4] — *An algebra of flow-algorithms*, ibid.

[5] R. W. Floyd, *Assigning meanings to programs*, in *Mathematical Aspects of Computer Science*, Proc. of Symp. in Applied Mathematics, Vol. 19 (J. T. Schwartz, ed.), pp. 19–32, American Mathematical Society (1967).

[6] L. A. Kaluzhnin, *Algorithmization of mathematical problems*, in *Problems of Cybernetics*, vol. II, pp. 371-391, New York 1961.

[7] J. McCarthy, *A basis for mathematical theory of computation*, in *Computer Programming and Formal Systems*, (P. Braffort and D. Hirschberg, eds.) pp. 33-69, Amsterdam 1963.

[8] — *Towards a mathematical science of computation*, in *Informatic ı Processing* 1962, Proc. IFIP Congress 1962, pp. 21-28, Amsterdam 1963

DISSERTATIONES
MATHEMATICAE
LXXXV (1971)

ERRATA

Page ligne	Au lieu de	Lire
17^3 17^4	$a_{i_1} := a_{j_n}$ $a_{j_{n+1}} := a_{j_{n+1}}$	$a_{i_1} := a_{j_1}$ $a_{j_{n+1}} := a_{i_{n+1}}$