Czechoslovak Mathematical Journal

Karel Čulík

Algorithmic algebras of computers

Czechoslovak Mathematical Journal, Vol. 23 (1973), No. 4, 670–689

Persistent URL: http://dml.cz/dmlcz/101207

Czechoslovak Mathematical Journal, 23 (98) 1973, Praha

ALGORITHMIC ALGEBRAS OF COMPUTERS*)

KAREL ČULÍK, Praha

(Received December 22, 1972)

A pure mathematical description of computer and of its activity is given. The functional unit of the computer is represented by an algebra, the operation of which concern the basic, i.e. not structured, objects only. The storage is represented by a set of functions (special sorts of storage are not distinguished). The concept of similarity of algorithms = programs formalizes the intuitive independency of computable functions on the input, output variables and labels, used in particular cases. The similarity of algorithms implies their functional equivalence. The width of an algorithm is introduced and the minimal width is determined as the chromatic number of a graph defined by the scopes of variables.

1. ALGORITHMIC ALGEBRAS

An *algebra* $\langle U, F\rangle$, where U, F is its set of elements, functions, respectively, is called *algorithmic*, if two sorts of its elements and of its functions are distinguished such that

(1.1) $U = Obj \cup Lab$, *where* $Obj \cap Lab = \emptyset$, *and* $F = Opr \cup Dec$, *and if the following requirements are satisfied*:

(1.2) $f^{(n)} \in Opr \cup Dec \Rightarrow \mathrm{Domain}\, f^{(n)} \subset Obj^n$,
$f^{(n)} \in Opr \Rightarrow \mathrm{Range}\, f^{(n)} \subset Obj$,
$f^{(n)} \in Dec \Rightarrow \mathrm{Range}\, f^{(n)} \subset Lab$.

The elements from *Obj*, *Lab* are called *objects*, *labels*, respectively, and the functions (which always may be partial) from *Opr*, *Dec* are called *operations*, *decisions*, respectively. Thus an algorithmic algebra is determined by a quadruple $AA = \langle Obj, Lab, Opr, Dec\rangle$.

*) This paper is a revised version of a talk presented at the International Symposium and Summer School on Mathematical Foundations of Computer Science, Warsaw, Jabłonna, August 21–27, 1972.

A *relational structure* $RS = \langle Obj, Opr, Rel \rangle$, where *Rel* is its set of relations, i.e. if $r^{(n)} \in Rel$ then Field $r^{(n)} = Obj^n$, may be considered as an algorithmic algebra $\langle Obj, \{true, false\}, Opr, Rel \rangle$, if the relations from *Rel* are considered as the well known truth-functions.

If we use an auxiliary set of symbols *Lab*, which satisfies (1.1) with respect to the *RS*, then there exists an other way how to get an algorithmic algebra from *RS*. In general it is admitted that *Rel* contains n-ary relations in k-valued logic $r^{(n)}_{(k)}$, where $n \geqq 1$ and $k \geqq 2$ are arbitrary integers, which are partial, i.e. one admits Field $r^{(n)}_{(k)} \subset$ $\subset Obj^n$.

If $r^{(n)}_{(k)} \in Rel$ and if $[\mathbf{1}, \mathbf{2}, \ldots, \mathbf{k}]$ is a fixed ordering of all k *truth values* (for $k = 2$ we assume $\mathbf{1} \equiv true$ and $\mathbf{2} \equiv false$), then by each k-tuple $[a_1, a_2, \ldots, a_k]$ of labels from *Lab* the following function $r^{(n)}_{[a_1, a_2, \ldots, a_k]}$ is defined:

(1.3) $r^{(n)}_{[a_1,\ldots,a_k]}(o_1, o_2, \ldots, o_n) =_{\mathrm{df}} a_i \Leftrightarrow$ *the truth value of the formula* $r^{(n)}_{(k)}(o_1, o_2, \ldots$ $\ldots, o_n)$ *is* $\mathbf{i}$, *where* $(o_1, o_2, \ldots, o_n) \in$ Field $r^{(n)}_{(k)}$ *and* $1 \leqq i \leqq k$.

The set

(1.4) $Rel_{[Lab]} = \{r^{(n)}_{[a_1,\ldots,a_k]};\ r^{(n)}_{(k)} \in Rel$ *and* $a_i \in Lab$ *for* $i = 1, 2, \ldots, k\}$

satisfies (1.2) and therefore it is the set of decisions of the algorithmic algebra $\langle Obj, Lab, Opr, Rel_{[Lab]} \rangle$ which corresponds to the *RS* and to the chosen set *Lab*.

2. COMPUTER AND ITS STORAGE

It is assumed that the only aim of a computer is to evaluate functions, but of many different types. The *basic type* corresponds to the *basic objects* the computer is dealing with, and the *structured types* correspond to the *structured objects* (e.g. strings or matrices of basic objects, etc.). It depends on the level of description which objects are considered as basic. Usually the sequences of zeros and ones of the length 64 are considered as basic objects, but we shall admit that among the basic objects are integers and real numbers (expressed in decimal system by sequences of a fixed length), sequences of Latin letters of fixed length, etc.

The *functional unit* of a computer allows to evaluate certain basic functions, called *hardware operations* and *relations*, because they are incorporated within its *hardware*, while all other functions are evaluated according to their *programs*, which are *algorithms* belonging to the *software* of the computer.

The functional unit determines a relational structure $\langle$*basic objects, hardware operations, hardware relations*$\rangle$ and therefore the corresponding algorithmic algebra of the form $AA = \langle Obj, Lab, Opr, Rel_{[Lab]} \rangle$ is called an *algorithmic algebra of the computer.*

There are many different sorts of *storage* (or memory) of a computer, which need not be distinguished for the purpose of this paper, but which are necessary for further purposes. Therefore let *Loc* be the set of symbols, called *locations of the computer*, and let us distinguish only two sorts of them:

(2.1) $Loc = Lab \cup Var$, $Lab \cap Var = \emptyset$, $Var \cap Obj = \emptyset$, *where the elements from Lab, Var are called labels, (individual) variables, respectively. This distinguishing concerns the use of locations within a program, and not the hardware.*

In order to define all possible software of the computer the concept of its *vocabulary Voc* must be introduced:

(2.2) $Voc = SymbOpr \cup SymbRel \cup Lab \cup Var \cup Aux$, *where* $Aux = \{,; =: ( ) \langle \rangle [\ ] \gamma\}$ *and each two of the five sets of symbols are disjoint.*

The symbols from *SymbOpr*, *SymbRel* are called *symbols of operations*, *symbols of relations*, respectively, and a one-to-one mapping *int*, called *interpretation*, is assumed such that

(2.3) Domain $int = SymbOpr \cup SymbRel$,
$int(\text{“}f^{(n)}\text{”}) = f^{(n)}$ *where* $\text{“}f^{(n)}\text{”} \in SymbOpr$ *and* $f^{(n)} \in Opr$,
$int(\text{“}r^{(n)}_{(k)}\text{”}) = r^{(n)}_{(k)}$ *where* $\text{“}r^{(n)}_{(k)}\text{”} \in SymbRel$ *and* $r^{(n)}_{(k)} \in Rel$.

The following three main types of uninterpreted *machine commands* are distinguished:

(*operational*) $\text{“}f^{(n)}\text{”}(x_1, x_2, \ldots, x_n) =: x_0$ where $\text{“}f^{(n)}\text{”} \in SymbOpr$ and $x_i \in Var$ for each $i = 0, 1, \ldots, n$;

(*decisional*) $\text{“}r^{(n)}_{[a_1,a_2,\ldots,a_k]}\text{”}(x_1, x_2, \ldots, x_n)$ where $\text{“}r^{(n)}_{(k)}\text{”} \in SymbRel$, $x_i \in Var$ for each $i = 1, 2, \ldots, n$, and $a_j \in Lab$ for each $j = 1, 2, \ldots, k$;

(*stopping*) *STOP*;

obviously, $f^{(n)}(x_1, \ldots, x_n) =: x_0$ and $r^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$ are *interpreted machine commands.*

Everywhere further we shall not distinguish between interpreted and uninterpreted case symbolically, because any confusion can be avoided by the context.

Among the operational commands the following type of commands may be distinguished:

(*restoring*) $y_0 =: x_0$ where $x_0, y_0 \in Var$,

if the omitted operational symbol was a symbol of unary identity operation. Using the restoring commands all kinds of shifting from one sort of memory to another one can be described.

Let *Com* be the set of all commands, i.e. strings over the vocabulary *Voc*.

The occurrence of the variable x_0 in the operational and restoring command is called a *defining occurrence*, and all other occurrences of variables in all types of commands are called *applied occurrences*.

A *state of storage* is a function $\sigma \in \Sigma_{AA}$ where

$$\Sigma_{AA} =_{\mathrm{df}} (Obj \cup Loc \cup Com)^{Loc}. \tag{2.4}$$

The different states of storage are used for description of the process of computation, because the activity of the computer consists in changing its state of storage step by step.

On the other hand the symbol "γ" belongs to the vocabulary *Voc*, and it may be used in the following two further types of commands (see [5]):

(*left 2-restoring*) $\quad \gamma(y_0) =: x_0 \quad$ where $\quad x_0, y_0 \in Var.$

(*right 2-restoring*) $\quad y_0 =: \gamma(x_0).$

In this case "γ" is denoting "*the current state of storage*", and the only activity of the storage, which is allowed by the hardware of the computer, is to evaluate the current state. If $\sigma \in \Sigma_{AA}$, $x \in Var$ and $\sigma(x) \in Var$, then $\sigma^2(x) = \sigma(\sigma(x))$ is defined which allows to describe the "*ref-mechanism*" in ALGOL 68 [17], but it is not the aim of this paper.

In the following section we do not intend to investigate the structured objects and to evaluate the functions of structured types. Therefore we do not need to assume that the locations are natural numbers, i.e. the assumed storage has no special structure, and further, we do not intend to use "table functions" which are the functions from locations to objects, and by which the algorithmic algebra may be enriched.

A *machine program* is a finite sequence of machine commands which are stored in certain locations = labels. Therefore a pair $\langle b, C \rangle$ is called a *labelled command*, where $b \in Lab$ and $C \in Com$.

3. ALGORITHMS OVER AN ALGORITHMIC ALGEBRA

Let $AA = \langle Obj, Lab, Opr, Rel_{[Lab]} \rangle$ be an algorithmic algebra of a computer and let *Var* be a set of individual variables such that (1.1) and (2.1) are satified. Further let *Voc* be the vocabulary introduced in Sect. 2. With respect to computers the term algorithm may be replaced by the term program.

A finite sequence $A = (K^{(1)}, \cup^{(2)}, \ldots, K^{(N)})$ of labelled commands $K^{(i)} = \langle b^{(i)}, C^{(i)} \rangle$ is called an *algorithm over Voc* if the following two requirements are satisfied:

(3.1) $\quad b^{(i)} \neq b^{(j)}$ *where* $i \neq j$, *for each* $i, j = 1, 2, \ldots, N$;

(3.2) *there exists at least one finite sequence* $lb = (K_1, K_2, \ldots, K_q)$ *of labelled commands from* A *(i.e.* $K_i = K^{(j)}$*) such that:*

(i) $K_1 = K^{(1)}$;

(ii) $K_q = K^{(j)}$ *where* $1 \leqq j \leqq N$, *and* $C^{(j)} = \text{STOP}$;

(iii) *there exists an integer* i, $1 \leqq i < q$ *such that* $K_i = K^{(j)}$ *where* $1 \leqq j \leqq N$ *and* $C^{(j)}$ *is an operational (or a restoring) command;*

(iv) *if* $K_i = K^{(j)}$ *where* $1 \leqq i < q$ *then one of the following two possibilities must occur: either* $C^{(j)}$ *is an operational command and then* $1 \leqq j < N$ *and* $K_{i+1} = K^{(j+1)}$, *or* $C^{(j)} = r^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$ *is a decisional command and then there are indices* h, $1 \leqq h \leqq k$ *and* p, $1 \leqq p \leqq N$ *such that* $a_h = b^{(p)}$ *and* $K_{i+1} = K^{(p)}$.

For theoretical purposes it is usefull to admit also *infinite algorithms* when the given sequence A is infinite.

A sequence lb, the existence of which is required in (3.2), is called *labelled branch of* A if it satisfies (3.2i, ii, iii and iv). Let LB_A be the set of all labelled branches of A.

A variable which occurs in a labelled branch $lb \in LB_A$ is called *input variable of* lb, if its first occurrence (from left to right) in the string lb is an applied occurrence. A variable which has its defining occurrence in the labelled branch $lb \in LB_A$ is called *output variable of* lb, if after the last defining occurrence (the first from right) of this variable in lb either no its occurrence appear or there appear only its applied occurrences in decisional commands. Let Inp_{lb}, $Outp_{lb}$ be the set of all input, output variables of lb, respectively. Further let us define

$$(3.3) \qquad Inp_A = \bigcup_{lb \in LB_A} Inp_{lb} \quad \text{and} \quad Outp_A = \bigcup_{lb \in LB_A} Outp_{lb},$$

where the variables from Inp_A, $Outp_A$ *are called input, output variables of the algorithm* A.

Lemma 3.1. *If* A *is an algorithm over Voc then* $LB_A \neq \emptyset$ *and therefore* $Inp_A \neq \emptyset \neq Outp_A$.

The proof follows immediately from the definitions.

It should be noted that using new restoring commands and new variables further output variables of an algorithm may be introduced.

The *application of the algorithm* $A = (K^{(1)}, \ldots, K^{(N)})$ to the *initial state* $\sigma_0 \in \Sigma_{AA}$ in the *algorithmic algebra* AA is an *activity*, which starts and can, but need not terminate, and which consists of elementary *steps* corresponding to particular commands: the first step consists in application of $K_1 =_{\text{df}} K^{(1)}$ to σ_0, and in the i-th step, $1 \leqq i$ by the application of $K_i =_{\text{df}} K^{(j)}$, $1 \leqq j \leqq N$, to the state σ_{i-1}, the next state σ_i

and the next command K_{i+1} are determined as follows:

(3.4) (i) *if* $C^{(j)} = f^{(n)}(x_1, \ldots, x_n) =: x_0$ *then*

a) $\sigma_i(x_0) =_{\mathrm{df}} int\, f^{(n)}(\sigma_{i-1}(x_1), \ldots, \sigma_{i-1}(x_n))$, $\sigma_i(z) =_{\mathrm{df}} \sigma_{i-1}(z)$ *for each* $z \in Var$ *such that* $z \neq x_0$;

b) $K_{i+1} =_{\mathrm{df}} K^{(j+1)}$;

(ii) if $C^{(j)} = r^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$ *then*

a) $\sigma_i =_{\mathrm{df}} \sigma_{i-1}$;

b) $K_{i+1} =_{\mathrm{df}} K^{(h)}$ *where* h *is the smalest integer such that* $b^{(h)} = int\, r^{(n)}_{[a_1,\ldots,a_k]}(\sigma_{i-1}(x_1), \ldots, \sigma_{i-1}(x_n))$;

(iii) *if* $C^{(j)} =$ STOP *then* K_i *is the last step and the activity finishes; the application is called stopped and* σ_{j-1} *is called the result state which corresponds to the initial state* σ_0;

(iv) *if in the case* (i) *either* $j = N$ *or* $(\sigma_{i-1}(x_1), \ldots, \sigma_{i-1}(x_n)) \notin \notin \mathrm{Domain}\,(int\, f^{(n)})$, *or in the case* (ii) *either* $a_s \neq b^{(h)}$ *for each* $s = 1, 2, \ldots, k$ *and each* $h = 1, 2, \ldots, N$, *or* $(\sigma_{i-1}(x_1), \ldots, \sigma_{i-1}(x_n)) \notin \notin \mathrm{Domain}\,(int\, r^{(n)}_{[a_1,\ldots,a_k]})$ *then* K_i *is the last step and the activity finishes; the application is called finished;*

(v) *it may happen that the activity does not finish and then the application is called not finished.*

Let $LB_{AA}(A, \sigma_0)$, $Cpt_{AA}(A, \sigma_0)$ denote the unique labelled branch $(K_1, \ldots, K_q)$, the sequence of states $(\sigma_0, \sigma_1, \ldots, \sigma_{q-1})$, respectively, which are determined by a stopped application of A to σ_0 in AA according to (3.4). $LB_{AA}(A, \sigma_0)$ is called a *labelled branch applied in* AA and $Cpt_{AA}(A, \sigma_0)$ is called *computation in* AA. Let $LB_{A,AA}$ be the set of all labelled branches applied in AA of the algorithm A. Obviously $LB_{A,AA} \subset LB_A$ for each AA.

The *state function* $\mathbf{S}_{A,AA}$, depending on the algorithm A and on the algorithmic algebra AA, is defined as follows:

(3.5) $\mathrm{Domain}\, \mathbf{S}_{A,AA} =_{\mathrm{df}} \{\sigma_0 \in \Sigma_{AA};\ Cpt_{AA}(A, \sigma_0)\ exists\}$,

$\mathrm{Range}\ \ \mathbf{S}_{A,AA} =_{\mathrm{df}} \{\sigma_{q-1};$ *there exists* $\sigma_0 \in \Sigma_{AA}$ *and* $Cpt_{AA}(A, \sigma_0) = (\sigma_0, \ldots, \sigma_{q-1})\}$,

$\mathbf{S}_{A,AA}(\sigma_0) = \sigma_{q-1}$ *where* $Cpt_{AA}(A, \sigma_0) = (\sigma_0, \ldots, \sigma_{q-1})$.

This state function $\mathbf{S}_{A,AA}$ is a transition function of a nonsequential machine (see [2]) or of a jumping machine (see [3]), and this transition function is the starting concept of Pawlak's approach (see [14]). The state function is not investigated here.

In the next Sect. 4 the following assertion will be proved easily:

Lemma 3.2. *Let A be an algorithm and let $\sigma_0, \sigma'_0 \in \Sigma_{AA}$ be two initial states such that $\sigma_0(x) = \sigma'_0(x)$ for each $x \in Inp_A$. Then* a) *$LB_{AA}(A, \sigma_0)$ exists $\Leftrightarrow$ $LB_{AA}(A, \sigma'_0)$ exists;* b) *if $LB_{AA}(A, \sigma_0)$ exists then $LB_{AA}(A, \sigma_0) = LB_{AA}(A, \sigma'_0)$ and $\sigma_{q-1}(y) = \sigma'_{q-1}(y)$ for each $y \in Outp_{lb}$ where $lb = LB_{AA}(A, \sigma_0)$ and $Cpt_{AA}(A, \sigma_0) = (\sigma_0, \ldots, \sigma_{q-1})$ and $Cpt_{AA}(A, \sigma'_0) = (\sigma'_0, \ldots, \sigma'_{q-1})$.*

The assertion 3.2 allows the following definition of the *system of functions* $\mathbf{f}_{A,AA}\langle Inp_A, Outp_A\rangle$ which is *computable* (or algorithmically definable) *by A in AA*:

(3.6) *the system $\mathbf{f}_{A,AA}\langle Inp_A, Outp_A\rangle$ contains $|Outp_A|$ functions of $|Inp_A|$ variables ($|N|$ is the number of elements of the set N); the functions of the system are distinguished by the particular output variables; if $\mathbf{f}_{A,AA,y} \in \mathbf{f}_{A,AA}$ where $y \in Outp_A$, and $Inp_A = \{x_1, x_2, \ldots, x_r\}$ then $\mathbf{f}_{A,AA,y}(\sigma_0(x_1), \sigma_0(x_2), \ldots, \sigma_0(x_r)) =_{\mathrm{df}} \sigma_{q-1}(y) \Leftrightarrow LB_{AA}(A, \sigma_0)$ exists and $Cpt_{AA}(A, \sigma_0) = (\sigma_0, \sigma_1, \ldots, \sigma_{q-1})$, where one fixed ordering $(x_1, x_2, \ldots, x_r)$ of all input variables is assumed (which is not determined by Inp_A itself).*

It can happen that $\mathbf{f}_{A,AA,y} = \mathbf{f}_{A,AA,y'}$ for some $y, y' \in Outp_A$, where $y \neq y'$. This is the reason why $\mathbf{f}_{A,AA}$ is considered as a system and not as a set of functions $\mathbf{f}_{A,AA,y}$, the only distinguishing thing among which is the output variable y.

If we are not interested in the input and output variables of the particular algorithms then the following equivalence of two systems may be introduced: one writes $\mathbf{f}_{A,AA} \equiv \mathbf{f}_{A',AA}$ if there exist two one-to-one mappings π_{inp} and π_{outp} such that Domain $\pi_{inp} = Inp_A$, Range $\pi_{inp} = Inp_{A'}$, Domain $\pi_{outp} = Outp_A$, Range $\pi_{outp} = Outp_{A'}$ and

(3.7) $\mathbf{f}_{A,AA,y}(x_1, x_2, \ldots, x_r) = \mathbf{f}_{A',AA,\pi_{outp}y}(\pi_{inp}x_1, \pi_{inp}x_2, \ldots, \pi_{inp}x_r)$ *for each $y \in Outp_A$, where x_i and $\pi_{inp}x_i$ are the corresponding variables for each $i = 1, 2, \ldots, r$.*

The usual *functional equivalence* $\equiv_{AA}$ of two algorithms A and A' in the algorithmic algebra AA is defined as follows:

(3.8) $$A \equiv_{AA} A' \Leftrightarrow \mathbf{f}_{A,AA} \equiv \mathbf{f}_{A',AA}\,.$$

Finally let Alg_{Voc} be the set of all algorithms over the vocabulary Voc. Then

(3.9) $$\mathbf{fct}_{AA} = \{\mathbf{f}_{A,AA,y};\ \mathbf{f}_{A,AA,y} \in \mathbf{f}_{A,AA},\ A \in Alg_{Voc}\}$$

and

(3.10) $$\mathbf{Fct}_{AA} = \{\mathbf{f}_{A,AA};\ A \in Alg_{Voc}\}$$

represent the *functional power* of the algorithmic algebra AA when the algorithms of Alg_{Voc} are used.

If the algorithmic algebra AA_0 is defined as follows: Obj_0 is the set of all natural numbers, Opr_0 contains only the successor function, Rel_0 contains only the equality relation, and Var_0, Lab_0 contain as many symbols as it is necessary, then the following theorem is proved in [6].

Theorem 3.3. $\mathbf{fct}_{AA_0}$ *contains all partial recursive functions.*

In the following section we shall assume that each algorithm $A = (K^{(1)}, \ldots, K^{(N)})$ satisfies the following requirement:

(3.2) (v) *each* $K^{(i)}$, *where* $1 \leqq i \leqq N$, *is contained at least in one labelled branch of* A.

The labelled commands of A which do not satisfy (3.2v) are called (*syntactically*) *superflous*, because they may be omitted (or added) without any effect to the set LB_A and therefore to the computable system $\mathbf{f}_{A,AA}$.

Lemma 3.4. *If* A^* *arises from the algorithm* A *over Voc by omission of all its superflous labelled commands, then* $LB_{A^*} = LB_A$ *and* $A^* \equiv_{AA} A$ *in each algorithmic algebra* AA *over Voc.*

The proof follows by the definition and by (3.4) and (3.6).

In fact the inductive definition of application (3.4), and all other inductive definitions, concerns only the labelled commands which are not superflous. E.g. if the last command of an algorithm is operational, then it is superflous.

In order to be able to speak about the particular symbols from *Voc*, which occur in a command $C \in Com$ the following auxiliary functions [4] must be introduced:

(3.11) (i) *if* C *is an operational, decisional command then* **symb** C *denotes the symbol of operation, decision, respectively, which occurs in* C; *if* C *is the stopping command then* **symb** C = STOP; *if* C *is a restoring command then* **symb** $C = Id$, *where "Id" is a new symbol (which may be added to SymbOpr) corresponding to the unary identity function;*

(ii) $\mathbf{arg}_i\ C$ *denotes the i-th (from left to right) applied occurrence of a variable in* C *for* $i = 1, 2, \ldots$;

(iii) **res** C *denotes the (unique if any) defining occurrence of a variable in* C;

(iv) $\mathbf{lab}_j\ C$ *denotes the j-th (from left to right) occurrence of a label in* C *for* $j = 1, 2, \ldots$

Obviously the defined functions are partial, e.g. **res** C and $\mathbf{lab}_j\ C$ cannot be defined simultaneously.

Further let **var** C, **lab** C be the set of all variables, labels, respectively, which occur in C. And if $K = \langle l, C \rangle$, $A \in Alg_{Voc}$, $lb \in LB_A$, etc., then the set of variables **var** K, **var** A, **var** lb, etc., and the sets of labels **lab** K, **lab** A, **lab** lb, etc., are defined in a similar way.

4. SCOPES OF VARIABLES. WIDTH

In an algorithm $A = (K^{(1)}, \ldots, K^{(N)})$ without superflous commands let us assume that all occurrences of all variables are numbered by right upperscripts from the left to the right, i.e. the k-th occurrence of the variable x is denoted as x^k. If $x \in Inp_A$ then the *zero-th occurrence* x^0 is added (for easier definitions of further notions).

Let us denote:

(4.1) *$Oc_A(x)$ is the set of all numbered occurrences of x in A (inclusively the zero-th occurrence if $x \in Inp_A$);*

$$Oc_A = \bigcup_{x \in var A} Oc_A(x);$$

$$oc_A(x) = |Oc_A(x)| \text{ and } oc_A = \sum_{x \in var A} oc_A(x).$$

In each labelled branch $lb = (K_1, \ldots, K_q)$ of A let us assume that all occurrences of all variables are numbered by left upperscripts from the left to the right, i.e. the h-th occurrence of the variable x is denoted as ${}^h x$. If $x \in Inp_{lb}$ then the *zero-th occurrence* ${}^0 x$ is added.

Let us denote:

(4.2) *$Oc_{lb}(x)$ is the set of all numbered occurrences of x in lb (inclusively ${}^0 x$ if $x \in Inp_{lb}$);* $Oc_{lb} = \bigcup_{x \in var lb} Oc_{lb}(x)$; $oc_{lb}(x) = |Oc_{lb}(x)|$ *and* $oc_{lb} = \sum_{x \in var lb} oc_{lb}(x)$.

Each zero-th occurrence is considered as a defining occurrence.

It is clear that there exist a mapping $\varkappa_{lb}$ such that Domain $\varkappa_{lb} = Oc_{lb}$, Range $\varkappa_{lb} \subset$ $\subset Oc_A$, and ${}^h x \in Oc_{lb}$ and $\varkappa_{lb}({}^h x) = x^k \in Oc_A$ either identify the same occurrence of the variable x in A, or both are zero-th occurrences. Therefore, obviously,

(4.3) *${}^h x \in Oc_{lb}$ is a defining (applied) occurrence of $x \Leftrightarrow \varkappa_{lb}({}^h x) \in Oc_A$ is a defining (applied) occurrence of x.*

The *scope of the defining occurrence* ${}^h x \in Oc_{lb}$ in the labelled branch $lb \in LB_A$ is the set $SC_{lb}({}^h x)$ of the following occurrences in lb:

(4.4) $SC_{lb}({}^h x) = \{{}^{h+i} x;\ 0 \leqq i \leqq p$ *and p is the maximal integer such that* ${}^{h+i} x$ *is an applied occurrence for* $i = 1, 2, \ldots, p\}$.

The number $|SC_{lb}({}^h x)| - 1$ is called the *length* of the scope $Sc_{lb}({}^h x)$ and ${}^{h+q} x$, where q is the length of the scope, is called *maximal occurrence* of the scope. Thus the maximal occurrence of the scope having length equal zero is a defining occurrence.

The first occurrence ${}^1 x \in Oc_{lb}$, which is an applied occurrence, is called the *input occurrence of x in lb*, and the last occurrence ${}^k x \in Oc_{lb}$, where $k = |oc_{lb}(x)|$, which is a defining one, is called the *output occurrence of x in lb.*

If ${}^k x$ is a maximal, input, output occurrence of x in lb then $\varkappa_{lb}({}^k x)$ is called *maximal, input, output occurrence of* x *in* A, respectively.

The scope of the defining occurrence $x^k \in Oc_A$ in the algorithm A is the set $SC_A(x^k)$ of the occurrences in A:

$$(4.5) \qquad SC_A(x^k) = \{\varkappa_{lb}({}^l x);\ lb \in LB_A,\ {}^l x \in SC_{lb}({}^h x) \quad \text{and} \quad \varkappa_{lb}({}^h x) = x^k\}\,.$$

The number $|SC_A(x^k)| - 1$ is called the *length* of the scope $SC_A(x^k)$.

Remark 4.1. *There exists at most one input and output occurrence of a variable in* $lb \in LB_A$, *but there can exist more than one input and output occurrences of a variable in* A. *There exists at most one maximal occurrence of a scope in* lb, *but there can exist more than one maximal occurrences of a scope in* A.

The proof follows from the definitions.

A labelled operational command $K^{(p)}$ of the algorithm $A = (K^{(1)}, \ldots, K^{(N)})$, $1 \leqq p \leqq N$, is called (*semantically*) *superflous* if in each labelled branch $lb = (K_1, \ldots, K_q) \in LB_A$ such that there exist $K_i = K^{(p)}$ and $\mathbf{res}^\sharp\, C_i$ is not an output occurrence, the following requirement is satisfied:

$$(4.6) \qquad \textit{the length of the scope } SC_{lb}(\mathbf{res}^\sharp\, C_i) \textit{ is zero}\,,$$

where the function "$\mathbf{res}^\sharp$" assignes the corresponding numbered occurrence of Oc_{lb} to C_i (while "**res**" of (3.11) assigned the variable itself). In a similar way the other functions "$\mathbf{arg}_i^\sharp$" and "$\mathbf{lab}_j^\sharp$" will be used.

Lemma 4.2. *Let* A^* *arise from* $A = (K^{(1)}, \ldots, K^{(N)}) \in Alg_{Voc}$, *when* A *is without syntactically superflous commands, by omitting of a* (*semantically*) *superflous operational command* $K^{(p)} = \langle b^{(p)}, C^{(p)}\rangle$ *where* $1 \leqq p < N$, *and by the following change of each decisional command* $K^{(s)}$, $1 \leqq s \leqq N$: *if* $\mathbf{symb}\ C^{(s)} = r^{(n)}_{[a_1,\ldots,a_k]}$ *then each* a_i, $1 \leqq i \leqq k$, *such that* $a_i = b^{(p)}$, *should be replaced by the label* $b^{(p+1)}$. *Then* $A^* \equiv_{AA} A$ *in each algorithmic algebra* AA.

The proof is obvious.

Lemma 4.3. *If* ${}^h x \in Oc_{lb}$ *where* $lb \in LB_A$, $A \in Alg_{Voc}$ *then there exists exactly one defining occurrence* ${}^k y \in Oc_{lb}$ *such that* ${}^h x \in SC_{lb}({}^k y)$, *and, obviously,* $y = x$.

If $x^h \in Oc_A$ *then there exists at least one defining occurrence* $x^k \in Oc_A$ *such that* $x^h \in SC_A(x^k)$.

Proof. If ${}^h x \in Oc_{lb}$ is a defining occurrence then by (4.4) ${}^h x \in SC_{lb}({}^h x)$. If ${}^h x$ is an applied occurrence then either there does not exist a defining occurrence ${}^k x$ such that $1 \leqq k < h$; in this case x is an input variable and therefore ${}^0 x \in Oc_{lb}$ and by (4.4) ${}^h x \in SC_{lb}({}^0 x)$; or there exists the mentioned defining occurrence ${}^k x$ with the

maximal number k; in this case by (4.4) ${}^{h}x \in SC_{lb}({}^{k}x)$. Thus each occurrence apears at least in one scope. The assumption $SC_{lb}({}^{h}x) \cap SC_{lb}({}^{k}x) \neq \emptyset$ for $h \neq k$ leads to a contradiction by (4.4) immediately.

The second part follows by (4.5) directly.

The set of all scopes in A is considered as the set of vertices of the undirected *scope graph* the edges of which are defined as follows: two scopes are joined by an edge if their set theoretical intersection is not void. Then the set theoretical union of all scopes, which belong to the same connected component of the scope graph, is denoted by $FaSC_A$ and called a *family of scopes.* It is clear that using the families of scopes the last assertion of Lemma 4.3 may be extended.

Let $A = (K^{(1)}, \ldots, K^{(N)})$, $A' = (K'^{(1)}, \ldots, K'^{(N')})$ be two algorithms over *Voc*, where $K^{(p)} = \langle b^{(p)}, C^{(p)} \rangle$ and $K'^{(p)} = \langle b'^{(p)}, C'^{(p)} \rangle$. We say that A *is similar to* A', and we write A ***sim*** A', if the following requirements are satisfied:

(4.7) (i) $N = N'$;

(ii) $\mathbf{symb}\ C^{(p)} = \mathbf{symb}\ C'^{(p)}$ *for* $p = 1, 2, \ldots, N$;

(iii) *if the mapping* ψ *is defined as follows:* $\psi(b^{(p)}) = b'^{(p)}$ *for* $p = 1, 2, \ldots, N$, *where* Domain $\psi = \{b^{(1)}, \ldots, b^{(N)}\}$ *and* Range $\psi = \{b'^{(1)}, \ldots, b'^{(N)}\}$, *then* $\mathbf{lab}_j\ C'^{(p)} = \psi(\mathbf{lab}_j\ C^{(p)})$ *for each* $j = 1, 2, \ldots$, *and each* p *such that* $1 \leqq p \leqq N$ *and* $C^{(p)}$ *is a decisional command;*

(iv) *if the mapping* φ *is defined as follows:* $\varphi = \bigcup_{p=1}^{N} \varphi_p$, *where* $\mathbf{arg}_i^{\sharp}\ C'^{(p)} = \varphi_p(\mathbf{arg}_i^{\sharp}\ C^{(p)})$ *for each* $i = 1, 2, \ldots$, *and each* p *such that* $1 \leqq p \leqq N$ *and* $C^{(p)}$ *is not the stopping command,* $\mathbf{res}^{\sharp}\ C'^{(p)} = \varphi_p(\mathbf{res}^{\sharp}\ C^{(p)})$ *for each* p *such that* $1 \leqq p \leqq N$ *and* $C^{(p)}$ *is an operational command, then:*

(*) $\mathbf{arg}_i^{\sharp}\ C^{(p)}$ *and* $\mathbf{arg}_j^{\sharp}\ C^{(p)}$ *are occurrences of the same variable* $\Leftrightarrow$ $\Leftrightarrow \varphi_p(\mathbf{arg}_i^{\sharp}\ C^{(p)})$ *and* $\varphi_p(\mathbf{arg}_j^{\sharp}\ C^{(p)})$ *are occurrences of the same variable, where* $1 \leqq p \leqq N$;

(**) $\mathbf{arg}_i^{\sharp}\ C^{(p)}$ *and* $\mathbf{arg}_j^{\sharp}\ C^{(r)}$ *are input occurrences of the same variable* $\Leftrightarrow$ $\Leftrightarrow \varphi_p(\mathbf{arg}_i^{\sharp}\ C^{(p)})$ *and* $\varphi_r(\mathbf{arg}_j^{\sharp}\ C^{(r)})$ *are input occurrences of the same variable, where* $1 \leqq p, r \leqq N$;

(***) $x^k \in SC_A(x^h) \Leftrightarrow \varphi(x^k) \in SC_{A'}(\varphi(x^h))$ *for all* $x^k, x^h \in Oc_A$, *where* φ *is extended to the zero occurrences in accordance to* (**) *uniquely;*

(****) $\mathbf{res}^{\sharp}\ C^{(p)}$ *and* $\mathbf{res}^{\sharp}\ C^{(r)}$ *are output occurences of the same variable* $\Leftrightarrow$ $\Leftrightarrow \varphi_p(\mathbf{res}^{\sharp}\ C^{(p)})$ *and* $\varphi_r(\mathbf{res}^{\sharp}\ C^{(r)})$ *are output occurences of the same variable, where* $1 \leqq p, r \leqq N$.

Theorem 4.4. *The relation* ***sim*** *is an equivalence relation in* Alg_{Voc}.

Proof. The relation ***sim*** is the intersection of four binary relations, which are defined by four requirements (4.7i, ii, iii and iv). The first two of these relations are equivalence relations evidently. The third concerns a one-to-one mapping ψ, and therefore it is clear, that it is an equivalence relation also. Finally φ is a one-to-one mapping again, and therefore one easy proves that the identity satisfies all four conditions (*)–(****); further if φ satisfies these conditions then also φ^{-1} satisfies them; and if φ_1 and φ_2 satisfy these conditions and Range φ_1 = Domain φ_2, then $\varphi_2\varphi_1$ satisfies them also. Thus the reflexivity, symmetry and transitivity of the fourth relation is proved.

Let us introduce the following sets of labels which correspond to scopes and to families of scopes:

(4.8) $LaSC_{lb}({}^hx) = \{l_j; \langle l_j, C_j\rangle = K_j,\ lb = (K_1, \ldots, K_q),\ 0 \leqq i < j \leqq i + k \leqq q,$ *where* ${}^hx$ *belongs to* K_i *if* $h > 0$ *and* $i > 0$, *while* $i = 0$ *if* $h = 0$, *and the maximal occurrence of this scope belongs to* $K_{i+k}\}$;

$$(4.9) \qquad LaSC_A(x^k) = \bigcup_{\substack{{}^hx_{lb}({}^hx)=x^k \\ lb \in LB_A}} LaSC_{lb}({}^hx)\,;$$

$$(4.10) \qquad LaFaSC_A(x^k) = \bigcup_{x^h \in FaSC_A} LaSC_A(x^h)\,.$$

Lemma 4.5. *If* A' *arises from* A *by the renaming* ϱ *of occurrences of variables such that* (4.11) *is satisfied, then* A' ***sim*** A, *where*

(4.11) (i) $x^k \in FaSC_A(x^h) \Rightarrow \varrho(x^k)$ *and* $\varrho(x^h)$ *are occurrences of the same variable;*

(ii) $LaFaSC_A(x^h) \cap LaFaSC_A(y^k) \neq \emptyset \Rightarrow \varrho(x^k)$ *and* $\varrho(x^h)$ *are occurrences of two different variables.*

(iii) x^k *and* y^h *are output occurences of the same variable* $\Leftrightarrow \varrho(x^k)$ *and* $\varrho(y^h)$ *are output occurences of the same variable.*

Proof. It is clear that A' and A satisfy (4.7i, ii and iii). Therefore it remains to prove that φ, defined in (4.7iv), satisfies (*)–(****). If $\boldsymbol{arg}_i^{\#}\, C^{(p)}$ and $\boldsymbol{arg}_j^{\#}\, C^{(p)}$ are occurrences of the same variable then they belong to the same family of scopes and therefore by (4.11i) $\varphi_p(\boldsymbol{arg}_i^{\#}\, C^{(p)})$ and $\varphi_p(\boldsymbol{arg}_j^{\#}\, C^{(p)})$ are occurrences of the same variable. On the other hand if $\boldsymbol{arg}_i^{\#}\, C^{(p)}$ and $\boldsymbol{arg}_j^{\#}\, C^{(p)}$ are occurrences of different variables then they belong to different families of scopes, such that their sets of labels have not void intersection, and therefore by (4.11ii) also $\varphi_p(\boldsymbol{arg}_i^{\#}\, C^{(p)})$ and $\varphi_p(\boldsymbol{arg}_j^{\#}\, C^{(p)})$ have not void intersection, and therefore by (4.11ii) also $\varphi_p(\boldsymbol{arg}_i^{\#}\, C^{(p)})$ and $\varphi_p(\boldsymbol{arg}_j^{\#}\, C^{(p)})$ are occurrences of different variables. Thus (*) is proved.

The renaming ϱ does not change the property of to be an input occurrence, and therefore (**) is also satisfied.

The condition (***) may be proved by contradiction. and the remaining requirement (****) is only a transcription of (4.11iii), when one uses the identity $\varphi(x^k) = \varrho(x^k)$.

The *family graph* of an algorithm A is the undirected graph G_A the vertices of which are families of scopes not containing output occurrences, and an edge connects $FaSC_A(x^h)$ with $FaSC_A(y^k)$ if $LaFaSc(x^h) \cap LaFaSC_A(y^k) \neq \emptyset$.

Theorem 4.6. *If $\chi(G_A)$ is the chromatic number of the family graph G_A of the algorithm A and* **Sim** (A) *is the class of all algorithms A' such that A'* **sim** *A, then*

$$\min |\textbf{var}\ A'| = \max \{\chi(G_A), |Outp_A|\}\ . \tag{4.12}$$

If $lb = (K_1, \ldots, K_q) \in LB_A$ then let us define the sequence $Med_{lb} = (M_0, M_1, \ldots, M_{q-1})$ of *sets of intermediate variables* as follows. If $K_i = \langle b_i, C_i \rangle$, $1 \leqq i \leqq q$, then let $H_{lb}(C_i)$ be the set of all variables the occurrences of which in C_i (within lb) are the maximal occurrences. Now

$$M_0 =_{\text{df}} Inp_{lb},\ M_{i+1} =_{\text{df}} (M_i - H_{lb}(C_i)) \cup \{\textbf{res}\ C_i\}\ \textit{for}\ i = 1, 2, \ldots, q-2. \tag{4.12}$$

The *width* **wid** (lb) *of the labelled branch lb* is defined by the requirement $\textbf{wid}\,(lb) = \max_{M_i \in Med_{lb}} |M_i|$, and the *width* **wid** (A) *of the algorithm A* is defined as follows: $\textbf{wid}\,(A) = \max_{lb \in LB_A} \textbf{wid}\,(lb)$.

The sets of intermediate variables give perfect knowledge concerning locations of all intermediate results after each step of computation. The sets of intermediate variables are necessary if the "interruptions" are described, and are usefull for description the "garbage collection".

5. SIMILARITY OF STATES AND COMMANDS

A one-to-one mapping π such that

(5.1) Domain $\pi \subset Var \cup Lab$, Range $\pi|_{Var} \subset Var$, Range $\pi|_{Lab} \subset Lab$, *is called a similarity. We say that the state σ is similar to the state σ' in the similarity π, and we write $\sigma \pi \sigma'$, if*

$$\sigma(x) = \sigma'(x) \quad \textit{for each} \quad x \in \text{Domain}\ \pi\ . \tag{5.2}$$

The following assertions follow immediately from the definitions:

$$\sigma \pi \sigma' \Leftrightarrow \sigma' \pi^{-1} \sigma\ ,\ \textit{and}\ \ \sigma \pi \sigma'\ \ \textit{and}\ \ \pi' \subset \pi \Rightarrow \sigma \pi' \sigma'\ . \tag{5.3}$$

We say that the command C *is similar to the command* C' *in the similarity* π, and we write $C\pi C'$, if

(5.4) (i) $\mathbf{symb}\ C' = \mathbf{symb}\ C$;

(ii) $\mathbf{arg}_i\ C' = \pi(\mathbf{arg}_i\ C)$ *for* $i = 1, 2, \ldots$;

(iii) $\mathbf{lab}_j\ C' = \pi(\mathbf{lab}_j\ C)$ *for* $j = 1, 2, \ldots$

Lemma 5.1. *If* $C\pi C'$ *then*

(5.4*) (i) $\mathbf{arg}_i\ C = \mathbf{arg}_j\ C \Leftrightarrow \mathbf{arg}_i\ C' = \mathbf{arg}_j\ C'$ *where* $i, j = 1, 2, \ldots$;

(ii) $\mathbf{lab}_i\ C = \mathbf{lab}_j\ C \Leftrightarrow \mathbf{lab}_i\ C' = \mathbf{lab}_j\ C'$ *where* $i, j = 1, 2, \ldots$

The proof follows directly from the definitions.

If C and C' are operational (or restoring) commands and $C\pi C'$ holds, then the new mapping π^*, called the *star similarity of* π, arises from π as follows:

(5.5) $\pi^*(\mathbf{res}\ C) =_{\mathrm{df}} \mathbf{res}\ C'$ *and* $\pi^* x =_{\mathrm{df}} \pi x$ *for each* $x \in \mathrm{Domain}\ \pi - \{\mathbf{res}\ C\}$;

the domain and range of π^* is changing according to the following five possibilities:

(5.6) $\left.\begin{matrix}0\\0\end{matrix}\right)$ *if* $\left\{\begin{matrix}\mathbf{res}\ C \notin \mathrm{Domain}\ \pi\\ \mathbf{res}\ C' \notin \mathrm{Range}\ \pi\end{matrix}\right\}$ *then* $\left\{\begin{matrix}\mathrm{Domain}\ \pi^* =_{\mathrm{df}} \mathrm{Domain}\ \pi \cup \{\mathbf{res}\ C\}\\ \mathrm{Range}\ \pi^* =_{\mathrm{df}} \mathrm{Range}\ \pi \cup \{\mathbf{res}\ C'\}\end{matrix}\right\}$,

$\left.\begin{matrix}0\\1\end{matrix}\right)$ *if* $\left\{\begin{matrix}\mathbf{res}\ C \notin \mathrm{Domain}\ \pi\\ \mathbf{res}\ C' \in \mathrm{Range}\ \pi\end{matrix}\right\}$ *then*

$\left\{\begin{matrix}\mathrm{Domain}\ \pi^* =_{\mathrm{df}} (\mathrm{Domain}\ \pi - \{\pi^{-1}\ \mathbf{res}\ C'\}) \cup \{\mathbf{res}\ C\}\\ \mathrm{Range}\ \pi^* =_{\mathrm{df}} \mathrm{Range}\ \pi\end{matrix}\right\}$,

$\left.\begin{matrix}1\\0\end{matrix}\right)$ *if* $\left\{\begin{matrix}\mathbf{res}\ C \in \mathrm{Domain}\ \pi\\ \mathbf{res}\ C' \notin \mathrm{Range}\ \pi\end{matrix}\right\}$ *then*

$\left\{\begin{matrix}\mathrm{Domain}\ \pi^* =_{\mathrm{df}} \mathrm{Domain}\ \pi\\ \mathrm{Range}\ \pi^* =_{\mathrm{df}} (\mathrm{Range}\ \pi - \{\pi\ \mathbf{res}\ C\}) \cup \{\mathbf{res}\ C'\}\end{matrix}\right\}$,

$\left.\begin{matrix}1\\1\end{matrix}\right)$ *if* $\left\{\begin{matrix}\mathbf{res}\ C \in \mathrm{Domain}\ \pi\\ \mathbf{res}\ C' \in \mathrm{Range}\ \pi\end{matrix}\right\}$ *then either*

a) $\pi\ \mathbf{res}\ C \neq \mathbf{res}\ C'$ *and* $\left\{\begin{matrix}\mathrm{Domain}\ \pi^* =_{\mathrm{df}} \mathrm{Domain}\ \pi - \{\pi^{-1}\ \mathbf{res}\ C'\}\\ \mathrm{Range}\ \pi^* =_{\mathrm{df}} \mathrm{Range}\ \pi - \{\pi\ \mathbf{res}\ C\}\end{matrix}\right\}$,

or

b) $\pi\ \mathbf{res}\ C = \mathbf{res}\ C'$ *and* $\left\{\begin{matrix}\mathrm{Domain}\ \pi^* =_{\mathrm{df}} \mathrm{Domain}\ \pi\\ \mathrm{Range}\ \pi^* =_{\mathrm{df}} \mathrm{Range}\ \pi\end{matrix}\right\}$.

All five possibilities are clarified by the following examples, where it is assumed that Domain $\pi = \{x, y\}$, Range $\pi = \{x', y'\}$ and $\pi x = x'$, $\pi y = y'$:

$\left.\begin{matrix}0\\0\end{matrix}\right)$ $(x + y =: z)\,\pi\,(x' + y' =: z') \Rightarrow$ Domain $\pi^* = \{x, y, z\}$, Range $\pi^* = \{x', y', z'\}$;

$\left.\begin{matrix}0\\1\end{matrix}\right)$ $(x + y =: z)\,\pi\,(x' + y' =: y') \Rightarrow$ Domain $\pi^* = \{x, z\}$, Range π^* = Range π;

$\left.\begin{matrix}1\\0\end{matrix}\right)$ $(x + y =: x)\,\pi\,(x' + y' =: z') \Rightarrow$ Domain π^* = Domain π, Range $\pi^* = \{y', z'\}$;

$\left.\begin{matrix}1\\1\end{matrix}\right)$ a) $(x + y =: x)\,\pi\,(x' + y' =: y') \Rightarrow$ Domain $\pi^* = \{x\}$, Range $\pi^* = \{y'\}$;

b) $(x + y =: x)\,\pi\,(x' + y' =: x') \Rightarrow$ Domain π^* = Domain π, Range π^* = Range π.

It is easy to see that star similarity π^*, which arised from the similarity π by (5.5) and (5.6), is a one-to-one mapping, and therefore also a similarity.

Lemma 5.2. *Let be $C\,\pi\,C'$, where C is an operational command, and let $\pi x = y$, where $x \in$ Domain π, $y \in$ Range π. If $x \neq$ **res** C and $y \neq$ **res** C', then $x \in$ Domain π^*, $y \in$ Range π^* and $\pi^* x = y$.*

Proof. With respect to (5.6) the possibilities $\left.\begin{matrix}0\\1\end{matrix}\right)$, $\left.\begin{matrix}1\\0\end{matrix}\right)$ and $\left.\begin{matrix}1\\1\end{matrix}\right)$a) should be taken in acount, and therefore it suffices to show that $x \neq \pi^{-1}$ **res** C' and $y \neq \pi$ **res** C. If it were $x = \pi^{-1}$ **res** C', it would be $y = \pi x =$ **res** C', which contradicts the assumption, and similarly in the second case. Thus $x \in$ Domain π^* and $y \in$ Range π^*. Now by (5.5) follows $\pi^* x = \pi x = y$.

Let $A = (K^{(1)}, \ldots, K^{(N)})$, $A' = (K'^{(1)}, \ldots, K'^{(N')})$ be two algorithms, where $K^{(i)} = \langle b^{(i)}, C^{(i)}\rangle$ and $K'^{(i)} = \langle b'^{(i)}, C'^{(i)}\rangle$, and let π be a similarity. We write $A\,\pi\,A'$, and say that *A is similar to A' in the similarity π*, if the following requirements are satisfied:

(5.7) (i) Domain $\pi|_{Var} \supset Inp_A$, Range $\pi|_{Inp_A} = Inp_{A'}$;

(ii) Domain $\pi|_{Lab} = \{b^{(1)}, \ldots, b^{(N)}\}$ *and* $\pi b^{(i)} = b'^{(i)}$ *for each* $i = 1, 2, \ldots, N$;

(iii) *if we define* $\pi^{(1)} =_{\mathrm{df}} \pi$ *then* $C^{(1)}\,\pi^{(1)}\,C'^{(1)}$ *holds, and if* $\pi^{(p)}$ *has been already defined and* $C^{(p)}\,\pi^{(p)}\,C'^{(p)}$ *holds, where* $1 \leqq p \leqq N$, *then*

either $C^{(p)}$ is an operational command and $1 \leqq p < N$, and then we define $\pi^{(p+1)} =_{\mathrm{df}} (\pi^{(p)})^*$ and it must hold $C^{(p+1)}\,\pi^{(p+1)}\,C'^{(p+1)}$,

*or $C^{(p)}$ is a decisional command and then we define $\pi^{(q)} =_{\mathrm{df}} \pi^{(p)}$ for each q such that $b^{(q)} =$ **lab**$_j\,C^{(p)}$ for an integer $j \geqq 1$, and it must hold $C^{(q)}\,\pi^{(q)}\,C'^{(q)}$.*

(iv) $\mathbf{res}\ C^{(p)} = \mathbf{res}\ C^{(r)} \Leftrightarrow \pi^{(p)}[\mathbf{res}\ C^{(p)}] = \pi^{(r)}[\mathbf{res}\ C^{(r)}]$ *for all similarities* $\pi^{(p)}, \pi^{(r)}$ *which are defined in* (5.7iii), *where* $\mathbf{res}\ C^{(p)}$ *and* $\mathbf{res}\ C^{(r)}$ *are output occurences, and* $1 \leqq p, r \leqq N$.

The similarity π of $A\ \pi\ A'$ is called *minimal* if the requirement (i) is replaced by the following stronger requirement

(5.7) (i*) Domain $\pi|_{Var} = Inp_A$, Range $\pi|_{Var} = Inp_{A'}$.

It should be mentioned that if a command $K^{(p)}$ of A is (syntactically) superflous, then there is no similarity $\pi^{(p)}$ defined for it in (5.7iii). On the other hand if $K^{(p)}$ is not superflous then there is at least one similarity $\pi^{(p)}$ defined for $K^{(p)}$. If s_p is the number of all similarities $\pi^{(p)}$ then $s_p \leqq r(b^{(p)}) + 1$, where $r(b^{(p)})$ is the number of different labelled commands $K^{(q)}$ such that $b^{(p)} = \mathbf{lab}_j\ C^{(q)}$ for an integer $j \geqq 1$.

Theorem 5.3. $A\ \mathbf{sim}\ A' \Rightarrow$ there exists a similarity π such that $A\ \pi\ A'$.

Proof. Let $A = (K^{(1)}, \ldots, K^{(N)})$ and $A' = (K'^{(1)}, \ldots, K'^{(N')})$, where $K^{(i)} = \langle b^{(i)}, C^{(i)} \rangle$ and $K'^{(i)} = \langle b'^{(i)}, C'^{(i)} \rangle$ and let us assume $A\ \mathbf{sim}\ A'$, i.e. $N = N'$ and there are mappings ψ and φ such that (4.7) is satisfied.

First of all let us define π as follows: $\pi|_{Lab} = \psi$ and therefore π satisfies (5.7ii); if $x^k \in Oc_A$ and $y^h \in Oc_{A'}$ are arbitrary input occurrences such that $\varphi(x^k) = y^h$, then we may define $\pi x = y$, because φ satisfies (**) of (4.7iv). With respect to the definition of input occurrences in A or A' it follows that Domain $\pi|_{Var} = Inp_A$ and Range $\pi|_{Var} = Inp_{A'}$. Thus π satisfies (5.7i).

It is clear that the requirement (5.7iv) follows immediately from (4.7****), and therefore it remains to prove (5.7iii), i.e. to prove the corresponding assertion concerning all possible similarities $\pi^{(p)}$ for each $C^{(p)}$, $1 \leqq p \leqq N$. If $lb = (K_1, \ldots, K_q) \in LB_A$, where $K_i = \langle b_i, C_i \rangle$, is an arbitrary labelled branch containing $C^{(p)}$, and if $lb' = (K'_1, \ldots, K'_q) \in LB_{A'}$, where $K'_i = \langle b'_i, C'_i \rangle$, is the corresponding labelled branch, which is determined uniquely by ψ, i.e. $b'^{(i)} = \psi(b^{(i)})$ for $i = 1, 2, \ldots, q$, then we may define mappings $\pi_1, \pi_2, \ldots, \pi_q$ as follows: $\pi_1 = \pi$ and if $1 \leqq t < q$ and π_t has been defined then either C_t is operational and we put $\pi_{t+1} = \pi_t^*$, or C_t is decisional and we put $\pi_{t+1} = \pi_t$. It is clear that each possible $\pi^{(p)} = \pi_t$, when $C^{(p)} = C_t$ and lb was chosen suitably. Therefore it is sufficient to prove the assertion for π_t (instead of $\pi^{(p)}$ from (5.7iii)), which says: $C_t \pi_t C'_t$. With respect to (4.7ii) and (4.7iii) it remains to prove only (5.4ii), i.e. $\mathbf{arg}_i\ C'_t = \pi_t\ \mathbf{arg}_i\ C_t$ for $i = 1, 2, \ldots$, and we shall prove it by induction as follows.

1) According to (3.2ii) $C_1 = C^{(1)}$ and $C'_1 = C'^{(1)}$ and therefore let us prove $\pi\ \mathbf{arg}_i\ C^{(1)} = \mathbf{arg}_i\ C'^{(1)}$ for each $i = 1, 2, \ldots$ If we denote $\mathbf{arg}_i^\sharp\ C^{(1)} = x^k$ and $\mathbf{arg}_i^\sharp\ C'^{(1)} = y^h$, where $1 \leqq i$, then x^k, y^h must be input occurrences and $\varphi_1(x^k) = y^h$. Therefore by the definition of π it holds $\pi x = y$.

2) Let us accept the following inductive assumption: $\pi_t\, \mathbf{arg}_i\, C^{(t)} = \mathbf{arg}_i\, C'^{(t)}$ for each $t = 1, 2, \ldots, p$ and for each $i = 1, 2, \ldots$, where $1 \leqq p < q$, and let us prove: $\pi_{p+1}\, \mathbf{arg}_i\, C^{(p+1)} = \mathbf{arg}_i\, C'^{(p+1)}$ for each $i = 1, 2, \ldots$

Let us denote ${}^k x = \mathbf{arg}_i^{\sharp}\, C^{(p+1)}$ and ${}^h y = \mathbf{arg}_i^{\sharp}\, C'^{(p+1)}$, where $1 \leqq i$. Using Lemma 4.3 let ${}^r x$, ${}^s y$ be the unique defining occurrences in lb, lb', respectively such that ${}^k x \in SC_{lb}({}^r x)$ and ${}^h y \in SC_{lb'}({}^s y)$.

a) If $r = 0$ then also $s = 0$, which follows by contradiction from (***) of (4.7iv). In this case $k = h$ and there is no defining occurrence of x, y, in C_t, C'_t, respectively, for $t = 1, 2, \ldots, p$. Now ${}^1 x$ and ${}^1 y$ are input occurrences and therefore $\pi_1 x = y$. By the definition of π_t and by repeated application of Lemma 5.2 we obtain $\pi_{p+1} x = y$.

b) If $r > 0$ then also $s > 0$, which follows by (***) again. In this case there exists t, $1 \leqq t \leqq p$, such that ${}^r x = \mathbf{res}^{\sharp}\, C_t$ and ${}^s y = \mathbf{res}^{\sharp}\, C'_t$. Using (5.5) we prove that $\pi_{t+1} x = y$. If $t = p$, we proved $\pi_{p+1} x = y$, and if $t < p$ then there is no defining occurrence of x, y in C_v, C'_v, respectively, for each v such that, $t + 1 \leqq v \leqq p$. Thus again the assumption for repeated application of Lemma 5.2 and of definition of π_v are satisfied, and we obtain $\pi_{p+1} x = y$ from $\pi_{t+1} x = y$, which finishes the proof.

Lemma 5.4. *Let σ, σ' be states, C, C' be operational (or restoring) commands, and π be a similarity such that $\sigma\, \pi\, \sigma'$ and $C\, \pi\, C'$ hold. If the next state $C\sigma$ exists then also $C'\sigma'$ exists and $(C\sigma)\, \pi^*\, (C'\sigma')$ holds, where π^* is the star similarity, defined by (5.5) and (5.6).*

Proof. By the assumption $C\, \pi\, C'$, i.e. by (5.4) it follows $\mathbf{symb}\, C = \mathbf{symb}\, C'$ and let us put $\mathbf{symb}\, C = f^{(n)}$. We want to prove $(C\sigma)(x) = (C'\sigma')(\pi^* x)$ for each $x \in$ $\in$ Domain π^*, and therefore, with respect to (5.6) let us distinguish the five possibilities:

$\left.\begin{matrix} 0 \\ 0 \end{matrix}\right)$ In this case Domain π^* = Domain $\pi \cup \{\mathbf{res}\, C\}$, Range π^* = Range $\pi \cup$ $\cup \{\mathbf{res}\, C'\}$, and further $\mathbf{res}\, C \notin$ Domain π, $\mathbf{res}\, C' \notin$ Range π. If $x \in$ Domain π^* and $x \neq \mathbf{res}\, C$, then $x \in$ Domain π and $\pi x \neq \mathbf{res}\, C'$. Therefore by (3.4) $(C\sigma)(x) = \sigma(x)$ and $(C'\sigma')(\pi x) = \sigma'(\pi x)$; by the assumption $\sigma\, \pi\, \sigma'$ it follows $\sigma(x) = \sigma'(\pi x)$, and by (5.5) $\pi^* x = \pi x$. Thus we have $(C\sigma)(x) = \sigma(x) = \sigma'(\pi x) = (C'\sigma')(\pi x) = (C'\sigma')(\pi^* x)$.

If $x \in$ Domain π^* and $x = \mathbf{res}\, C$ then by (4.5) $\pi^*(\mathbf{res}\, C) = \mathbf{res}\, C'$. By (3.4) $(C\sigma)(\mathbf{res}\, C) = f^{(n)}(\sigma(\mathbf{arg}_i\, C), \ldots, \sigma(\mathbf{arg}_n\, C))$ and $(C'\sigma')(\pi^*\, \mathbf{res}\, C) = f^{(n)}(\sigma'(\mathbf{arg}_1\, C'), \ldots, \sigma'(\mathbf{arg}_n\, C'))$; by the assumption $C\, \pi\, C'$, i.e. by (5.4), it follows $\mathbf{arg}_i\, C' = \pi(\mathbf{arg}_i\, C)$ for each $i = 1, 2, \ldots, n$, and by the assumption $\sigma\, \pi\, \sigma'$ it follows $\sigma'(\pi(\mathbf{arg}_i\, C)) = \sigma(\mathbf{arg}_i\, C)) = \sigma(\mathbf{arg}_i\, C)$ for $i = 1, 2, \ldots, n$. Thus $(C\sigma)(\mathbf{res}\, C) = f^{(n)}(\sigma(\mathbf{arg}_1\, C), \ldots, \sigma(\mathbf{arg}_n\, C)) = f^{(n)}(\sigma'[\pi(\mathbf{arg}_1\, C)], \ldots, \sigma'^2\pi(\mathbf{arg}_n\, C]) = f^{(n)}(\sigma'(\mathbf{arg}_1\, C'), \ldots, \sigma'(\mathbf{arg}_n\, C')) = (C'\sigma')\, \pi^*\, \mathbf{res}\, C)$.

Therefore $(C\sigma)\, \pi^*\, (C'\sigma')$ is true.

Let us briefly prove the next possibility $\binom{0}{1}$, where Domain $\pi^* = (\text{Domain } \pi -$ $- \{\pi^{-1}\ \textbf{res}\ C'\}) \cup \{\textbf{res}\ C\})$, Range $\pi^* = \text{Range } \pi$, and $\textbf{res}\ C \notin \text{Domain } \pi$, $\textbf{res}\ C' \in$ $\in \text{Range } \pi$. If $x \in \text{Domain } \pi^*$ and $x \neq \textbf{res}\ C$ then $\pi x \neq \textbf{res}\ C'$. Similarly as above using (3.4), (5.5) and the assumption $\sigma\ \pi\ \sigma'$ we get $(C\sigma)(x) = \sigma(x) = \sigma'(\pi x) =$ $= (C'\sigma')(\pi x) = (C'\sigma')(\pi^* x)$.

If $x \in \text{Domain } \pi^*$ and $x = \textbf{res}\ C$, then by (5.5) $\pi^*(\textbf{res}\ C) = \textbf{res}\ C'$. Further by (3.4) and by the assumption $C\ \pi\ C'$ and $\sigma\ \pi\ \sigma'$ we get, similarly as above, $(C\sigma)(\textbf{res}\ C) = f^{(n)}(\sigma(\textbf{arg}_1\ C), \ldots, \sigma(\textbf{arg}_n\ C)) = f^{(n)}(\sigma'[\pi(\textbf{arg}_1\ C)], \ldots, \sigma'[\pi(\textbf{arg}_n\ C)]) =$ $= f^{(n)}(\sigma'(\textbf{arg}_1\ C'), \ldots, \sigma'(\textbf{arg}_n\ C')) = (C'\sigma')(\pi^*\ \textbf{res}\ C)$.

The remaining three possibilities can be proved in a analogous way.

Theorem 5.5. *If* $A\ \pi\ A'$ *then* $\boldsymbol{f}_{A,AA}\langle Inp_A, Outp_A\rangle = \boldsymbol{f}_{A',AA}\langle Inp_{A'}, Outp_{A'}\rangle$.

Proof. Let us assume that $A = (K^{(1)}, \ldots, K^{(N)})$, where $K^{(p)} = \langle b^{(p)}, C^{(p)}\rangle$, and $A' = (K'^{(1)}, \ldots, K'^{(N)})$, where $K'^{(p)} = \langle b'^{(p)}, C'^{(p)}\rangle$. Further let π be the minimal similarity, and let σ_0, σ'_0 be two arbitrary initial states such that $\sigma_0\ \pi\ \sigma'_0$.

If we put $\pi_1 =_{\text{df}} \pi$ and $C_1 = C^{(1)}$, $C'_1 = C'^{(1)}$ then $\sigma_0\ \pi_1\ \sigma'_0$, and by (5.7iii) also $C_1\ \pi_1\ C'_1$ holds, which will be the starting assertion of an induction. The inductive assumption is as follows: $\sigma_{p-1}\ \pi_p\ \sigma'_{p-1}$ and $C_p\ \pi_p\ C'_p$ hold, where $LB_{A,AA}(A, \sigma_0) =$ $= (K_1, \ldots, K_q)$, $K_p = \langle b_p, C_p\rangle$, and $Cpt_{A,AA}(A', \sigma_0) = (\sigma_0, \sigma_1, \ldots, \sigma_{q-1})$ and $LB_{A',AA}(A', \sigma'_0) = (K'_1, \ldots, K'_{q'})$, $K'_p = \langle l'_p, C'_p\rangle$, and $Cpt_{A',AA}(A', \sigma'_0) = (\sigma'_0, \ldots$ $Q \ldots, \sigma'_{q'-1})$. Let us denote $lb' = (K_1, \ldots, K_q)$ and $lb' = (K'_1, \ldots, K'_{q'})$. We want prove $\sigma_p\ \pi_{p+1}\ \sigma'_p$ if $1 \leqq p < q$. If C_p is an operational (or restoring) command then it follows by Lemma 5.4, because $\sigma_p = C_p\sigma_{p-1}$ and $\sigma'_p = C'_p\sigma'_{p-1}$, and if C_p is a decisional command then by (3.4ii) $\sigma_p = \sigma_{p-1}$ and $\sigma'_p = \sigma'_{p-1}$, and by (5.7iii) $\pi_{p+1} = \pi_p$, which proves $\sigma_p\ \pi_{p+1}\ \sigma'_p$ also. By this induction is proved $\sigma_{q-1}\ \pi_q\ \sigma'_{q-1}$.

We need to determine two one-to-one mappings π_{inp} and π_{outp}, which are required in (3.7). Obviously we define $\pi_{inp} = \pi$ and, with respect to (5.7iv) it is possible to define $\pi_{outp}[\textbf{res}\ C^{(p)}] = \pi^{(p)}[\textbf{res}\ C^{(p)}]$ for each output occurrence $\textbf{res}\ C^{(p)}$ and each p, $1 \leqq p \leqq N$. It is clear that $\pi_{outp}|_{Outp_{lb}} = \pi_q$, i.e. π_{outp} is an extension of π_q for each $lb \in LB_A$; thus according to (3.6) if $y \in Outp_{lb}$ and $Inp_A = \{x_1, \ldots, x_r\}$ then

$$\boldsymbol{f}_{A,AA,y}(\sigma_0(x_1), \ldots, \sigma_0(x_r)) = \sigma_{q-1}(y) = \sigma'_{q-1}(\pi_q y) = \sigma'_{q-1}(\pi_{outp} y) =$$
$$= \boldsymbol{f}_{A',AA,\pi_{outp}y}(\sigma'_0(\pi_{inp} x_1), \ldots, \sigma'_0(\pi_{inp} x_r)).$$

This proves (3.8) if σ_0 and σ'_0 vary for all possible cases.

Proof of Lemma 3.2.

Let us take $A' = A$ and let π be the identical similarity such that Domain $\pi =$ $= Inp_A$. Then $A\ \pi\ A'$ and the assumption $\sigma_0(x) = \sigma'_0(x)$ for each $x \in Inp_A$ from Lemma 3.2 may be rewritten as $\sigma_0\ \pi\ \sigma'_0$. Now using the induction from the proof of

Theorem 5 we shall recognize that all π_p are identical mappings. Therefore $\sigma_{q-1}\,\pi_q\,\sigma'_{q-1}$ implies $\sigma_{q-1}(x) = \sigma'_{q-1}(\pi_q x) = \sigma'_{q-1}(x)$ for each $x \in Outp_{lb}$ which is the required assertion c).

6. SIMPLE ALGORITHMS AND THEIR COMPLEXITY

An algorithm $A \in Alg_{Voc}$ is called *simple* if all its commands are operational (the stopping command may be omitted because there exist just one branch of A). In [7] by the simple algorithm A its *algorithmic net* (without cycles) N_A is determined such that N_A is an oriented acyclic multigraph with labelled vertices and edges. The algorithmic net is a generalization of the concept of term and of usual binary rooted trees, which correspond to arithmetical expressions in two respects: more than one root is admited, and the output degrees of inner vertices may be arbitrary (and not necessary always 2).

By an algorithmic net its partially ordered set is determined, the linear extensions of which represent all possible *courses* of evaluation of the net. These courses are simple algorithms which are functionally equivalent in all interpretations. On the other hand the proper concept of homomorphism of algorithmic nets allows to get a complete characterization of all algorithmic nets, the courses of which are functionally equivalent.

The *with of algorithmic net* is determined in [8] and [9], where in several special cases more general results are obtained than in [13], [15] and [16], but the general problem remains still open.

References

[1] *Blikle, A.:* Automata and Grammars (Polish), Państwowe Wydawnictwo Naukove, Warsaw 1971.

[2] *Čulík, K.:* 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] *Čulík, K.* and *Arbib, M. A.:* Sequential and jumping machines and their relation to computers (in print in Acta Informatica).

[4] *Čulík, K.:* Theory of algorithms and programming languages (Czech), Textbook for the Czech Institute of Technology (not published), Prague 1970.

[5] *Čulík, K.:* Classifications of programming theories and language (in print in Information Processing Machines).

[6] *Čulík, K.:* Algorithmization of algebras and relational structures, Comentationes Mathematicae Universitatis Carolinae *13*, 3 (1972), 457—477.

[7] *Čulík, K.:* Combinatorial problems in theory of complexity of algorithmic nets without cycles for simple computers, Apl. Math. *16* (1971), 188—202.

[8] *Čulík, K.:* A note on complexity of algorithmic nets without cycles, Apl. Math. *16* (1971), 297—301.

[9] *Čulík, K.:* Optimalization of special programs with respect to the economy of storage (Czech), Proceedings of Conference of the Czech Institute of Technology, Prague 1971, 1—10.
[10] *Engeler, E.:* Algorithmic Approximations, Jour. Comp. and Syste. Sciences *5* (1971), 67—82.
[11] *Luckham, D. C., D. M. R. Park, M. S. Paterson:* On formalized computer programs, Jour. Comp. and Syste. Sciences *4* (1970), 220—249.
[12] *Milner, R.:* Equivalences on Program Schemes, Jour. Comp. and Syste. Sciences *4* (1970), 205—219.
[13] *Nagata, I.:* On compiling for arithmetic expressions, Comm. ACM *10* (1967), 492—494.
[14] *Pawlak, Z.:* On the notion of a computer, Logic. Math. and Phil. Sci. *3* (1968), 255—267.
[15] *Redziejowski, R. R.:* On arithmetic expressions and trees, Comm. ACM *12* (1969), 81—84.
[16] *Sethi, R.* and *J. D. Ullman:* The generation of optimal code for arithmetic expressions, Jour. ACM *17* (1970), 715—728.
[17] *Wijngaarden, A. van* (editor): ALGOL 68, Math. Centrum, Amsterdam 1968.

Author's address: 160 00 Praha 6-Vokovice, Lužná ul., ČSSR (Výzkumný ústav matematických strojů).