et $B(y)$ est un opérateur linéaire :

$$B(y) = \beta_i^h \eta^i e_h.$$

Les équations (2) deviennent :

$$(41) \qquad \frac{\partial \eta^k}{\delta \xi^h} = \alpha_{th}^k \alpha_{il}^t \alpha^i \beta_s^l \eta^s.$$

Evidemment, l'ensemble de solutions de (1) a une structure d'espace linéaire.

Si l'on suppose de plus que $A(x)$ est lui aussi, linéaire, le système (2) prend la forme :

$$(42) \qquad \frac{\partial \eta^k}{\partial \xi^h} = \alpha_{th}^k \alpha_{il}^t \alpha_r^i \beta_s^l \xi^r \eta^s.$$

Les conditions d'intégrabilité s'écrivent :

$$(43) \qquad \alpha_{il}^t \beta_s^l (\alpha_{th}^k \alpha_u^i - \alpha_{tu}^k \alpha_h^i) = 0.$$

lPour les opérateurs :

$$A(x) = (\alpha_{tu}^k \xi^u) e_k$$

es relations précédentes sont identiquement satisfaites.

D'une manière analogue on peut traiter le cas :

$$F(x, y) = \sum_{s=1}^{m} A_s(x) \cdot B_s(y),$$

avec $A_s(x)$ des opérateurs quelconques et $B_s(x)$ des opérateurs linéaires et aussi les cas où touts les $A_s(x)$, $B_s(y)$ sont linéaires.

Reçu le 24.XII.1977

Faculté de Mathématiques
Université de Jassy
R. S. Roumanie

BIBLIOGRAPHIE

Moisil Grigore C. — *Sur une classe de systèmes d'équations aux dérivées partielles de la Physique Mathématique.* Imprimerie F. Göbl Fils, S.A. Bucureşti, 1931.

Moisil Grigore C. — *Sisteme de ecuaţii cu derivate parţiale şi numere hipercomplexe.* Analele Acad. R.S.R., sect. şt. mat. fiz. chim. Seria A, T. II, memorial 14.

Analele ştiinţifice ale Universităţii „Al. I. Cuza" Iaşi
Tomul XXIV, f. 2, s. I a, 1978

AN ATTEMPT TO A GENERAL THEORY OF ALGORITHMS

BY

C. CAZACU and M. CAZACU

1. Introduction. The algorithms present both structural and functional sides. Structurally, an algorithm is a system of related components. The relations between components determine the running of the algorithm. Functionally, leaving aside its structure, the algorithm may be considered as a function allowing, given a state, to compute a new state and take a decision. The components of an algorithm are in fact algorithms looked functionally. The functional and structural points of view are relative : sometimes we look the algorithms as functional things, other times they are structural things (flow-charts etc.). In Computer Science we meet frequently the above situation (see the concept of subroutine or structured programming).

Starting from these general ideas we introduce the functional and structural algorithms as mathematical notions. Some closure properties and simulation relations between sets of algorithms are emphasized. Finally, the universal algorithms and some of their properties are presented.

2. Preliminaries. Let M and N be sets. A *function* $f: M \to N$ is a subset $f \subset M \times N$ so that

$$\forall m_1 m_2 n_1 n_2 ((m_1, n_1) \in f \wedge (m_2, n_2) \in f \wedge m_1 = m_2 \Rightarrow n_1 = n_2).$$

The sets $\mathrm{Def}(f) = \{m / m \in M \wedge \exists n\, (m, n) \in f\}$ and $\mathrm{Val}(f) = \{n / n \in N \wedge \exists m\, (m, n) \in f\}$ are the *domain* and the *range* of f respectively. The function f is *total* if $\mathrm{Def}(f) = M$, *finite* if $\mathrm{Def}(f)$ is finite and *onto* if $\mathrm{Val}(f) = N$. If there is $n \in N$ so that $f(m) = n$ (that is, $(m, n) \in f$) then f is *defined* in m ($f(m) \downarrow$), otherwise f is *undefined* in m ($f(m) \uparrow$).

An equality $o_1 = o_2$ between two objects which may be undefined, for exemple $f(m) = g(m)$ where f and g are functions, means that both o_1 and o_2 are defined and equal or both o_1 and o_2 are undefined. An object built by total defined means, using, for this, partial defined things, will be considered defined iff all components of the construction are defined. For example, the pair $(a, f(b, g(c, d)))$ where f and g are functions, will be defined iff $a, b, c, d, g(c, d)$ and $f(b, g(c, d))$ are defined.

3. Functional and structural algorithms. Let S be a set of *states* and D a set of *decisions*. A *functional algorithm* on S is a function $A: S \to S \times D$ for which the set $\{d / d \in D \wedge \exists s_1 s_2\, A(s_1) = (s_2, d)\}$ is finite. The algorithm A defines a *transformation function* $A^\tau: S \to S$ and a *decision function* $A^\delta: S \to D$ as follows. If $A(s) \downarrow$ then $A^\tau(s) \downarrow$, $A^\delta(s) \downarrow$ and

$(A^\tau(s), A^\delta(s)) = A(s)$. Otherwise, $A^\tau(s)\downarrow$ and $A^\delta(s)\uparrow$. By definition, the set Val (A^δ) is finite. The elements of this set are the *decisions* of A.

A structural algorithm is a 7-uple

$$A = (X, D, \alpha, S, \mathcal{A}, \beta, x_1)$$

where X is a finite set of *addresses*, D is a set of *decisions*, $\alpha: X \times D \to X$ is a finite function, S is a set of *states*, $\mathcal{A}$ is a set of functional algorithms on S, $\beta: X \to \mathcal{A} \cup D$ is a total function and $x_1 \in X$ is the *initial address*.

The algorithm A will be said to be *structural* on $\mathcal{A}$. The functional algorithms and decisions of the finite set Val (β) are *components* and *decisions* of A. Sometimes we shall use the notation β_x for the component or decision $\beta(x)$. The address x is *algorithmic* or *decisional* according to $\beta_x \in \mathcal{A}$ or $\beta_x \in D$.

The structural algorithm A may be represented by a finite oriented graph $G = (X, U)$ with marked nodes and edges as follows. The set of edges is the set

$$U = \{(x, y) \mid x, y \in X \wedge \exists d\,(d \in D \wedge \alpha(x, d) = y)\}.$$

To every node $x \in X$ is associated β_x. To every edge $(x, y) \in U$ is associated the set of decisions $D(x, y) = \{d \mid d \in D \wedge \alpha(x, d) = y\}$. We observe that if $(x, y_1), (x, y_2) \in U$ and $y_1 \neq y_2$ then $D(x, y_1) \cap D(x, y_2) = \varnothing$. For example, in fig. 1 a structural algorithm is represented by a graph. This algorithm has the addresses $x_1, x_2, \ldots, x_7$, (x_1 is the initial address), the components $A_1, A_2, \ldots, A_5$ and the decisions 2 and 4. We have $\beta(x_1) = A$, $\beta(x_6) = 4$ etc., $\alpha(x_1, 1) = x_2$, $\alpha(x_1, 3)\uparrow$ etc.

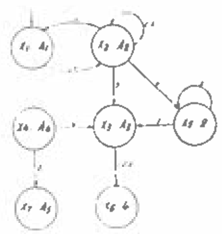


Fig. 1

Roughly speaking, a structural algorithm is a flowchart whose components are functional algorithms or decisions.

For a structural algorithm A in this representation, $(x : o_1, \Delta, y : o_2)$ will denote an edge (x, y) so that $\beta_x = o_1$, $\beta_y = o_2$ and $D(x, y) = \Delta$. For example, for the algorithm of fig. 1 we have $(x_1 : A_1, \{1, 2\}, x_2 : A_2)$, $(x_4 : A_4, \{3\}, x_7 : A_7)$ etc.

Every structural algorithm A defines the following objects:

i) *A set* $\mathcal{C} = S \times X$ *of configurations*

ii) *A function* $A' : \mathcal{C} \to \mathcal{C}$ defined as

$$A'(s, x) = (\beta_x^\tau(s), \alpha(x, \beta_x^\delta(s))),$$

if $\beta_x \in \mathcal{A}$, $\beta_x(s)\downarrow$, $\alpha(x, \beta_x^\delta(s))\downarrow$ and undefined otherwise.

iii) *A set of processes. A process* defined by A is a finite sequence

$$(s_0, y_0), (s_1, y_1), \ldots, (s_n, y_n)$$

of configurations so that $y_0 = x_1$, $A'(s_i, y_i)\downarrow$ and $A'(s_i, y_i) = (s_{i+1}, y_{i+1})$ for $i = 0, 1, \ldots, n-1$ and y_n is a decision address.

iv) *A functional algorithm* $A'' = fa(A)$ defined as follows: If there exists the process

$$(s, x_1), (s_1, y_1), \ldots, (s_n, y_n)$$

defined by A and $\beta(y_n) = d \in D$, then $A''(s) = (s_n, d)$. Otherwise, $A''(s)\uparrow$.

Further on, we suppose all structural or functional algorithms having the decisions in a fixed set D.

Let $\mathcal{A}$ be a set of functional algorithms on S. By $SA(\mathcal{A})$ we denote the set of all structural algorithms on $\mathcal{A}$. The algorithms $A, B \in SA(\mathcal{A})$ are *equivalents* ($A \sim B$) if $fa(A) = fa(B)$. Let $FA(\mathcal{A})$ be the partition of $SA(\mathcal{A})$ generated by this equivalence relation. We identify every class C of this partition with the unique functional algorithm defined by all algorithms of C. Thus, $FA(\mathcal{A})$ is a set of functional algorithms. A structural algorithm A will be considered as *representing* $fa(\mathcal{A})$.

4. Examples. a) *Turing algorithms.* Let $B = \{b_0, b_1, \ldots, b_n\}$ be an unvoid finite set of symbols called *alphabet*, $\ast \notin B$ and let S be the set of states of the form

$$s = \ldots a_2 a_1 a_0 \ast c_0 c_1 c_2 \ldots$$

where $a_i, c_i \in B$ and all but finitely many a_i, c_i are b_0. Let us consider the functional algorithms L, R and T on S defined as follows, for every state s of above form:

$$L(s) = (\ldots a_2 a_1 \ast a_0 c_0 c_1 c_2 \ldots, a_0),$$

$$R(s) = (\ldots a_2 a_1 a_0 c_0 \ast c_1 c_2 \ldots, c_1) \text{ and}$$

$$T(s) = (\ldots a_2 a_1 a_0 \ast c_0' c_1 c_2 \ldots, c_0'),$$

where $b_i' = b_{i+1}$ for $i = 0, 1, 2, \ldots, n-1$ and $b_n' = b_0$.

The algorithms of the set $\mathcal{T} = FA(\{L, R, T\})$ are called *Turing (functional) algorithms*. In the same manner Turing algorithms with k tapes or with left-bounded tape can be defined.

b) *Recursive algorithms.* Let S be the set of states having the form $s = (a_0, a_1, a_2, \ldots)$ where a_i are natural numbers and all but finitely many a_i are 0. Let P_i and M_i $(i = 0, 1, 2 \ldots)$ the algorithms on S defined by

$$P_i(s) = ((a_0, a_1 \ldots a_{i-1}, a_i + 1, a_{i+1} \ldots), 1) \text{ and}$$

$$M_i(s) = ((a_0, a_1, \ldots, a_{i-1}, a_i - 1, a_{i+1} \ldots), sg(a_i \dot{-} 1)),$$

where $0 \dot{-} 1 = 0$, $(a + 1) \dot{-} 1 = a$, $sg(0) = 0$ and $sg(a + 1) = 1$ for all natural numbers a. The *recursive algorithms* are the algorithms of the set $\mathcal{R} = FA(\{P_0, M_0, P_1, M_1, \ldots\})$. These algorithms are useful tools in the theory of effective computability.

c) *Operatorial algorithms.* S is the set of natural numbers of the form $2^a 3^b 5^c$ and M_2, M_3, M_5 and Q are defined as

$$M_i(s) = (s \times i, 1) \text{ for } i = 2, 3, 5 \text{ and}$$

$$Q(s) = \begin{cases} (s/30, 1) & \text{if } 30/s, \\ (s, 0) & \text{otherwise.} \end{cases}$$

The algorithms of $\mathcal{O} = FA(\{M_2, M_3, M_5, Q\})$ are *operatorial* [2].

d) *Markov algorithms.* Let $B = \{b_0, b_1, \dots, b_n\}$ be a finite unvoid alphabet and let $S = B^*$ be the set of all words on B. Let, for all $u, v \in S$, the algorithms $T_{u,v}$, be defined as follows: Let $s \in S$. If u is a subword in s then $T_{u,v}(s) = (s', 1)$, where s' is obtained from s by changing the first left subword u by v. Otherwise, $T_{u,v}(s) = (s, 0)$. The algorithms of the set $\mathcal{M} = FA(\{T_{u,v} \mid u, v \in S\})$ are *Markov algorithms* [1].

5. Closed sets of algorithms. Let us assume that $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are sets of functional algorithms on the same set of states.

Theorem 1. $\mathcal{A} \subset FA(\mathcal{A})$.

Proof. Let $A \in \mathcal{A}$ and $d_1, d_2, \dots, d_n$ are all decisions of A. Then the algorithm B of fig. 2 is in $SA(\mathcal{A})$ and $A = fa(B)$. Therefore $A \in FA(\mathcal{A})$. Q.E.D.

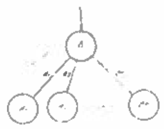


Fig. 2

Theorem 2. *If $\mathcal{B} \subset FA(\mathcal{A})$ then $FA(\mathcal{B}) \subset FA(\mathcal{A})$.*

Proof. Let $A \in FA(\mathcal{B})$.

Therefore it must exist $\tilde{A} \in SA(\mathcal{B})$ so that $A = fa(\tilde{A})$. Let $B_1, B_2, \dots B_n \in \mathcal{B}$ be the components of $\tilde{A}$. Since $\mathcal{B} \subset FA(\mathcal{A})$, there are the structural algorithms $\tilde{B}_i = (X_i, D, \alpha_i, S, \mathcal{A}, \beta_i, x_{i1})$ $(i = 1, 2, \dots, n)$ so that $B_i = fa(\tilde{B}_i)$. Now, let us consider $\tilde{B}_i$ as flowcharts (represented by means of graphs). We suppose that the sets $X_1, X_2, \dots, X_n$ are disjoints.

For every $\tilde{B}_i$ and for every edge $\gamma = (x : C, \Delta, y : d)$ of $\tilde{B}_i$, where C is an algorithm of $\mathcal{A}$ and d is a decision, let us perform the following transformations:

(a) If $\tilde{A}$ has an edge $(u : B_i, \Sigma, v : B_j)$ so that $d \in \Sigma$, we change the edge γ by the edge $(x : C, \Delta, x_{j1} : \beta_j(x_{j1}))$;

(b) if $\tilde{A}$ has an edge $(u : B_i, \Sigma, v : d')$ so that $d \in \Sigma$ and $d' \in D$, we change the decision d in γ by d'.

(c) Otherwise we eliminate the edge γ in $\tilde{B}_i$.

Thus, we obtain a new structural algorithm $\tilde{A}$ on $\mathcal{A}$ with initial address x_{11} and $A = fa(\tilde{A})$. Therefore $A \in FA(\mathcal{A})$. Q.E.D.

Corollary 3.

i) *If $\mathcal{A} \subset \mathcal{B}$ then $FA(\mathcal{A}) \subset FA(\mathcal{B})$.*

ii) *If $\mathcal{B} \subset FA(\mathcal{A})$ and $\mathcal{A} \subset FA(\mathcal{B})$ then $FA(\mathcal{A}) = FA(\mathcal{B})$.*

iii) *$FA(FA(\mathcal{A})) = FA(\mathcal{A})$.*

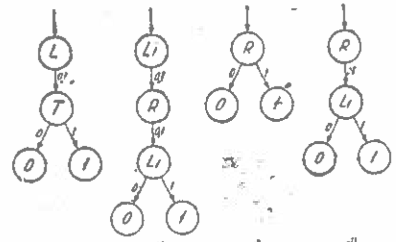
A set $\mathcal{A}$ of functional algorithms is *closed* if $FA(\mathcal{A}) = \mathcal{A}$. By corollary 3, $FA(\mathcal{A})$ is closed. The sets $\mathcal{T}, \mathcal{R}, \mathcal{O}$ and $\mathcal{M}$ considered as examples are closed. A subset $\mathcal{B}$ of the closed set $\mathcal{A}$ is *complete in* $\mathcal{A}$ if $FA(\mathcal{B}) = \mathcal{A}$. A subset $\mathcal{B}$ complete in $\mathcal{A}$ is *base in* $\mathcal{A}$ if there is not a proper subset of $\mathcal{B}$, complete in $\mathcal{A}$.

Corollary 4. *If $\mathcal{B}$ is complete in $\mathcal{A}$, $\mathcal{C} \subset \mathcal{A}$ and $\mathcal{B} \subset FA(\mathcal{C})$ then $\mathcal{C}$ is complete in $\mathcal{A}$ too.*

Proof. By theorem 2 and corollary 3, $FA(\mathcal{B}) \subset FA(\mathcal{C})$.

By hypothesis $\mathcal{A} = FA(\mathcal{B})$ and therefore $\mathcal{A} \subset FA(\mathcal{C})$. The inclusion $\mathcal{C} \subset \mathcal{A}$ implies $FA(\mathcal{C}) \subset FA(\mathcal{A}) = \mathcal{A}$ ($\mathcal{A}$ is closed). Therefore $\mathcal{A} = FA(\mathcal{C})$. Q.E.D.

For example, let $\mathcal{T}$ be with the alphabet $B = \{0,1\}$. The set $\{L, R, T\}$ is a base in $\mathcal{T}$. Now, let L_1 be the functional algorithm represented by the algorithm of fig. 3a). We observe, the algorithms L, R, T are represented by the algorithms of fig. 3b), c), d). These are structural algorithms with components L and R. Therefore the set $\{L_1, R\}$ is complete (base) in $\mathcal{T}$.



a) b) c) d)

Fig. 3

6. The simulation of algorithms. Let S_1 and S_2 be sets. Every *onto* function $\varphi : S_1 \to S_2$ is a *code* of the set S_2 in the set S_1. If $\varphi(s_1) = s_2$ then s_1 is a φ*-code* of s_2. The set $\varphi^{-1}(s_2)$ is the set of all φ-codes for s_2.

Let A and B be functional algorithms on the sets of states S_1 and S_2 respectively and let $\varphi : S_1 \to S_2$ be a code. The algorithm A φ*-simulates* B if for $s_1 \in S_1$ and $\varphi(s_1)\downarrow$ we have $\varphi A^\tau s_1 = B^\tau \varphi s_1$ and $A^\delta s_1 = B^\delta \varphi s_1$. Let us denote this relation by $A \underset{\varphi}{\geq} B$ or by $B \underset{\varphi}{\leq} A$. The algorithm *$A$ simulates B* ($A \geq B$ or $B \leq A$) if there exists a code φ so that $A \underset{\varphi}{\geq} B$.

If $A \underset{\varphi}{\geq} B$ then for every $s_2 \in S_2$,

$$A^\tau \varphi^{-1}(s_2) \subset \varphi^{-1} B^\tau(s_2) \text{ and } A^\delta \varphi^{-1}(s_2) = \{B^\delta(s_2)\}$$

i.e. „A^τ (for every φ-code of s_2) is a φ-code of $B^\tau(s_2)$" and „the decision A^δ (for every φ-code of s_2) is the decision $B^\delta(s_2)$".

Let $\mathcal{A}$ and $\mathcal{B}$ be sets of functional algorithms on S_1 and S_2 respectively and let $\varphi : S_1 \to S_2$ be code. The set $\mathcal{A}$ φ*-simulates* the set $\mathcal{B}$ if

$$\forall B \in \mathcal{B},\ \exists A \in \mathcal{A}\ (A \underset{\varphi}{\geq} B).$$

Let us denote this relation by $\mathcal{A} \underset{\varphi}{\geq} \mathcal{B}$ or by $\mathcal{B} \underset{\varphi}{\leq} \mathcal{A}$. The set $\mathcal{A}$ *simulates* the set $\mathcal{B}$ ($\mathcal{A} \geq \mathcal{B}$) if there is a code φ so that $\mathcal{A} \underset{\varphi}{\geq} \mathcal{B}$.

The relations $\underset{\varphi}{\geq}$ and $\geq$ are reflexive and transitive. If $\mathcal{A} \subseteq \mathcal{B}$ then $\mathcal{A} \leq \mathcal{B}$.

Lemma 5. *The relation $\mathcal{A} \underset{\varphi}{\geq} \mathcal{B}$ induces the code $\bar{\varphi} : \mathcal{A} \to \mathcal{B}$ defined as $\bar{\varphi}(A) = B$ iff $A \underset{\varphi}{\geq} B$.*

Proof. Let S_1, S_2 be the sets of states for $\mathcal{A}, \mathcal{B}$ and let $A \underset{\varphi}{\geq} B_1$, $A \underset{\varphi}{\geq} B_2$ and $\varphi(s_1) = s_2$. Since $B_i^\tau \varphi s_1 = \varphi A^\tau s_1$ and $B_i^\delta \varphi s_1 = A^\delta s_1$ for $i = 1, 2$, it follows $B_1 = B_2$. Therefore $\bar{\varphi}$ is a function. From definition of the relation $\mathcal{A} \underset{\varphi}{\geq} \mathcal{B}$ it follows that $\bar{\varphi}$ is an onto function. Q.E.D.

The sets $\mathcal{A}$ and $\mathcal{B}$ are *similar* ($\mathcal{A} \approx \mathcal{B}$) if $\mathcal{A} \geqslant \mathcal{B}$ and $\mathcal{B} \geqslant \mathcal{A}$.

Theorem 6. If $\mathcal{A} \overset{\varphi}{\geqslant} \mathcal{B}$ *then* $FA(\mathcal{A}) \overset{\varphi}{\geqslant} FA(\mathcal{B})$.

Proof. Let $\mathcal{A}$, $\mathcal{B}$ be on S_1, S_2 respectively and let $B \in FA(\mathcal{B})$. Therefore there is $\tilde{B} \in SA(\mathcal{B})$ so that $B = fa(\tilde{B})$. Let $\tilde{B} = (X, D, \alpha, S_2, \mathcal{B}, \beta, x_1)$ and let $B_1, B_2, \ldots, B_n \in \mathcal{B}$ be the components of $\tilde{\mathcal{B}}$. Since $\mathcal{A} \overset{\varphi}{\geqslant} \mathcal{B}$ it follows that there is $A_1, A_2, \ldots, A_n \in \mathcal{A}$ so that $A_i \overset{\varphi}{\geqslant} B_i$ for $i = 1, 2, \ldots, n$. Now, let us consider the algorithm $\tilde{A} = (X, D, \alpha, S_1, A, B, x_1)$ where, for all $x \in X$, $\bar{\beta}_x = A_i$ if $\beta_x = B_i$ and $\bar{\beta}_x = \beta_x$ if $\beta_x \in D$. Suppose $s_1 \in S_1$, $\varphi(s_1)\downarrow$, $x \in X$ and $\beta_x = B_i$. Then $\tilde{B}'(\varphi s_1, x) = (B_i^\tau \varphi s_1, \alpha(x, B_i^\delta \varphi s_1)) = (\varphi A_i^\tau s_1, \alpha(x, A_i^\delta s_1))$ and $\tilde{A}'(s_1, x) = (A_i^\tau s_1, \alpha(x, A_i^\delta s_1))$. Therefore $\tilde{B}'(\varphi s_1, x) = (\varphi s_1', x')$ iff $\tilde{A}'(s_1, x) = (s_1', x')$. That is, $\tilde{B}$ defines the process $(\varphi s_1, x_1), (\varphi s_2, x_2), \ldots, (\varphi s_n, x_n)$ iff A defines the process $(s_1, x_1), (s_2, x_2), \ldots, (s_n, x_n)$. Hence, $\varphi A^\tau s_1 = B^\tau \varphi s_1$ and $A^\delta s_1 = B^\delta \varphi s_1$ and therefore $A \overset{\varphi}{\geqslant} B$. Q.E.D.

Corollary 7. *Let $\mathcal{A}$ and $\mathcal{B}$ be two sets of algorithms, $\mathcal{A}$ being closed:*

i) *If $\mathcal{A} \geqslant \mathcal{B}$ then $\mathcal{A} \geqslant FA(\mathcal{B})$.*

ii) *If $\mathcal{B}$ is closed and $\mathcal{A}$ simulates a complete set in $\mathcal{B}$, then $\mathcal{A} \geqslant \mathcal{B}$.*

iii) *If $\mathcal{A} \approx \mathcal{B}$ then $\mathcal{B} \approx FA(\mathcal{B})$.*

Using this corollary, we may give simple proofs of the simulation or similarity for different sets of algorithms. For example, let us show that $\mathcal{T} \geqslant \mathcal{O}$. Let S_1 be the set of states for $\mathcal{T}$ (with the alphabet $B = \{0,1\}$) and S_2 the set of states for $\mathcal{O}$. For every state $s_1 = * 01^a 01^b 01^c$ (here 1^n means the string $11\ldots1$ where 1 is n-times repeated) define $\varphi(s_1) = 2^a 3^b 5^c$ and consider $\varphi(s_1)\uparrow$ otherwise. The function $\varphi: S_1 \to S_2$ is a code. To prove that $\mathcal{T} \geqslant \mathcal{O}$, it will be sufficient to prove that the algorithms M_2, M_3, M_5 and Q which form a base in $\mathcal{O}$ are simulated by four algorithms of $\mathcal{T}$. We see, the functional algorithms represented by the algorithms of fig. 4 have these properties.

In an analogous way it is possible to prove other simulations and finally to prove that $\mathcal{T} \approx \mathcal{O} \approx \mathcal{R} \approx \mathcal{M}$ (with appropriate alphabets B for $\mathcal{T}$ and $\mathcal{M}$).

7. Universal algorithms. Let $\mathcal{A}$ be a set of functional algorithms on S and let $\bar{S} = \mathcal{A} \times S \cup S \times D$. (Formally, suppose $\mathcal{A} \times S \cap S \times D = \varnothing$). The algorithm U on $\bar{S}$ is *basic universal algorithm for* $\mathcal{A}$ if $U(A, s) = (A(s), 1)$ for $A \in \mathcal{A}$, $s \in S$ and $U(A, s)\uparrow$ otherwise. The algorithm V is φ*-universal* for $\mathcal{A}$ if $V \overset{\varphi}{\geqslant} U$ and *universal for* $\mathcal{A}$ if $\exists\, \varphi\, (V \overset{\varphi}{\geqslant} U)$.

If V is φ-universal for $\mathcal{A}$, $A \in \mathcal{A}$ and $s \in S$ then „V^τ (for every φ-code of the pair (A, s)) is a φ-code for $A(s)$". If $W \geqslant V$ and V is *universal* for $\mathcal{A}$ then W is universal too for $\mathcal{A}$.

Theorem 8. *If $\mathcal{A} \geqslant \mathcal{B}$ and U, V are basic universal algorithms for $\mathcal{A}$ and $\mathcal{B}$ respectively then $U \geqslant V$.*

Proof. Let S_1 and S_2 be the sets of states for $\mathcal{A}$ and $\mathcal{B}$. Let us consider the code

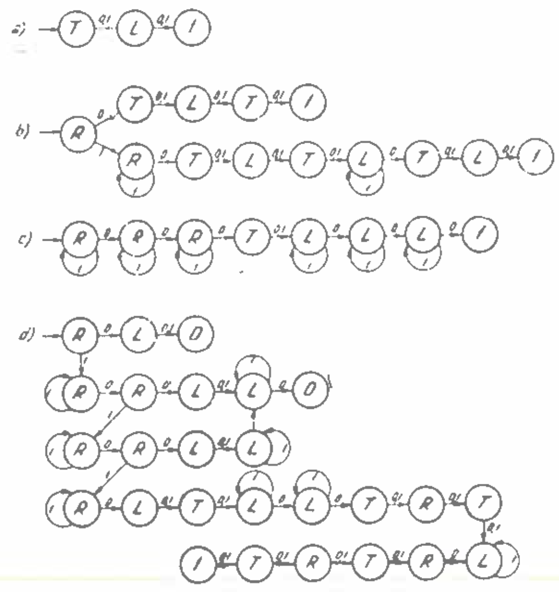


fig. 4

$$\psi : \mathcal{A} \times S_1 \cup S_1 \times D \to \mathcal{B} \times S_2 \cup S_2 \times D$$

defined as

$$\psi(A, s_1) = (\bar{\varphi}(A), \varphi s_1) \quad \text{if } A \in \mathcal{A}, s_1 \in S_1 \text{ and}$$

$$\psi(s_1, d) = (\varphi(s_1), d) \quad \text{if } s_1 \in S_1, d \in D,$$

where $\bar{\varphi}: \mathcal{A} \to \mathcal{B}$ is the code induced by φ (see lemma 5).

Let $(a, b) \in \mathcal{A} \times S_1 \cup S_1 \times D$ and $\psi(a, b)\downarrow$.

a) Suppose $a = A \in \mathcal{A}$ and $b = s_1 \in S_1$. Hence, there is $B \in \mathcal{B}$ and $s_2 \in S_2$, so that $A \overset{\varphi}{\geqslant} B$, $\varphi(s_1) = s_2$ and $\psi(A, s_1) = (B, s_2)$. We obtain $V^\tau \psi(A, s_1) = V^\tau(B, s_2) = B(s_2) = (B^\tau s_2, B^\delta s_2) = (B^\tau \varphi s_1, B^\delta \varphi s_1) = (\varphi A^\tau s_1, A^\delta s_1) = \psi(A^\tau s_1, A^\delta s_1) = \psi A(s_1) = \psi U^\tau(A, s_1)$ and $V^\delta \psi(A, s_1) = V^\delta(B, s_2) = U^\delta(A, s_1)$.

b) Suppose $a = s_1 \in S_1$ and $b = d \in D$. Hence, there is $s_2 \in S_2$, so that $\varphi(s_1) = s_2$ and $\psi(s_1, d) = (s_2, d)$. We obtain $V^\tau \psi(s_1, d) = V^\tau(s_2, d)\uparrow$, $\psi U^\tau(s_1, d)\uparrow$, $V^\delta \psi(s_1, d) = V^\delta(s_2, d)\uparrow$ and $U^\delta(s_1, d)\uparrow$.

Hence, for all $(a, b) \in \mathcal{A} \times S_1 \cup S_1 \times D$, if $\psi(a, b)\downarrow$ then $V^\tau \psi(a, b) = \psi U^\tau(a, b)$ and $V^\delta \psi(a, b) = U^\delta(a, b)$, i.e. $U \geqslant V$. Q.E.D.

Corollary 9. *If V is universal for $\mathcal{A}$ and $\mathcal{A} \overset{\varphi}{\geqslant} \mathcal{B}$ then V is universal for $\mathcal{B}$.*

Proof. Let U_1 and U_2 be basic universal algorithms for $\mathcal{A}$ and $\mathcal{B}$. By theorem 8, $U_1 \geqslant U_2$. But $V \geqslant U_1$ and therefore $V \geqslant U_2$. Q.E.D.

The set $\mathcal{A}$ is *universal* if there is an universal algorithm for $\mathcal{A}$ contained in $\mathcal{A}$.

Corollary 10. *If $\mathcal{A}$ is universal and $\mathcal{A} \approx \mathcal{B}$ then $\mathcal{B}$ is universal.*

Proof. Suppose U_1, U_2 be basic universal for $\mathcal{A}$, $\mathcal{B}$, and let V_1 be universal for $\mathcal{A}$ and $V_1 \in \mathcal{A}$. We have $V_1 \geqslant U_1$ and since $\mathcal{A} \geqslant \mathcal{B}$ by theorem 8 it follows $U_1 \geqslant U_2$. Therefore $V_1 \geqslant U_2$. On the other hand, since $\mathcal{B} \geqslant \mathcal{A}$ and $V_1 \in \mathcal{A}$ it must exist $V_2 \in \mathcal{B}$ so that $V_2 \geqslant V_1$. So, $V_2 \geqslant U_2$ and hence V_2 is universal for $\mathcal{B}$. Q.E.D.

Using some known constructions one can show that $\mathcal{T}$ is universal. Hence, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{M}$ and all sets similar with $\mathcal{T}$ are universal too.

REFERENCES

1. Brainerd W. S., Landweber L. H. — *Theory of Computation*. John Wiley and Sons, New York, 1974.
2. Mal'cev A. I. — *Algorithms and Recursive Functions*.(Russian). Nauka, Moskwa, 1965.

Received 16.II.1978

Faculty of Mathematics
University of Iaşi
Iaşi, R. S. Romania

Analele ştiinţifice ale Universităţii „Al. I. Cuza" Iaşi
Tomul XXIV, f. 2, s. I a, 1978

ON SET FUNCTIONS OF BOUNDED p-VARIATION

BY

VASILE POSTOLICĂ

The concept of p-variation of a real function defined on a real compact interval was introduced by F. Riesz and it has been used to characterize the primitives of the p-absolutely integrable functions ([6], p. 75). Thus, there is a strong connection between the space of functions of bounded p-variation and the space of almost everywhere derivable functions with p-absolutely integrable derivative, space of functions with applications in the theory of distributions [1].

On the other hand, the p-variation leads naturally to the p-absolute continuity ; but as shown by the investigations of C. Upton [8] and N. Gheorghiu, T. Precupanu [2], for $p > 1$ these functions coincide with the functions of bounded p-variation.

In this paper we extend the notion of p-variation to set functions, studying in the same time the connection between the set functions of bounded p-variation and the p-absolutely continuous set functions. To this end, we establish an integral representation of the set functions of bounded p-variation [Theorem 1.6].

1. Set function of bounded p-variation. Let $\mathcal{F}$ be a σ-algebra over a space X, μ a finite measure defined on $\mathcal{F}$ and $F : \mathcal{F} \to \overline{R}$ a set function with $F(\varnothing) = 0$. We consider a non-void set $A \in \mathcal{F}$ and we denote the σ-algebra induced by $\mathcal{F}$ on A by $\mathcal{F}_A$. Throughout the paper we make the following calculation conventions:

$$0\,\infty = \infty\,0 = 0, \quad 0^0 = 1, \quad \frac{a}{0} = \infty \text{ for every } a > 0. \tag{1.1}$$

Definition 1.1. *We shall call $\mathcal{F}$-partition of the set A any finite family $(B_i)_{i=\overline{1,m}}$ of disjoint sets $B_i \in \mathcal{F}$ which covers A i. e., $\bigcup_{i=1}^{m} B_i = A$.*

For $p \geqslant 1$ we define the *p-variation of F on A with respect to a $\mathcal{F}$-partition* $\Delta = (A_i)_{i=\overline{1,n}}$ by

$$\sigma_p(F, \Delta) = \left\{ \sum_{i=1}^{n} \frac{|F(A_i)|^p}{[\mu(A_i)]^{p-1}} \right\}^{1/p} \tag{1.2}$$

and we denote