KYBERNETIKA — VOLUME 14 (1978), NUMBER 4

Conditional Programmed Automata

JAN MAREŠ

A conditional programmed automaton and functions computable on it are defined and, utilizing certain graphs, connections between those automata and functions are studied.

DENOTATIONS

N denotes the set of all positive integers,

$$N_0 = N \cup \{0\} ;$$

for $n \in N_0$ denote

$$\hat{n} = \{1, 2, \ldots, n\} \quad (\text{thus, } \hat{0} = \emptyset) .$$

If A, B, C are sets, write

$$C = A \mathbin{\dot{\cup}} B \quad \text{if} \quad (C = A \cup B \text{ et } A \cap B = \emptyset) .$$

Let $f : A \to B$ denote that f is a mapping of A into B. A mapping g of a nonempty set $A_0 \subset A$ into a set B is called a mapping *from* A into B and denoted by $g : A \nrightarrow B$. $\operatorname{Dom} f$ and $\operatorname{Ran} f$ denote the domain and the range of f, respectively.

Let $n \in N$, let $A_1, \ldots, A_n$ be nonempty sets and let

$$A = A_1 \times \ldots \times A_n ;$$

then for $i \in \hat{n}$ define *projections* $\pi_i^A : A \to A_i$ by a prescription

$$\pi_i^A(a_1, \ldots, a_n) = a_i .$$

(The superscript A will be deleted, if possible.)

1. INTRODUCTION

Mathematical systems aspiring after to model real computers are frequently investigated in the literature. We have in mind e.g. Pawlak's machines [9], called also iterative systems in [1] and examined by many other authors. Further, Čulík's notions of sequential and jumping machines (see [2], [3], [4], [5]) ought to be mentioned here. In fact, the notion of a conditional programmed automaton, dealt with in this article, took its origin in [2].

Our approach is nearer to the latter conception than to the former one. Some our notions correspond to some extent to certain concepts e.g. in [8] or in [10], some other (e.g. a piece-wise computability, an operational tree) seem to be new. Nevertheless, all these concepts proved to be useful for our purposes.

A *conditional programmed automaton* (CPA) is a system

$$\mathscr{A} = (I, S, L, \delta),$$

where I is a nonempty set – the *set of instructions* of $\mathscr{A}$,

$$I = I_S \cup I_L \cup \{!\};$$

here it is $I_S \neq \emptyset$ and finite, $!$ is a *stop-instruction*; S is a nonempty set – the *set of internal states* of $\mathscr{A}$, L is a set – the *set of labels of* $\mathscr{A}$, $S \cap L = \emptyset$ and δ is a mapping from the set $I \times S$ into the set $S \cup L$ such that

$$\delta = \delta_S \cup \delta_L,$$

where

$$\delta_S : (I_S \times S) \mapsto S, \quad \delta_L : (I_L \times S) \mapsto L.$$

A CPA is a model of a computer in the following sense:

The set S represents the set of all possible occupations of the memory of the computer in question by *data* only; programs are understood as "external". This distinguishes our approach from the one of [9].

The set I is a set of instructions; there are distinguished operational from jumping ones – cf. the notions: "a jumping machine", "a pure jump" and "a pure command" in [3] in this connection. Operational instructions correspond to elementary actions, jumping instructions represent elementary tests and next-command decisions of the computer. The set of labels L serves to labelling instructions and the mapping δ specifies the activity of instructions.

Note that our approach is purely abstract, i.e. we have in mind abstract automata and operational instructions and hence the computed functions are simply mappings from S into S without any further structure.

The instructions from the set I_S or I_L are called *operational* or *jumping*, respectively. Each element $c \in (L \times I) \cup I$ is called a *command* (cf. [4]). A command

$c \in L \times I$ or $c \in (L \times I_S) \cup I_S$ or $c \in (L \times I_L) \cup I_L$ is called *labelled* or *operational* or *jumping*, respectively.

Denote $C_{\mathscr{A}} = (L \times I) \cup I$.

Let $c \in C_{\mathscr{A}}$ be a command. If it is $c = \langle z, x \rangle$, where $z \in L$, $x \in I$, set

$$\lambda(c) = z\,, \quad \iota(c) = x\,;$$

for $c = x$ set

$$\iota(c) = x \;(\lambda(c) \text{ is undefined})\,.$$

A *program* is a finite sequence of commands from $C_{\mathscr{A}}$, $\mathscr{P} = [c^1, \ldots, c^p]$ such that the following conditions hold:

(1) $\forall\, i, j \in \hat{p}(\lambda(c^i) = \lambda(c^j) \Rightarrow i = j)$,

(2) $\exists\, i \in \hat{p}(\iota(c^i) = \,!)$.

The set of all programs is denoted by $C^+_{\mathscr{A}}$. Cf. [2], [4], [10].

If for some CPA $\mathscr{A} = (I, S, L, \delta)$ a program $\mathscr{P} \in C^+_{\mathscr{A}}$, $\mathscr{P} = [c^1, \ldots, c^p]$ and a state $s \in S$ are given, then the CPA $\mathscr{A}$ provides the *computation* (by the program $\mathscr{P}$ from the state s) by the following prescription (cf. [4]): Set $s_0 = s$, $c_1 = c^1$, $\alpha_1 = 1$.

Let $i \geqq 1$ and let the state s_{i-1}, the integer α_i and the command $c_i = c^{\alpha_i}$ be determined. Then $\mathscr{A}$ proceeds as follows.

If it is $\iota(c_i) = \,!$, the computation finishes at the i-th step.

If $\iota(c_i) \neq \,!$, the computation finishes at the i-th step provided that $(\iota(c_i), s_{i-1}) \notin$ $\notin \operatorname{Dom} \delta$; otherwise two possibilities are to be distinguished:

(1) $\iota(c_i) \in I_S$; then for $\alpha_i = p$ the computation finishes at the i-th step; whether $\alpha_i < p$, set

$$s_i = \delta(\iota(c_i), s_{i-1})\,, \quad \alpha_{i+1} = \alpha_i + 1\,, \quad c_{i+1} = c^{\alpha_i + 1}\,.$$

(2) $\iota(c_i) \in I_L$; then set

$$z = \delta(\iota(c_i), s_{i-1})\,;$$

if there is no $j \in \hat{p}$ such that $z = \lambda(c^j)$, the computation finishes at the i-th step; whether such a (uniquely determined) j exists, set

$$s_i = s_{i-1}\,, \quad \alpha_{i+1} = j\,, \quad c_{i+1} = c^j\,.$$

A finite or infinite (just now defined) sequence $c_1, c_2, \ldots$ (which is finite and has m members if and only if the computation finishes at the m-th step) is called a *branch* of $\mathscr{P}$ from s and is denoted by $\beta(\mathscr{P}, s)$. Cf. [3], [4].

Denote $\mathfrak{F}_S$ the set of all mappings from S into S.

We say that a program $\mathscr{P} \in C^+_{\mathscr{A}}$ *computes* on the CPA $\mathscr{A}$ a function $f \in \mathfrak{F}_S$ defined as follows: if $\beta(\mathscr{P}, s) = c_1, \ldots, c_m$ and $\iota(c_m) = \,!$, then $s \in \operatorname{Dom} f$ and $f(s) = s_{m-1}$; otherwise $s \notin \operatorname{Dom} f$.

A function $f \in \mathfrak{F}_S$ is called *computable* on the CPA $\mathscr{A}$ if there is a program $\mathscr{P} \in C^+_{\mathscr{A}}$ computing f on $\mathscr{A}$.

Let $\mathscr{P} \in C^+_{\mathscr{A}}$, $s \in S$ and let a sequence $c_{i_1}, c_{i_2}, \ldots$ arise from the (finite or infinite) sequence $\beta(\mathscr{P}, s)$ by deleting all the commands which are not operational. Then the sequence of operational instructions

$$\omega(\mathscr{P}, s) = \iota(c_{i_1}), \iota(c_{i_2}), \ldots$$

is called an *operational branch* of $\mathscr{P}$ from s.

Thus $\omega(\mathscr{P}, s)$ is the sequence of operational instructions which are applied to s; e.g. if $\omega(\mathscr{P}, s) = x_1 \ldots x_m$, then for the function f computed by $\mathscr{P}$ the equality

$$f(s) = f_m(f_{m-1} \ldots (f_1(s)) \ldots)$$

holds, where $f_j = \delta(\cdot\,, x_j)$ for $j \in \hat{m}$.

Example 1. Let $\mathscr{A}_0 = (I_0, S_0, L_0, \delta_0)$ be a CPA such that

$$I_0 = \{x_1, x_2, y_1, y_2\} \cup \{y^{(i)} \mid i \in N_0\}\,,$$
$$S_0 \subset N_0\,, \quad S_0 = \{0, 1, 2, 3, 4, 5, 8, 10, 16\}\,, \quad L_0 = N_0\,,$$

x_1, x_2 and $y_1, y_2, y^{(i)}$ are operational and jumping instructions, respectively and

$$\delta_0(x_1, s) = 3s + 1 \quad \text{for } s \in S_0 \text{ and } s \text{ odd, undefined else}\,,$$
$$\delta_0(x_2, s) = \tfrac{1}{2}s \quad \text{for } s \in S_0 \text{ and } s \text{ even, undefined else}\,,$$

$$\delta_0(y_1, s) = \begin{cases} 1 & \text{for } s \text{ odd}\,, \\ 2 & \text{for } s \text{ even}\,, \end{cases}$$

$$\delta_0(y_2, s) = \begin{cases} 3 & \text{for } s = 1\,, \\ 4 & \text{else} \end{cases}$$

and finally for each $i \in L_0$

$$\delta_0(y^{(i)}, s) = i \quad \text{for all } s \in S_0\,.$$

Let $\mathscr{P}_0 \in C^+_{\mathscr{A}_0}$ be a program,

$$\mathscr{P}_0 = [\langle 0, y_2\rangle, \langle 3, !\rangle, \langle 4, y_1\rangle, \langle 1, x_1\rangle, \langle 2, x_2\rangle, \langle 5, y^{(0)}\rangle]\,.$$

It is easy to see that the program $\mathscr{P}_0$ computes on the CPA $\mathscr{A}_0$ the function $f_0 \in \mathfrak{F}_{S_0}$ defined as follows:

$$f_0(s) = 1 \quad \text{for } s \neq 0, f_0(0) \text{ undefined}\,.$$

Further, e.g.

$$\beta(\mathscr{P}_0, 2) = \langle 0, y_2\rangle, \langle 4, y_1\rangle, \langle 2, x_2\rangle, \langle 5, y^{(0)}\rangle, \langle 0, y_2\rangle, \langle 3, !\rangle\,,$$
$$\omega(\mathscr{P}_0, 3) = x_1x_2x_1x_2x_2x_2x_2\,,$$
$$\omega(\mathscr{P}_0, 2) = x_2\,.$$

A *sequential programmed automaton* (SPA) (cf. [2]) is a system $A = (I, S, \delta)$, where I is a finite nonempty set — the *set of instructions of* A, S is a nonempty set — the *set of internal states* of A and δ is a mapping from the set $I \times S$ into the set S.

Denote δ^* the well known extension of the function δ to the mapping from $I^* \times S$ into S. (I^* denotes the set of all strings over I.) An arbitrary string $\boldsymbol{P} \in I^*$ is called a *sequential program*.

We say that a sequential program $\boldsymbol{P} \in I^*$ *computes* on the SPA A a function $f \in \mathfrak{F}_S$ defined by $f = \delta^*(\boldsymbol{P}, \cdot)$.

A function $f \in \mathfrak{F}_S$ is called *computable* on the SPA A if there is a sequential program $\boldsymbol{P} \in I^*$ computing f on A.

The SPA and functions computable on them are dealt with in [7].

3. THE SEQUENTIAL KERNEL OF A CPA

Let $\mathscr{A} = (I, S, L, \delta)$ be a CPA. Then the SPA Ker $\mathscr{A} = (I_S, S, \delta_S)$ is called a *sequential kernel* of $\mathscr{A}$. Cf. [3].

Let $f \in \mathfrak{F}_S$. If the function f is computed on the SPA Ker $\mathscr{A}$ by a sequential program $\boldsymbol{P} = x_1 \dots x_m$, then f is computed on the CPA $\mathscr{A}$ by the program $\mathscr{P} = [x_1, \dots, x_m, !]$.

The opposite statement in general does not hold: the function f_0 (see Example 1) is obviously not computable on the SPA Ker $\mathscr{A}_0$.

Denote

$$I_L^{\mathrm{TOT}} = \{y \in I_L \mid \mathrm{Dom}\, \delta(y, \cdot) = S \quad \text{et} \quad \mathrm{card}\, \mathrm{Ran}\, \delta(y, \cdot) \in \boldsymbol{N}\}\ .$$

A program $\mathscr{P} \in C_{\mathscr{A}}^+$, $\mathscr{P} = [c^1, \dots, c^p]$ is called *regular* if the following conditions hold:

(1) $\forall\, i \in \hat{p}(\iota(c^i) \in I_L \Rightarrow \iota(c^i) \in I_L^{\mathrm{TOT}})$.

(2) If T is the set of all $s \in S$ for which the branch $\beta(\mathscr{P}, s)$ is finite, then

$$\forall\, s \in T(\iota(c_s) = \,! \quad \text{vel} \quad (\omega(\mathscr{P}, s), s) \notin \mathrm{Dom}\, \delta^*)\,,$$

where c_s denotes the last command of the branch $\beta(\mathscr{P}, s)$.

Thus, by condition (1), in regular programs only "total" jumping instructions may be used; moreover, each of them only finitely many jumps can realize. The condition (2) is purely technical — it prevents obvious syntactic errors in the program.

It is easy to prove the following lemma.

Lemma. A function $f \in \mathfrak{F}_S$ is computable on the SPA Ker $\mathscr{A}$ if and only if there is a regular program $\mathscr{P} \in C_{\mathscr{A}}^+$ computing f on the CPA $\mathscr{A}$ such that the following two conditions are valid:

(1) For each $s \in S$ the branch $\beta(\mathscr{P}, s)$ is finite.

(2) $\forall s_1, s_2 \in S(\omega(\mathscr{P}, s_1) = \omega(\mathscr{P}, s_2))$.

4. THE PIECE-WISE COMPUTABILITY

In this section we essentially generalize the notion of computability on a SPA in order to simulate computation on a CPA. Being a function $f : S \leftrightarrow S$ given, it is possible that f is computed on a SPA "piece-wise", i.e.: to different states correspond generally also different sequential programs, which, being applied to the states, yield the required results. This informal idea is made more precise in the following definition. Note that the defined notion is slightly more general because an auxiliary set S' is introduced such that the states from the set $S - S'$ are "neglected". Cf. [3] in this connection.

Let $A = (I, S, \delta)$ be a SPA.

Let $M \subset S$ be a nonempty set and let Δ be a partition of the set M, i.e.

$$\Delta = \{M_j \mid j \in J\},$$

where $J \neq \emptyset$ is a (finite or infinite) index set,

$$M = \dot{\bigcup_{j\in J}} M_j$$

and $M_j \neq \emptyset$ for each $j \in J$.

We shall write $\Delta = \mid_{j\in J} M_j$, too. The sets M_j are called *classes* of Δ.

Denote $\mathfrak{P}_M$ the set of all partitions of M.

If the index set J is finite, we call the partition Δ *finite*. The set of all finite partitions of M denote $\mathfrak{P}_M^{\mathrm{FIN}}$.

A function $f \in \mathfrak{F}_S$ is said to be *piece-wise computable* on the SPA A if there are a set S' such that $\mathrm{Dom} f \subset S' \subset S$ and a partition $\Delta = \mid_{i\in J} S_i$ of the set S' such that for each $i \in J$ a function $f_i \in \mathfrak{F}_S$ is on A computable such that

$$\mathrm{Dom} f_i \cap S_i = \mathrm{Dom} f \cap S_i$$

and

$$f_i(s) = f(s) \quad \text{for} \quad s \in \mathrm{Dom} f \cap S_i .$$

If the function f_i is on A computed by a sequential program $\boldsymbol{P}_i \in I^*$, then the set

$$\varrho = \{(\boldsymbol{P}_i, S_i) \mid i \in J\}$$

is said to be a *partition of the function* f regarding the SPA A (and the set S').

Example 2. The function f_0 has regarding the SPA Ker $\mathscr{A}_0$ and the set $S_0' =$

$= S_0 - \{0\}$ the partition

$$\varrho_0 = \{(\boldsymbol{P}_i, \{i\} \mid i \in S_0'\} ,$$

where e.g.

$$\boldsymbol{P}_1 = \Lambda \quad \text{(the empty string)}, \quad \boldsymbol{P}_2 = x_2 ,$$
$$\boldsymbol{P}_3 = x_1x_2x_1x_2x_2x_2x_2 , \qquad \boldsymbol{P}_4 = x_2x_2 ;$$

thus, f_0 is piece-wise computable on Ker $\mathscr{A}_0$.

If a function $f \in \mathfrak{F}_S$ is computed on a CPA $\mathscr{A} = (I, S, L, \delta)$ by a regular program $\mathscr{P}$, then f is piece-wise computable on the SPA Ker $\mathscr{A}$. Really, set

$$S' = \{s \in S \mid \beta(\mathscr{P}, s) \text{ is finite}\} ,$$

$\boldsymbol{P}_s = \omega(\mathscr{P}, s)$ for $s \in S'$; then Dom $f \subset S'$ and the set $\{(\boldsymbol{P}_s, \{s\}) \mid s \in S'\}$ is a partition of f regarding $\mathscr{A}$ and S'. (It is namely $s \in \text{Dom} f$ if and only if $(\boldsymbol{P}_s, s) \in \text{Dom}\, \delta^*$; this need not be true for $\mathscr{P}$ non-regular.)

The opposite implication in general does not hold. Its validity for a given CPA $\mathscr{A}$ depends clearly on "partition possibilities" of jumping instructions in the set I_L^{TOT}.

5. THE REALIZATION OF FINITE PARTITIONS

Let $\mathscr{A} = (I, S, L, \delta)$ be a CPA. We say that a jumping instruction $y \in I_L^{\text{TOT}}$ *realizes* a finite partition $\Delta = S_1| \dots | S_q$ of the set S if there are pair-wise different labels $z_1, \dots, z_q \in L$ such that for each $i \in \hat{q}$

$$\delta(y, \cdot)(S_i) = \{z_i\} .$$

Let $T \subset S$ be a nonempty set. We say that a partition $\Delta \in \mathfrak{P}_T^{\text{FIN}}$, $\Delta = T_1| \dots | T_q$ *is realizable* on $\mathscr{A}$ if there is a regular program $\mathscr{P} \in C_{\mathscr{A}}^+$ such that the following two conditions hold:

(1) For each $t \in T$ the sequence $\omega(\mathscr{P}, t)$ is empty.

(2) For each $t \in T$ the branch $\beta(\mathscr{P}, t)$ is finite and if the last command of $\beta(\mathscr{P}, t)$ is c_t, then $c_t \in \text{Dom}\, \lambda$ and there are pair-wise different labels $z_1, \dots, z_q \in L$ such that the implication

$$t \in T_i \Rightarrow \lambda(c_t) = z_i$$

holds for each $i \in \hat{q}$.

Then we also say that the program $\mathscr{P}$ *realizes* the partition Δ on $\mathscr{A}$. Note that $\iota(c_t) = \,!$ for each $t \in T$ and that the function computed by $\mathscr{P}$ is identical on the set T.

Thus, the program $\mathscr{P}$ makes it possible to distinguish two states of T if and only if they belong to different classes of the partition Δ. It is important that $\mathscr{P}$ uses no operational instructions – see (1); in fact it would be possible to assume that $\mathscr{P}$ consists of jumping and stop instructions only.

Example 3. The jumping instruction $y_1 \in I_0$ realizes the partition

$$\{1, 3, 5\} \mid \{0, 2, 4, 8, 10, 16\}$$

of the set S_0. Further, the program

$$\mathscr{P}_1 = [\langle 0, y_1\rangle, \langle 1, y_2\rangle, \langle 2, !\rangle, \langle 3, !\rangle, \langle 4, !\rangle]$$

realizes the partition

$$\Delta_1 = \{1\} \mid \{3, 5\} \mid \{0, 2, 4, 8, 10, 16\}$$

of the set S_0 on $\mathscr{A}_0$.

A partition $\Delta \in \mathfrak{P}_S^{\mathrm{FIN}}$ is said to be an *elementary partition* of the CPA $\mathscr{A}$ if there is a jumping instruction $y \in I_L^{\mathrm{TOT}}$ realizing Δ.

The aim of this paper is to find some connections between a given CPA $\mathscr{A}$ and the functions computable on $\mathscr{A}$. Here elementary partitions of $\mathscr{A}$ play a similar role as elementary functions of $\mathscr{A}$ (i.e. functions $\delta(x, \cdot)$ for $x \in I_S$) for investigation of the question for SPA. Some results dealing with composition of elementary functions are mentioned in [7]. Furthermore, in [7] it is defined a notion of composition of finite partitions and it is proved, that a partition $\Delta \in \mathfrak{P}_S^{\mathrm{FIN}}$ is realizable on $\mathscr{A}$ if and only if Δ is a composition of some elementary partitions of $\mathscr{A}$.

In what follows we shall not investigate the questions of composition (either functions or partitions), but rather the following problem: If the functions piece-wise computable on Ker $\mathscr{A}$ and the partitions realizable on $\mathscr{A}$ are known, what functions are then computable on $\mathscr{A}$?

The essential role in the investigation play special graphs – program graphs. A program graph of a given partition ϱ of a function f describes a possible structure of the operational part of a program which computes f and corresponds in a sense defined below to the partition ϱ.

6. ORIENTED MULTIGRAPHS

A quadruple $\mathscr{G} = (V, E, \boldsymbol{b}, \mathbf{e})$ is said to be an *oriented multigraph* if V is a set – the *set of vertices* of $\mathscr{G}$, E is a set – the *set of edges* of $\mathscr{G}$ and

$$\boldsymbol{b} : E \to V, \quad \mathbf{e} : E \to V.$$

The vertex $\boldsymbol{b}(e)$ or $\mathbf{e}(e)$ is called the *begin* or the *end* of the edge $e \in E$, respectively.

For each $v \in V$ define

$$E_i(v) = \{e \in E \mid \mathbf{e}(e) = v\}, \quad \mathrm{i}(v) = \mathrm{card}\, E_i(v),$$
$$E_o(v) = \{e \in E \mid \boldsymbol{b}(e) = v\}, \quad \mathbf{o}(v) = \mathrm{card}\, E_o(v).$$

By a *path* in $\mathcal{G}$ is understood each finite or infinite sequence $\boldsymbol{p} = v_0 e_1 v_1 e_2 v_2 \dots$ such that for each i it is

$$v_i \in V\,, \quad e_i \in E\,, \quad \mathbf{b}(e_i) = v_{i-1}\,, \quad \mathbf{e}(e_i) = v_i$$

and if a is the last member of $\boldsymbol{p}$, then $a \in V$.

We shall sometimes briefly write

$$\boldsymbol{p} = v_0 v_1 v_2 \dots \quad \text{or} \quad \boldsymbol{p} = e_1 e_2 \dots .$$

Set

$$V_{\mathrm{FIN}} = \{v \in V \mid \mathbf{o}(v) = 0\}\,, \quad E_{\mathrm{FIN}} = \{e \in E \mid \mathbf{e}(e) \in V_{\mathrm{FIN}}\}\,,$$

$$V_{\mathrm{CON}} = \{v \in V \mid \mathbf{o}(v) \geqq 2\}\,, \quad E_{\mathrm{CON}} = \{e \in E \mid \mathbf{b}(e) \in V_{\mathrm{CON}}\}\,.$$

The sets V_{FIN}, E_{FIN}, V_{CON}, E_{CON} are called the *set of final vertices, final edges, conditional vertices* and *conditional edges* of $\mathcal{G}$, respectively.

7. PROGRAM GRAPHS

Let $\mathcal{A} = (I, S, L, \delta)$ be a CPA.

By a *program graph* (cf. [10]) is understood a sixtuple $\mathcal{G} = (V, E, \mathbf{b}, \mathbf{e}, \varphi, \psi)$ such that $(V, E, \mathbf{b}, \mathbf{e})$ is an oriented multigraph and the following conditions hold:

(1) $\forall\, v \in V(\mathrm{i}(v) \in \boldsymbol{N}_0$ et $\mathrm{o}(v) \in \boldsymbol{N}_0)$.

(2) There is exactly one $v \in V$ such that $\mathrm{i}(v) = 0$. This v is called the *initial vertex* of $\mathcal{G}$.

(3) $\varphi : E \to I_S \cup \{!, ?\}$ such that

$$\varphi(E_{\mathrm{FIN}}) \subset \{!, ?\} \quad \text{and} \quad \varphi(E - E_{\mathrm{FIN}}) \subset I_S\,.$$

(4) $\psi : E_{\mathrm{CON}} \to \exp S$ such that

$$\forall\, v \in V_{\mathrm{CON}}\,, \quad \forall\, e_1\,, \quad e_2 \in E_0(v)\,(e_1 \neq e_2 \Rightarrow \psi(e_1) \cap \psi(e_2) = \emptyset)\,.$$

For $v \in V_{\mathrm{CON}}$ denote $\Delta_v = \underset{e \in E_0(v)}{\big|}\ \psi(e)$ and say that the vertex v *realizes* the partition Δ_v of the set $\bigcup_{e \in E_0(v)} \psi(e)$.

Thus, the graph $\mathcal{G}$ describes a branching of some operational sequences. The mapping φ assignes operational instructions to the edges of $\mathcal{G}$, ψ serves for describing a dependence of the branching on the "initial states".

Now a dependence of $\mathcal{G}$ on a partition ϱ of f will be specified. First we shall define such a path in $\mathcal{G}$ one moves on if begins in the initial vertex of $\mathcal{G}$ with a given state $s \in S$ and proceeds in the direction of oriented edges, respecting branching in the conditional vertices in such a way that as the next edge is always the one labelled by s chosen.

Let $\mathscr{G}$ be a program graph. For each $s \in S$ define a path $\boldsymbol{p}_s$ as follows. Let $\boldsymbol{p}_s = v_0 e_1 v_1 e_2 v_2 \ldots$ be a finite or infinite path in $\mathscr{G}$ such that the following conditions hold:

(1) $\mathrm{i}(v_0) = 0$.

(2) For each $n \in \boldsymbol{N}_0$, if $v_n \in V_{\mathrm{CON}}$ is not the last vertex of $\boldsymbol{p}_s$, then $s \in \psi(e_{n+1})$.

(3) If for some $n \in \boldsymbol{N}_0$ is v_n the last vertex of $\boldsymbol{p}_s$, then either $v_n \in V_{\mathrm{FIN}}$ or

$$v_n \in V_{\mathrm{CON}} \quad \text{et} \quad \forall\, e \in E_0(v_n)\,(s \notin \psi(e))\,.$$

Clearly, the path $\boldsymbol{p}_s$ is by the state s uniquely determined. Note that if $\boldsymbol{p}_s$ is finite, then

$$v_i \neq v_j\,, \quad e_i \neq e_j \quad \text{for} \quad i \neq j\,.$$

For $v \in V_{\mathrm{CON}}$ denote

$$S_v = \{s \in S \mid v \text{ is a vertex of } \boldsymbol{p}_s\}\,.$$

Let $f \in \mathfrak{F}_S$ and let $\varrho = \{(\boldsymbol{P}_j, S_j) \mid j \in J\}$ be a partition of f regarding Ker $\mathscr{A}$ and $S' \subset S$.

A program graph $\mathscr{G} = (V, E, \boldsymbol{b}, \mathbf{e}, \varphi, \psi)$ is said to be a *program graph of the partition* ϱ if the following conditions hold:

(1) $\forall\, v \in V_{\mathrm{CON}}(S_v = \bigcup\limits_{e \in E_0(v)} \psi(e))\,.$

(2) For each $s \in S$ the path $\boldsymbol{p}_s$ is finite, $\boldsymbol{p}_s = v_0 e_1 v_1 \ldots e_n v_n$ and either it is $s \in S - S'$ and then $n \geqq 1$ and $\varphi(e_n) = ?$ or it is $s \in S_j$ for some $j \in J$, $\boldsymbol{P}_j = x_1 \ldots x_m$ $(m \in \boldsymbol{N}_0)$ and then $n = m + 1$ and $\varphi(e_i) = x_i$ for $i \in \hat{m}$, $\varphi(e_n) = !$.

Note that the condition (1) and the finiteness of $\boldsymbol{p}_s$ imply $v_n \in V_{\mathrm{FIN}}$. Note further that the partition ϱ can possess more than one various program graphs.

Example 4. Let $\mathscr{G}_0$ be a graph possessing the diagram presented in Fig. 1. $\mathscr{G}_0$

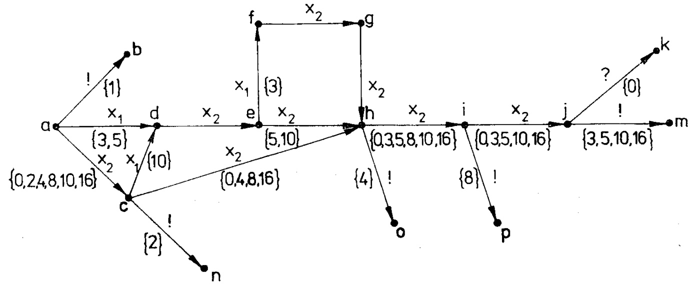

Fig. 1.

has 6 final vertices (and edges): b, k, m, n, o, p, 6 conditional vertices: a, c, e, h, i, j and 14 conditional edges.

Further, $\mathscr{G}_0$ is a program graph with the initial vertex a. E.g. $\varphi(a, c) = x_2$, $\varphi(j, k) = ?$, $\psi(c, h) = \{0, 4, 8, 16\}$, $\boldsymbol{p}_0 = a\ c\ h\ i\ j\ k$, $S_e = \{3, 5, 10\}$.

Finally, it would be clear that $\mathscr{G}_0$ is a program graph of the partition ϱ_0 of the function f_0.

8. PROGRAMABLE PROGRAM GRAPHS

The aim of this section is to formulate conditions that enable us to realize the partitions required in conditional vertices.

Let $T \subset S$, $T \neq \emptyset$ be a set. We say that a partition $\Delta \in \mathfrak{P}_T^{\mathrm{FIN}}$, $\Delta = T_1 | \dots | T_q$ *separates* sets $M_1, \dots, M_r \subset T$ if

$$\forall\, i, j \in \hat{r}(M_i, M_j \neq \emptyset \Rightarrow \exists\, m, n \in \hat{q}(M_i \subset T_m$$
$$\text{et} \quad M_j \subset T_n \quad \text{et} \quad (i \neq j \Rightarrow m \neq n)))\,.$$

Let $f \in \mathfrak{F}_S$, let ϱ be a partition of f (regarding Ker $\mathscr{A}$) and let $\mathscr{G}$ be a program graph of ϱ. Further, let v be an arbitrary conditional vertex of $\mathscr{G}$ and let v realize a partition $\Delta = V_1 | \dots | V_q$ of the set S_v.

Let $s \in S_v$; if $\boldsymbol{p}_s = v_0 e_1 v_1 \dots e_n v_n$, denote $\boldsymbol{P}_s = \varphi(e_1) \dots \varphi(e_m)$, where $m \in \hat{n} \cup \{0\}$ is a (uniquely determined) integer such that $v_m = v$. Set

$$f_s = \delta^*(\boldsymbol{P}_s, \cdot)\,.$$

Finally, for $j \in \hat{q}$ denote

$$T_v^j = \{f_s(s) \mid s \in V_j \cap \operatorname{Dom} f_s\}\,.$$

Thus, the sets T_v^j are just the ones that ought to be distinguished in the conditional vertex v; the states s for which $s \notin \operatorname{Dom} f_s$ holds may be (and in fact are) neglected.

The conditional vertex v is said to be *programable* on $\mathscr{A}$ if there are a set $T \neq \emptyset$ such that $\bigcup_{j \in \hat{q}} T_v^j \subset T \subset S$ and a partition $\Delta \in P_T^{\mathrm{FIN}}$ separating the sets $T_v^1, \dots, T_v^q$ and realizable on $\mathscr{A}$.

Say that a program graph is *programable* on $\mathscr{A}$ if each its conditional vertex is.

Example 5. Consider the conditional vertices c, h of $\mathscr{G}_0$. Here

$$S_c = \{0, 2, 4, 8, 10, 16\}\,, \quad S_h = \{0, 3, 4, 5, 8, 10, 16\}$$

and e.g.

$$\boldsymbol{P}_s^c = x_2 \quad \text{for} \quad s \in S_c\,, \quad f_s^c(s) = \tfrac{1}{2}s \quad \text{for} \quad s \in S_c\,,$$
$$\boldsymbol{P}_0^h = x_2 x_2\,, \quad \boldsymbol{P}_3^h = x_1 x_2 x_1 x_2 x_2\,, \quad \boldsymbol{P}_{10}^h = x_2 x_1 x_2 x_2\,,$$
$$f_5^h(5) = 4\,, \quad f_4^h(4) = 1\,.$$

Further, for the conditional vertex c,

$$q = 3\,, \quad V_1 = \{10\}\,, \quad V_2 = \{0, 4, 8, 16\}\,, \quad V_3 = \{2\}\,;$$

hence

$$T_c^1 = f_{10}^c(V_1) = \{5\}\,, \quad T_c^2 = \{0, 2, 4, 8\}\,, \quad T_c^3 = \{1\}\,;$$

thus the sets T_c^1, T_c^2, T_c^3 are separated by the partition Δ_1, which is realized by the program $\mathscr{P}_1$ on $\mathscr{A}_0$ (see Example 3). Thus, the vertex c is programable on $\mathscr{A}_0$.

9. THE OPERATIONAL TREE

In this section for a given regular program $\mathscr{P}$ a special graph describing the operational branches of $\mathscr{P}$ and their branching will be defined. On this purpose some further notions are necessary.

Let $\mathscr{A} = (I, S, L, \delta)$ be a CPA and let $\mathscr{P} \in C_{\mathscr{A}}^+$ be a regular program. Assume (without any loss of generality) that each command of $\mathscr{P}$ is labelled.

For $s \in S$ let $\omega(s)$ be the sequence arising from the branch $\beta(\mathscr{P}, s)$ by deleting all jumping commands.

Suppose that $\mathscr{P}$ has the following property: If $s \in S$ is such a state that the sequence $\beta(\mathscr{P}, s)$ is finite and if $\omega(s) = c_1, \ldots, c_m$, then $\lambda(c_i) \neq \lambda(c_j)$ for $i \neq j$, $i, j \in \hat{m}$. Programs, possessing the just mentioned property, will be called *acyclic*.

Note that the program $\mathscr{P}_0$ is regular but not acyclic.

Now we want to define certain sequences λ^s for $s \in S$. Distinguish two cases:

Let firstly $s \in S$ be such that the branch $\beta(\mathscr{P}, s)$ is infinite ($\omega(s)$ may be finite), $\omega(s) = c_1, c_2, \ldots$. Then we are going to cut off the sequence $\omega(s)$ in order to make it finite. Let $p \in N_0$ be the largest integer such that there is $t \in S$ for which the branch $\beta(\mathscr{P}, t)$ is finite (if no finite $\beta(\mathscr{P}, t)$ exists, set $p = 0$) and the sequence $\omega(t)$ has the first p members the same as the sequence $\omega(s)$. (Such a p exists, because $\mathscr{P}$ is supposed to be acyclic and hence the set of lengths of all sequences $\omega(t)$ for which $\beta(\mathscr{P}, t)$ is finite is bounded.) Set

$$\omega'(s) = c_1, \ldots, c_p, c\,, \quad \text{where} \quad c = \langle z, ? \rangle\,, \quad z \in L\,.$$

The label z is the same for all s and is different from all labels used in $\mathscr{P}$.

If secondly $s \in S$ is such that $\beta(\mathscr{P}, s)$ is finite, set

$$\omega'(s) = \omega(s)\,.$$

Now, let for $s \in S$ be $\omega'(s) = c_1^s, \ldots, c_{m_s}^s$. Note that $m_s \geqq 1$.

Denote $\lambda^s = z_1^s, \ldots, z_{m_s}^s$, where $z_j^s = \lambda(c_j^s)$ ($j = 1, \ldots, m_s$). Note that $z_i^s \neq z_j^s$ for $i \neq j$, $i, j \in \hat{m}_s$.

Define a set

$$W = \{(z_i^s, s) \mid s \in S \quad \text{et} \quad i \in \hat{m}_s\}$$

and for $(z_i^s, s), (z_j^t, t) \in W$ write

$$(z_i^s, s) \equiv (z_j^t, t)$$

if $i = j$, $z_i^s = z_i^t$ and either $i = 1$ or $(z_{i-1}^s, s) \equiv (z_{i-1}^t, t)$. It is clear that $\equiv$ is an equivalence relation on W. It is not difficult to see that the index of $\equiv$ is finite.

Denote V_0 the set of all $\equiv$-classes of the set W, set $V_1 = V_0 \cup \{v_0\}$ and define

$$E_1 = \{(\langle(z_j^s, s)\rangle, \langle(z_{j+1}^s, s)\rangle) \mid s \in S \quad \text{et} \quad 1 \leqq j \leqq m_s - 1\} \cup$$
$$\cup \{(v_0, \langle(z_1^s, s)\rangle) \mid s \in S\} ,$$

where $\langle w \rangle$ denotes the class containing $w \in W$.

It is easy to see that the oriented multigraph

$$\mathcal{G}_1 = (V_1, E_1, \pi_1^{E_1}, \pi_2^{E_1})$$

is a tree, i.e. $\mathrm{i}(v_0) = 0$ and $\mathrm{i}(v) = 1$ for each $v \in V_0$.

Furthermore, define a mapping

$$\varphi_1 : E_1 \to I_S \cup \{!, ?\}$$

as follows: let $e \in E_1$ and let $w \in W$ be such that $\langle w \rangle = \pi_2(e)$; then set $\varphi_1(e) = \iota(\lambda^{-1}(\pi_1(w)))$. (The definition is correct: $w' \equiv w$ implies $\pi_1(w') = \pi_1(w)$.)

Let $u_1, \ldots, u_p$ be all vertices of $\mathcal{G}_1$ such that $\mathbf{o}(u_i) = 0$ and $\varphi_1(e_i) \in I_S$, where e_i is the edge such that $\pi_2(e_i) = u_i$ $(i \in \hat{p})$. Define

$$V = V_1 \cup \{v_1, \ldots, v_p\} , \quad \text{where} \quad v_i \neq v_j \quad \text{for} \quad i \neq j,\ i, j \in \hat{p} ,$$
$$E = E_1 \cup \{(u_i, v_i) \mid i \in \hat{p}\} ,$$
$$\mathcal{G} = (V, E, \pi_1^E, \pi_2^E) ,$$
$$\varphi = \varphi_1 \cup \{((u_i, v_i), !) \mid i \in \hat{p}\} .$$

For each $s \in S$ define $\boldsymbol{p}^s$ as a path in $\mathcal{G}$ such that

$$\boldsymbol{p}^s = v_0 , \quad \langle(z_1^s, s)\rangle, \ldots , \quad \langle(z_{m_s}^s, s)\rangle , \quad u_1, \ldots, u_k ,$$

$k \in \boldsymbol{N}_0$ and the last vertex of $\boldsymbol{p}^s$ is a final vertex of $\mathcal{G}$. If more than one such paths exist, choose one of them as $\boldsymbol{p}^s$.

Now we can define a mapping $\psi : E_{\mathrm{CON}} \to \exp S$ by the following way: for $e \in E_{\mathrm{CON}}$ set

$$\psi(e) = \{s \in S \mid e \text{ is an edge of } \boldsymbol{p}^s\} .$$

Finally, define the *operational tree* of the program $\mathcal{P}$ as a labelled multigraph $\mathcal{T}_{\mathcal{P}} = (V, E, \pi_1^E, \pi_2^E, \varphi, \psi)$. It is easy to verify, that $\mathcal{T}_{\mathcal{P}}$ is really a tree and that it is a program graph.

Theorem 1. Let $\mathscr{A} = (I, S, L, \delta)$ be a CPA and let a function $f \in \mathfrak{F}_S$ be on $\mathscr{A}$ computed by an acyclic (regular) program $\mathscr{P}$. Then there are a finite partition ϱ of f (regarding Ker $\mathscr{A}$) and a programable program graph $\mathscr{G}$ of ϱ, which, moreover, is a tree.

Proof. For each $s \in S$ denote e^s the last edge of the path $\boldsymbol{p}^s$ in the operational tree $\mathscr{T}_{\mathscr{P}}$ of $\mathscr{P}$. Set

$$S' = \{s \in S \mid \varphi(e^s) = !\} ;$$

then $\operatorname{Dom} f \subset S'$.

For $s, t \in S'$ define $s \equiv t$ if $\boldsymbol{p}^s = \boldsymbol{p}^t$. Clearly, the relation $\equiv$ is an equivalence relation on S' and its index is finite (because $\mathscr{T}_{\mathscr{P}}$ is finite).

Let $S_1 | \dots | S_k$ be the partition of S' induced by $\equiv$. Let $j \in \hat{k}$, $s \in S_j$ and let $\boldsymbol{p}^s = e_1 \dots e_n$ $(n \geqq 1)$; then set $\boldsymbol{P}_j = \varphi(e_1) \dots \varphi(e_{n-1})$.

It is easy to see that $\varrho = \{(\boldsymbol{P}_1, S_1), \dots, (\boldsymbol{P}_k, S_k)\}$ is a finite partition of f regarding Ker $\mathscr{A}$ and S'.

Prove further, that $\mathscr{T}_{\mathscr{P}}$ is a programable program graph of ϱ. It is not difficult to see that for each $s \in S$ the equality $\boldsymbol{p}^s = \boldsymbol{p}_s$ holds. Hence, the conditions (1), (2) of the definition of a program graph of ϱ in Section 7 hold.

Let v be an arbitrary conditional vertex of $\mathscr{T}_{\mathscr{P}}$ and let v realize a partition $V_1 | \dots | V_q$ $(q \geqq 2)$ of the set S_v. In order to complete the proof we shall prove that the vertex v is programable on $\mathscr{A}$.

Remember the definition of the sets T_v^j for $j \in \hat{q}$:

$$T_v^j = \{f_s(s) \mid s \in V_j \cap \operatorname{Dom} f_s\} .$$

Here, of course, the situation is simpler than in a general case of a program graph because $\mathscr{T}_{\mathscr{P}}$ is a tree: For $s, t \in S_v$ $f_s = f_t$ holds; thus we can set $f_v = f_s$ for $s \in S_v$. Now we have

$$T_v^j = f_v(V_j \cap \operatorname{Dom} f_v)$$

and setting

$$T_v = \bigcup_{j \in \hat{q}} T_v^j \quad \text{and} \quad U_v = S_v \cap \operatorname{Dom} f_v$$

we gain $T_v = f_v(U_v)$.

Let $s \in U_v$ and let $\beta(\mathscr{P}, s) = c_1^s, c_2^s, \dots$. Then there are integers $i_s, j_s \in N_0$ such that

$$j_s - i_s \geqq 2 , \quad \langle(\lambda(c_{i_s}^s), s)\rangle = v$$

(where $\langle w \rangle$ denotes the class of the equivalence from the definition of $\mathscr{T}_{\mathscr{P}}$ which contains w),

$$\iota(c_i^s) \in I_L \quad \text{for} \quad i_s + 1 \leqq i \leqq j_s - 1 \quad \text{and} \quad \iota(c_{j_s}^s) \in I_S .$$

If v equals the initial vertex v_0 of $\mathcal{T}_{\mathcal{P}}$, we set formally

$$\langle(\lambda(c_0^s), s)\rangle = v_0 ;$$

thus $i_s = 0$.

Note that $\lambda(c_{j_t}^t) = \lambda(c_{j_u}^u)$ if and only if $t, u \in V_j \cap \operatorname{Dom} f_v$ for some $j \in \hat{q}$. Denote

$$z_j = \lambda(c_{j_s}^s) \quad \text{for} \quad s \in V_j \cap \operatorname{Dom} f_v .$$

Define

$$C_s = \{c_{i_s+1}^s, \ldots, c_{j_s-1}^s\} .$$

Now let

$$\mathcal{P}_v = [c^1, \ldots, c^p, \langle z_1, !\rangle, \ldots, \langle z_q, !\rangle]$$

be such a program that the following conditions hold:

(1) $c^1 = c_{i_s+1}^s$ for $s \in U_v$ $(t, u \in U_v$ implies $c_{i_t+1}^t = c_{i_u+1}^u)$,

(2) $\{c^1, \ldots, c^p\} = \bigcup_{s \in U_v} C_s$.

As it is clear, the program $\mathcal{P}_v$ realizes some partition Δ of the set T_v. We are going to prove now that Δ separates the sets $T_v^1, \ldots, T_v^q$.

Let $j \in \hat{q}$ and $t \in T_v^j$. Then there is $s \in V_j \cap \operatorname{Dom} f_v$ such that $t = f_v(s)$. If we again have

$$\beta(\mathcal{P}, s) = c_1^s, c_2^s, \ldots, c_{i_s}^s, \ldots, c_{j_s}^s, \ldots ,$$

where $\lambda(c_{j_s}^s) = z_j$, then it is clear that the "branch" of the program $\mathcal{P}$ beginning with the command $c_{i_s+1}^s$ and in the state t results from the branch $\beta(\mathcal{P}, s)$ by removing the first i_s members. Hence

$$\beta(\mathcal{P}_v, t) = c_{i_s+1}^s, c_{i_s+2}^s, \ldots, c_{j_s-1}^s, \langle z_j, !\rangle . \qquad \square$$

Let $\mathcal{G}$ be a program graph. A path $\boldsymbol{p} = v_1 \ldots v_m$ $(m \geqq 2)$ in $\mathcal{G}$ is said to be a *chain* in $\mathcal{G}$ if the following conditions hold:

(1) $\mathrm{i}(v_j) \,.\, \mathrm{o}(v_j) \neq 1$ for $j = 1, m$.

(2) $\mathrm{i}(v_j) \,.\, \mathrm{o}(v_j) = 1$ for $j = 2, \ldots, m - 1$.

The vertices v_1 and v_m are called the *begin* and the *end* of the chain $\boldsymbol{p}$, respectively.

A chain in $\mathcal{G}$ is said to be *final* if its end is a final vertex of $\mathcal{G}$.

A CPA $\mathcal{A} = (I, S, L, \delta)$ is said to be *semi-complete* if the condition

$$\forall z \in L \ \exists y \in I_L^{\mathrm{TOT}} \ \forall s \in S \ (\delta(y, s) = z)$$

holds. Such a y will be denoted by $w[z]$ in what follows.

Theorem 2. Let $\mathcal{A} = (I, S, L, \delta)$ be a semi-complete CPA. Let $f \in \mathfrak{F}_S$ be a function such that there are a finite partition ϱ of f (regarding Ker $\mathcal{A}$) and a programable program graph of ϱ. Then f is on $\mathcal{A}$ computed by an acyclic program.

Proof. Let $\varrho = \{(\boldsymbol{P}_1, S_1), \ldots, (\boldsymbol{P}_k, S_k)\}$ and let $\mathscr{G}$ be a (finite) programable program graph of ϱ.

Let v be a conditional vertex of $\mathscr{G}$, let v realize a partition $V_1 | \ldots | V_q$ of the set S_v, let $T \subset S$ and let $\Delta \in \mathfrak{P}_T^{\mathrm{FIN}}$ be a partition separating the sets $T_v^1, \ldots, T_v^q$, which is realized on $\mathscr{A}$ by a regular program $\mathscr{P}_\Delta = [c^1, \ldots, c^p]$. Suppose $T_v^i \neq \emptyset$ for $i \in \hat{q}$ (each empty T_v^i may be deleted).

For $s \in T$ denote c_s the last command of $\beta(\mathscr{P}_\Delta, s)$ and let for $s \in T_v^i$ be $\lambda(c_s) = z_i$ $(i = 1, \ldots, q)$.

Set $\mathscr{P}_v = [\bar{c}^1, \ldots, \bar{c}^p]$, where

$$\bar{c}^1 = \langle \bar{z}, \iota(c^1) \rangle\,,$$

$$\bar{c}^j = \begin{cases} \langle \lambda(c^j), w[\bar{z}_i] \rangle & \text{if} \quad \lambda(c^j) = z_i \quad \text{for some} \quad i \in \hat{q} \quad (j = 2, \ldots, p)\,, \\ c^j & \text{else} \end{cases}$$

where $\bar{z}, \bar{z}_1, \ldots, \bar{z}_q$ are pair-wise different and different from $\lambda(c^j)$ for $j \in \hat{p}$, too. Denote

$$\mathfrak{I}(\mathscr{P}_v) = \bar{z}\,, \quad \mathfrak{O}_{V_i}(\mathscr{P}_v) = \bar{z}_i \quad (i = 1, \ldots, q)\,.$$

Further, let $\boldsymbol{p}$ be a chain in $\mathscr{G}$ which is not final, $\boldsymbol{p} = e_1 \ldots e_m$. Set

$$\mathscr{P}_p = [\langle z, \varphi(e_1) \rangle, \varphi(e_2), \ldots, \varphi(e_m), w[z']]$$

and denote $\mathfrak{I}(\mathscr{P}_p) = z$, $\mathfrak{O}(\mathscr{P}_p) = z'$.

If $\boldsymbol{p}$ is a final chain in $\mathscr{G}$, $\boldsymbol{p} = e_1 \ldots e_m e$, set for $m \geqq 1$

$$\mathscr{P}_p = [\langle z, \varphi(e_1) \rangle, \varphi(e_2), \ldots, \varphi(e_m), c]\,,$$

where

$$c = \begin{cases} ! & \text{for} \quad \varphi(e) = ! \\ \langle z', w[z'] \rangle & \text{for} \quad \varphi(e) = ?\,; \end{cases}$$

for $m = 0$ set

$$\mathscr{P}_p = \begin{cases} \langle z, ! \rangle & \text{for} \quad \varphi(e) = ! \\ \langle z, w[z] \rangle & \text{for} \quad \varphi(e) = ?\,. \end{cases}$$

Denote $\mathfrak{I}(\mathscr{P}_p) = z$.

Now, let $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_u$ be all chains, $v_1, \ldots, v_w$ all conditional vertices of $\mathscr{G}$. It follows from the semi-completeness of $\mathscr{A}$ that the labels of all commands in sequences $\mathscr{P}_{p_1}, \ldots, \mathscr{P}_{p_u}, \mathscr{P}_{v_1}, \ldots, \mathscr{P}_{v_w}$ can be chosen pair-wise different, unless by the following conditions is required the contrary.

(1) If the end of a chain $\boldsymbol{p}$ is a vertex v such that $\mathbf{o}(v) \neq 0$, then

$$\mathfrak{O}(\mathscr{P}_p) = \begin{cases} \mathfrak{I}(\mathscr{P}_q), \text{ if } \mathbf{o}(v) = 1 \text{ and } v \text{ is the begin of a chain } \boldsymbol{q}\,, \\ \mathfrak{I}(\mathscr{P}_v) \text{ else}\,. \end{cases}$$

(2) If the begin of a chain $\boldsymbol{p}$ is a conditional vertex v and if the first edge of $\boldsymbol{p}$ is e, then

$$\mathfrak{I}(\mathscr{P}_p) = \mathfrak{O}_{\psi(e)}(\mathscr{P}_v) \,.$$

Set $\mathscr{P} = [c^1, \ldots, c^r]$, where the sequence $c^1, \ldots, c^r$ is a juxtaposition of the sequences $\mathscr{P}_{p_1}, \ldots, \mathscr{P}_{p_u}, \mathscr{P}_{v_1}, \ldots, \mathscr{P}_{v_w}$ in an arbitrary order, but such that if the initial vertex v_0 of $\mathscr{G}$ is conditional, then the leftmost sequence is $\mathscr{P}_{v_0}$ and if it is not the case and if v_0 is the begin of a chain $\boldsymbol{p}$, then the leftmost sequence is $\mathscr{P}_p$.

Now it is not difficult to prove formally that the program $\mathscr{P}$ computes the function f on $\mathscr{A}$. Furthermore, it follows from the choice of labels in $\mathscr{P}$ that $\mathscr{P}$ is acyclic. □

Theorem 1 provides a necessary condition for a function to be computed on a CPA $\mathscr{A}$ by an acyclic program. By Theorem 2 the condition is even sufficient provided that the CPA $\mathscr{A}$ is semi-complete. Note that the semi-completeness is a quite natural and reasonable demand; it realizes the first step to the label-independence: not labels attainable by jumping instructions are important for the computational capacity of $\mathscr{A}$ but only realizable partitions are. This independence is achieved in [7] by means of the notion of completeness, mainly in connection with compositions of partitions mentioned above.

Both theorems show that the concepts of a partition of a function as well as of (programable) program graph are adequate tools for describing relations between the set of instructions (i.e. elementary functions and partitions) on the one hand and the set of computable functions on the other.

Perhaps the most interesting corollary of both the theorems (for semi-complete CPA) is the following one: If f has a finite partition with a programable program graph $\mathscr{G}$, then $\mathscr{G}$ can be "transformed" into a tree which remains to be a programable program graph of some finite partition of f. Or in other words: For each acyclic program there is an equivalent (i.e. computing the same function) program the operational structure of which has a form of a tree.

It is clear that further generalization of the concepts defined in this article is possible and desirable. For example non-acyclic and even non-regular programs and corresponding compositions and graphs are to be studied in more details.

(Received December 31, 1976.)

REFERENCES

[1] A. Blikle, A. Mazurkiewicz: An Algebraic Approach to the Theory of Programs, Algorithms, Languages and Recursiveness. Proc. Symp. on Math. Found. of Computer Sci., Jablonna 1972.

[2] K. Čulík: On Sequential and Non-Sequential Machines and their Relation to the Computation in Computers. Mimeographed in IFIP WG 2.2 Bulletin, No. 6, February 1970.

[3] K. Čulík, M. A. Arbib: Sequential and Jumping Machines and their Relation to Computers. Acta Informatica *2* (1973), 162–171.

[4] K. Čulík: Algorithmic Algebras of Computers. Czechoslovak Math. Journal *23* (1973), 670—689.

[5] K. Čulík: Some Notes on Logical Analysis of Programming Languages. Teorie a metoda *3* (1971), 101—111.

[6] Ю. И. Янов: О логических схемах алгоритмов. Проблемы кибернетики *1* (1958), 75—127.

[7] J. Mareš: Programmed Automata. (In Czech.) CSc. — thesis, Prague 1973 (not published).

[8] R. Milner: Equivalences on Program Schemes. Jour. Comp. and Syst. Sciences *4* (1970), 205—219.

[9] Z. Pawlak: Stored Program Computers. (In Polish.) Algorytmy Vol. *5*, No. 10 (1969), 5—19.

[10] H. Rossner: Formalization of the Notion of the Program. (In Polish.) Algorytmy Vol. *5*, No. 10 (1969), 25—43.

RNDr. Jan Mareš, CSc., fakulta jaderná a fyzikálně inženýrská ČVUT (Faculty of Nuclear and Physical Engineering — Czech Technical University), Břehová 7, 115 19 Praha 1. Czechoslovakia.