Commentationes Mathematicae Universitatis Carolinae

Karel Čulík

Algorithmization of algebras and relational structures

Commentationes Mathematicae Universitatis Carolinae, Vol. 13 (1972), No. 3, 457--477

Persistent URL: http://dml.cz/dmlcz/105434

Commentationes Mathematicae Universitatis Carolinae
13,3 (1972)

ALGORITHMIZATION OF ALGEBRAS AND RELATIONAL STRUCTURES

Karel ČULÍK, Praha

The simple and natural concept of algorithm over a relational structure is introduced which is an essential generalization of the traditional concept of term and is related to the concept of program in programming languages for computers. The individual operations of the relational structure correspond to elementary algorithms and the individual relations of the relational structure allow the branching of the algorithm. Branch equivalence of algorithms is introduced.

1. Relational structures with relations in k-valued logics

A relational structure is determined 1) by its set of objects $Obj \neq \emptyset$, 2) by its set of operation Opr, when an n-ary operation $f^{(n)} \in Opr$, where $n \geq 1$, is a function such that $\emptyset \neq Domain\ f^{(n)} \subset Obj^n$ and $Range\ f^{(n)} \subset Obj$, and finally 3) by its set of relations Rel, when an n-ary relation $g^{(n)} \in Rel$, where $n \geq 1$, is characterized extensionally by the requirement

---

AMS, Primary 02E10, 68A05 Ref. Ž. 2.65, 8.741
Secondary 68A20, 08A05

$Field\ g^{(n)} \subset Obj^n$. The equality relation "$=_A$" belongs to Rel always.

It is tacitly assumed that each $g^{(n)} \in Rel$ is an n-ary relation in the usual 2-valued logic, which means that $g^{(n)}$ may be considered as an n-ary function such that Domain $g^{(n)} = Obj^n$ and Range $g^{(n)} \subset$ $\subset \{true, false\}$, where true and false are the two truth values of the 2-valued logic, which should be different from all objects of the relational structure.
Therefore a complete characterization of the relational structure should be as follows: $\langle Obj, \{true, false\}, Opr, Rel \rangle$, where in Rel are functions of certain sort also; thus we call such a structure an algorithmic algebra.

If we assume for a moment that for an arbitrary $k \geq$ ≥ 2, that $(\underline{1}, \underline{2}, \ldots, \underline{k})$ is a fixed denotation and ordering of all k truth values of k-valued logic (the truth value should be distinguishable from the objects), then an n-ary relation in k-valued logic $g^{(n)}_{(k)}$ may be considered as a function such that Domain $g^{(n)}_{(k)} = Obj^n$ and Range $g^{(n)}_{(k)} \subset \{\underline{1}, \underline{2}, \ldots, \underline{k}\}$, which is characterized extensionally by the sequence $(g^{(n)}_1, g^{(n)}_2, \ldots, g^{(n)}_k)$ of length k such that:

(1.1) $g^{(n)}_i \subset Obj^n$, $g^{(n)}_i \cap g^{(n)}_j = \emptyset$ for all $i, j = 1, 2, \ldots, k$, where $i \neq j$, and $\bigcup_{i=1}^{k} g^{(n)}_i = Obj^n$;

(1.2) $g^{(n)}_{(k)}(o_1, \ldots, o_n) = \underline{i} \Longleftrightarrow (o_1, \ldots, o_n) \in g^{(n)}_i$,

where $1 \leq i \leq \aleph$, for all $o_j \in Obj$ and each $j = 1, 2, \ldots, m$.

For example, if we identify $1 \equiv$ true, $2 \equiv$ false and $g^{(m)}_{(2)} \equiv g^{(m)}$, then according to (1.1) th[illegible]tion $g^{(m)}$ (in 2-valued logic) is charac[illegible]ne following sequence: $(Field\ g^{(m)}, Obj^m - Field\ g^{(m)})$.

Everywhere further in an algebra it will be admitted that there are particular relations in $\aleph$-valued logic for many different values of $\aleph$. Moreover it will be admitted that the operations are partial, and in fact, also partial relations may be admitted.

2. Enrichment of the language of terms

Ignoring the quantifiers, an axiom of an algebra is usually the following string of symbols: $T_1 =_A T_2$ where T_1 and T_2 are terms and "$=_A$" is the symbol of the equality relation from Rel. The terms are defined, using a set Var of symbols called (individual) variables, as follows: 1) each variable is a term; 2) the string $f^{(m)}(x_1, \ldots, x_m)$ is a term (called elementary) if $x_i \in Var$ for $i = 1, 2, \ldots, m$, and "$f^{(m)}$" is the symbol of an operation from Opr; 3) the string $f^{(m)}(T_1, \ldots, T_m)$ is a term if T_i is a term for $i = 1, 2, \ldots, m$, and "$f^{(m)}$" is the symbol of an operation from Opr (usually for $m = 2$ the string $(x_1 f^{(2)} x_2)$ is used instead of the string $f^{(2)}(x_1, x_2)$). Obviously the set Opr must be

distinguished from the set $Symb\,Opr$ of symbols of operations from Opr .

Each term determines an algorithm, i.e. a complete prescription for the consecutive application of operations from Opr , in a familiar way, which will be recalled by the following example in numerical algebra. The term $T = ((x-y)_t \swarrow (y+z)_v)_w$, all the right-hand brackets of which are labelled by mutually different variables not occurring in it, is transformed into the algorithm

$$(2.1) \qquad A_T = (\underbrace{x-y=:t}_{C_1}\,;\ \underbrace{y+z=:v}_{C_2};\ \underbrace{t/v=:w}_{C_3})\ ,$$

where three applications of operations C_1 , C_2 and C_3 (separated by semicolons) are distinguished and called operational rules or commands, and " $=:$ " is a new symbol (called assignation in programming languages). The commands are executed consecutively from the left to the right.

The above mentioned transformation of T into A_T is unique if the following requirement is accepted; always the left most possible occurrence of operation must be applied. Without this requirement one may get an other algorith

$$(2.1^*) \qquad A^*_T = (y+z=:v;\ x-y=:t\ ;\ t/v=:w)\ .$$

An algorithm of operational commands (as a prescription) is used as follows. An operational command C is, in general, the following string

(2.2) $$C = (f^{(n)}(x_1, \dots, x_n) =: x_0) ,$$

where on the left, right hand side of the assignation is an elementary term, a variable, respectively. If some values (i.e. objects) of the variables x_i, $1 \leq i \leq n$, are prescribed, then it means that a function $\sigma \in Obj^{Var}$, called (initial) state, is prescribed, such that $\sigma(x_i)$ is the corresponding value=object. The operation $f^{(n)}$ should be applied to the prescribed values of $x_1, x_2, \dots, x_n$ and the resulting function value $f^{(n)}(\sigma(x_1), \sigma(x_2), \dots, \sigma(x_n))$ should be assigned to the variable x_0, which means that a new (resulting) state $\sigma^* = C\sigma$ is determined as follows:

(2.3)
$$\sigma^*(x_0) = f^{(n)}(\sigma(x_1), \sigma(x_2), \dots, \sigma(x_n)) ,$$
$$\sigma^*(t) = \sigma(t) \text{ for each } t \in Var \text{ such that } t \neq x_0 .$$

E.g. in the example (2.1) let us start with the input state $\sigma_0 \in Obj^{Var}$ such that $\sigma_0(x) = 10$, $\sigma_0(y) = 1$ and $\sigma_0(z) = 2$ (the remaining variables do not matter), i.e. the arithmetic expression $((10 - 1) / (1 + 2))$ should be evaluated. Then according to (2.3) one will get consecutively: $\sigma_i = C_i \sigma_{i-1}$ for $i = 1, 2, 3$, and $\sigma_1(a) = \sigma_0(x) - \sigma_0(y) = 10 - 1 = 9$, $\sigma_2(b) = \sigma_1(y) + \sigma_1(z) = \sigma_0(y) + \sigma_0(z) = 1 + 2 = 3$, and $\sigma_3(c) = \sigma_2(a) / \sigma_2(b) = \sigma_1(a) / \sigma_2(b) = 9 / 3 = 3$.

We stopped with the output state σ_3 .

Two terms T_1 and T_2 are equivalent if there is a one-to-one mapping φ of the set of variables of T_1 onto the set of variables of T_2 such that after replacement each occurrence of a variable x in T_1 by $\varphi(x)$ the term T_2 arises from the term T_1 .

The variables x, y, z (which occur in T) are called input variables of the algorithm A_T , and the variable c (which does not occur in T) is called the output variable of A_T . The algorithm A_T is abbreviated by a single generalized operational command

$$A_T(x, y, z) =: c \quad , \tag{2.4}$$

where on the left, right hand side of the assignation occur all the input, output variables of A_T, respectively (their ordering is unessential), and A_T may denote a composed operation $F^{(3)}$, i.e. Domain $F^{(3)} \subset Obj^3$ and Range $F^{(3)} \subset Obj$, usually denoted by the original term T , which is determined and evaluated by the algorithm A_T with respect to all possible input states and the corresponding output states.

The enrichment of the language of terms, which was necessary in order to be able to express the algorithms of operational commands, is rather simple. On the other hand the language of terms itself is unsufficient, because a lot of composed operations used in numerical algebras cannot be expressed in it. The simplest example of such composed operation is the absolute value $|x|$, defined usually as follows:

(2.5) $$|x| = \begin{cases} = x & \text{for } x \geq 0 \ , \\ = -x & \text{for } x < 0 \ , \end{cases}$$ i.e. for x which does not satisfy $x \geq 0$, where " $-$ ", " $\geq$ " and " $<$ " are well known symbols of operations and relations in numerical algebras. It is clear that the relations must be taken in account and the branching of algorithms must be allowed and uniquely determined.

3. Conditions

First of all all the truth values required in Sect. 1 are superfluous and may be forgotten, because they may be replaced by the variables as follows. If $g^{(m)}_{(k)} \in Rel$ and $[a_1, a_2, \ldots, a_k]$ is a k-tuple of variables, then let $g^{(m)}_{[a_1, a_2, \ldots, a_k]}$ be a function, called k-valued condition (derived from the relation in k-valued logic $g^{(m)}_{(k)}$ by Var), and defined in accordance with (1.2) as follows:

(3.1) $$g^{(m)}_{[a_1, a_2, \ldots, a_k]}(o_1, \ldots, o_m) = a_i \ , \text{ where } 1 \leq i \leq k,$$

if $i = g^{(m)}_{(k)}(o_1, \ldots, o_m)$, where $o_j \in Obj$ for each $j = 1, 2, \ldots, m$.

Therefore Domain $g^{(m)}_{[a_1, a_2, \ldots, a_k]} = Obj^m$ and Range $g^{(m)}_{[a_1, a_2, \ldots, a_k]} \subset \{a_1, a_2, \ldots, a_k\}$

Let $Rel_{[Var]}$ be the set of all conditions derived from Rel by Var. Now a complete characterization of an

arbitrary algorithmic algebra is as follows:
$\langle Obj, Var, Opr, Rel_{[Var]} \rangle$, where the truth values are omitted, although all relations in k-valued logic are admitted for each $k = 2, 3, \ldots$.

A k-valued condition $q^{(n)}_{[a_1, \ldots, a_k]}$, where $a_i \neq a_j$ for $i \neq j$ and $i, j = 1, 2, \ldots, k$, allows the decision making among k possibilities, and therefore this is a suitable tool for the determination of branching algorithms (it is assumed $k \geq 2$).

On the other hand if $a_i = a_j$ for $i, j = 1, 2, \ldots, k$, there is no decision making, because only one possibility is admitted. Such a degenerated condition determines no branching (and it could be replaced by its single variable a_i which is its value independently on the state).

The string

$$C = q^{(n)}_{[a_1, \ldots, a_k]}(x_1, \ldots, x_n) \quad , \tag{3.2}$$

where $q^{(n)}_{[a_1, \ldots, a_k]} \in Rel_{[Var]}$ and $x_i \in Var$ for $i = 1, 2, \ldots, n$, is called decision command (it corresponds to the elementary term over the operations and if $n = k = 2$ then it is replaced by the following string "$x_1 q^{(2)} x_{2[a_1, a_2]}$"). If $\sigma \in Obj^{Var}$ is a (current) state, then the execution of the decision command (3.2) means only the determination of the resulting variable

$$q^{(n)}_{[a_1, \ldots, a_k]}(\sigma(x_1), \ldots, \sigma(x_n)) \in \{a_1, a_2, \ldots, a_k\} \quad ,$$

and does not cause any change of the state σ .

E.g. " $\geq_{[a_1, a_2]}(x, y)$ " or " $x \geq y_{[a_1, a_2]}$ " is a decision command and if $\sigma(x) = 1$, $\sigma(y) = 2$, then the resulting variable is a_2 , etc.

It is convenient to have the stopping command STOP, the execution of which means that no further command may be executed.

In order to make the further considerations easier let us distinguish two sorts of variables: the labels from $\mathcal{Lab}$ and the proper variables from $P\mathcal{Var}$, where $\mathcal{Lab} \cup P\mathcal{Var} = \mathcal{Var}$ and $\mathcal{Lab} \cap \mathcal{Var} = \emptyset$. Then in the decision command (3.2) always $a_i \in \mathcal{Lab}$ for $i = 1, 2, \ldots, k$ and $x_j \in P\mathcal{Var}$ for $j = 1, 2, \ldots, m$. Thus the algorithmic algebra is characterized by $\langle \mathcal{Obj}, P\mathcal{Var}, \mathcal{Opr}, \mathcal{Rel}_{[\mathcal{Lab}]} \rangle$. Let $\mathcal{Com}$ be the set of all commands over this algebra, where we add (for rather formal reasons) the following strings

(3.3) $\qquad x =: y$, where $x, y \in P\mathcal{Var}$,

which are called restoring commands. Obviously the restoring commands correspond to the identity operation in $\mathcal{Obj}$.

Further let us add the following strings

(3.4) $\qquad o =: y$, where $o \in \mathcal{Obj}$, $y \in P\mathcal{Var}$,

which are called input commands and which correspond to the constant operations. It is assumed that $\mathcal{Obj}$ is the set of symbols, which may be written here and which do not denote anything further but only themselves (it is superfluous to distinguish between the number 2 and the numeral 2 ; here only the numerals are concerned).

It is convenient to include the restoring and the input commands into the set of operational commands, and it is clear what change of state is caused by them.

The particular commands may be considered as the elementary algorithms and the main question is how to compose them in order to get all possible algorithms over the algebra under the consideration.

The occurrence of a proper variable on the right hand side of the assignation in an operational, restoring or input command is called the defining occurrence and all other occurrences of proper variables in all sorts of commands are called applied occurrences of proper variables.

4. Algorithms over an algebra

A finite (totally) ordered set $A = (K^{(1)}, \dots, K^{(N)})$ of pairs $K^{(i)} = \langle \mathfrak{l}^{(i)}, C^{(i)} \rangle$, where $\mathfrak{l}^{(i)} \in \mathfrak{Lab}$, $C^{(i)} \in \mathfrak{Com}$ and

(4.1) $\mathfrak{l}^{(i)} \neq \mathfrak{l}^{(j)}$ for $i \neq j$ and $i, j = 1, 2, \dots, N$,

is called algorithm (or program) over the algorithmic algebra $\langle \mathfrak{Obj}, \mathrm{PVar}, \mathfrak{Opr}, \mathfrak{Rel}_{[\mathfrak{Lab}]} \rangle$ if there exists at least one branch of A , which is defined as follows.

A finite sequence $B = (K_1, K_2, \dots, K_p)$ is called a branch of the set A , if the following requirements are satisfied:

(4.2) (i) $K_i = K^{(j_i)}$ for each $i = 1, 2, \dots, p$, where $1 \leq j_i \leq N$;

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

(iii) $K_p = K^{(i)}$ where $C^{(i)}$ = STOP and $1 \leq i \leq N$;

(iv) if $K_i = K^{(j)}$ where $1 \leq i < p$, then $C^{(j)} \neq$ STOP, and if $C^{(j)}$ is operational, then $1 \leq j \leq N$ and $K_{i+1} = K^{(j+1)}$, but if $C^{(j)} = g^{(m)}_{[a_1,\dots,a_k]}(x_1,\dots,x_m)$ is a decision command then there exist integers r, $1 \leq r \leq k$, and s, $1 \leq s \leq N$ such that $l^{(s)} = a_r$ and $K_{i+1} = K^{(s)}$;

(v) there exists i, $1 \leq i \leq p$ such that $K_i = K^{(j)}$ where $1 \leq j \leq N$ and $C^{(j)}$ is an operational command, which is not an input command.

The finite sequence $OB = (C_{i_1}, C_{i_2}, \dots, C_{i_q})$ is called operational branch of the branch B of A if OB arised from B by omitting of

(4.3) (i) all labels $l^{(i)}$, $1 \leq i \leq N$; and

(ii) all decision and stopping commands (and all superfluous brackets and commas).

A proper variable which occurs in the branch B, i.e. in a command of B, is called input, output variable of B, if its first, last occurrence in B respectively, is the applied, the defining occurrence, respectively. Let Inp_B, $Outp_B$ be the set of all input, output variables of B, respectively. Obviously $Inp_B \neq \emptyset \neq Outp_B$. A proper variable, which occurs in an input command of B, is

called a parameter of B. Let $\mathcal{P}ar_B$ be the set of all parameters of B.

Let us define

$$(4.4)\quad \mathcal{I}nf_A = \bigcup_{B \in Br_A} \mathcal{I}nf_B,\ \mathcal{O}utp_A = \bigcup_{B \in Br_A} \mathcal{O}utp_B,\ \mathcal{P}ar_A = \bigcup_{B \in Br_A} \mathcal{P}ar_B$$

where Br_A is the set of all branches of the algorithm A, and let each $x \in \mathcal{I}nf_A$, $x \in \mathcal{O}utp_A$ be called input, output variable of the algorithm A, respectively. Finally let

$$(4.5)\quad A(\mathcal{I}nf_A) =: \mathcal{O}utp_A$$

be the generalized operational command, which serves as an abbreviation of the algorithm A when it is used within an other algorithm.

The algorithm $A = (K^{(1)}, \dots, K^{(N)})$ is applied to an arbitrary input state $\sigma_0 \in \mathcal{O}bj^{Var}$ (and we do matter only the partial state $\sigma_0 \big| \mathcal{I}nf_A$) as follows: we start with $K_1 = K^{(1)}$ and σ_0, and, in general, if $K_i = K^{(j)}$, where $i \le j \le N$, and σ_{i-1} have been determined, then the following three possibilities must be distinguished:

a) $C^{(j)} = f^{(m)}(x_1, x_2, \dots, x_m) =: x_0$; if $j < N$ and $(\sigma_{i-1}(x_1), \dots, \sigma_{i-1}(x_m)) \in Domain\ f^{(m)}$, then $\sigma_i = C^{(j)} \sigma_{i-1}$ and $K_{i+1} = K^{(j+1)}$, otherwise the algorithm is finished without any result (the restoring and input commands are included in this case);

b) $C^{(j)} = g^{(m)}_{[a_1, \dots, a_k]}(x_1, \dots, x_m)$; if

$(\sigma_{i-1}(x_1), \ldots, \sigma_{i-1}(x_m)) \in Domain\ g^{(m)}_{[a_1,\ldots,a_k]}$ and if there exists h, $1 \leq h \leq N$ such that $g^{(m)}_{[a_1,\ldots,a_k]}(\sigma_{i-1}(x_1), \ldots, \sigma_{i-1}(x_m)) = b^{(h)}$, then $\sigma_i = \sigma_{i-1}$ and $K_{i+1} = K^{(h)}$, otherwise the algorithm is finished without any result;

c) $C^{(j)}$ = STOP; then the algorithm is finished (called stopped) and σ_{i-1} is the result called the output state corresponding to the input state σ_0 in A.

A more formalized description of the execution of the algorithm requires the following generalization of state: a state is a function $\sigma \in (Obj \cup Var \cup Com)^{Var}$, and, further, that each input state σ_0 of algorithm $A = (K^{(1)}, \ldots, K^{(N)})$ must satisfy: $\sigma_0(b^{(i)}) = C^{(i)}$ for $i = 1, 2, \ldots, N$. Using this fact the next command which should be executed after a decision command $C = g^{(m)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_m)$, when σ is the current state, is denoted by the following expression:

$$\sigma(g^{(m)}_{[a_1,\ldots,a_k]}(\sigma(x_1), \ldots, \sigma(x_k))) \quad . \tag{4.6}$$

If σ_n is the output state which corresponds to the input state σ_0 in the algorithm A then the sequence $(K_1, \ldots, K_n, K_{n+1})$ defined by a), b) and c) is a branch B of A; if $Inp_A = \{x_1, \ldots, x_k\}$ and $Outp_B = \{y_1, \ldots, y_s\}$ then the object $\sigma_n(y_j)$, where $1 \leq j \leq s$, is assigned to the n-tuple of objects $(\sigma_0(x_1), \ldots, \sigma_0(x_k))$. Therefore in this way there are q, where $q = |Outp_A|$, n-ary functions determined by the algorithm A (if all possible input states are taken in account). We say that

these functions are evaluated by A .

If $q = 1$ then the unique function, which is evaluated by the algorithm A , will be denoted as f_A (or in a more detailed form as $f_A(x_1, \dots, x_n)$, or using the generalized operational command as $f_A(x_1, \dots, x_n) =: y_1$).

The algorithm abs , which evaluates the absolute value (2.5), is as follows:

$$(4.7)\quad abs = (\underbrace{\langle l_0, \overbrace{0 =: t}^{C^{(0)}} \rangle}_{K^{(0)}}; \underbrace{\langle l_1, \overbrace{x \geq t_{[l_2, l_4]}}^{C^{(1)}} \rangle}_{K^{(1)}}; \underbrace{\langle l_2, \overbrace{x =: y}^{C^{(2)}} \rangle}_{K^{(2)}};$$

$$\underbrace{\langle l_3, \overbrace{STOP}^{C^{(3)}} \rangle}_{K^{(3)}}; \underbrace{\langle l_4, \overbrace{-x =: y}^{C^{(4)}} \rangle}_{K^{(4)}}; \underbrace{\langle l_5, \overbrace{STOP}^{C^{(5)}} \rangle}_{K^{(5)}}) .$$

Then $B_1 = (K^{(0)}, K^{(1)}, K^{(2)}, K^{(3)})$ and $B_2 = (K^{(0)}, K^{(1)}, K^{(4)}, K^{(5)})$ are the only two branches of the algorithm abs ,

$Inp_{B_1} = Inp_{B_2} = Inp_A = \{x\}$, $Outp_{B_1} = Outp_{B_2} = Outp_A = \{y\}$ and $Par_{B_1} = Par_{B_2} = Par_A = \{t\}$.

Let Fct_{Alra} be the set of all functions which may be evaluated by algorithms over the algebra $Alra = \langle Obj, PVar, Opr, Rel_{[Obj]} \rangle$. Further let $Alra_1$ be the following algebra of the simplest arithmetic: Obj_1 is the set of all natural numbers, Opr_1 contains only the successor function $suc^{(1)}$ (i.e. $suc^{(1)}(x) = x + 1$ for $x = 0, 1, \dots$), Rel_1 contains only the equality relation

" $=_A$ " and $PVar$, Lab contains the small letters from the end, beginning of the Latin alphabet respectively, which are provided by indices being natural numbers.

Theorem 1. Fct_{Alra_1} contains all partial recursive functions, and, using the Church thesis, each function from Fct_{Alra_1} is a partial recursive function.

Proof. Let $0 \in Obj$ be the number zero. Then the following four algorithms evaluate the successor function $suc^{(1)}$, the zero function $zer^{(1)}(x) = 0$, the unary identity $id^{(1)}(x) = x$ and the binary identity $id^{(2)}(x,y) = y$, respectively:

$$Suc^{(1)} = (\langle l_1, suc^{(1)}(x) =: y\rangle;\ \langle l_2, STOP\rangle) \ldots\ldots Suc^{(1)}(x) =: y;$$

$$Zer^{(1)} = (\langle l_1, x =: x\rangle;\ \langle l_2, 0 =: x\rangle;\ \langle l_3, STOP\rangle) \ldots\ldots$$

$$\ldots\ldots Zer^{(1)}(x) =: x;\ Par_{zer} = \{x\};$$

$$Id^{(1)} = (\langle l_1, x =: x\rangle; \langle l_2, STOP\rangle) \ldots\ldots \ldots\ldots Id^{(1)}(x) =: x;$$

$$Id^{(2)} = (\langle l_1, x = x_{[l_2, l_2]}\rangle; \langle l_2, y =: y\rangle;\ \langle l_3, STOP\rangle) \ldots\ldots$$

$$\ldots\ldots Id^{(2)}(x,y) =: y.$$

It is known that starting with these four functions all partial recursive functions may be obtained by iterative applications of three operators (of superposition, of primitive recursion and the μ-operator). Therefore the theorem will be proved by the following three steps:

a) superposition: if the function $f_0^{(m)}$ is evaluated by the algorithm $F_0^{(m)}$ over $Alra_1$ and if

the function $f_i^{(m)}$ is evaluated by the algorithm $F_i^{(m)}$ for each $i = 1, 2, \ldots, m$, then the function

$$f^{(m)}(x_1, \ldots, x_m) = f_0^{(m)}(f_1^{(m)}(x_1, \ldots, x_m), \ldots, f_m^{(m)}(x_1, \ldots, x_m))$$

is evaluated by the following algorithm over $Alra_1$

$$F^{(m)} = (\langle l_1, F_1^{(m)}(x_1, \ldots, x_m) =: y_1 \rangle ; \langle l_2, F_2^{(m)}(x_1, \ldots, x_m) =: y_2 \rangle ;$$
$$\ldots ; \langle l_m, F_m^{(m)}(x_1, \ldots, x_m) =: y_m \rangle ; \langle l_{m+1}, F_0^{(m)}(y_1, \ldots, y_m)$$
$$=: x_0 \rangle ; \langle l_{m+2}, \text{STOP} \rangle) ,$$

where $Inp_{F^{(m)}} = \{x_1, \ldots, x_m\}$ and $Outp_{F^{(m)}} = \{x_0\}$;

b) primitive recursion: if the function $f^{(m)}, f^{(m+2)}$ is evaluated by the algorithm over $Alra_1$ $F^{(m)}$, $F^{(m+2)}$, respectively, then the function $f^{(m+1)}$, which is determined by the known requirements:

$$\begin{cases} f^{(m+1)}(x_1, \ldots, x_m, 0) = f^{(m)}(x_1, \ldots, x_m) , \\ f^{(m+1)}(x_1, \ldots, x_m, y+1) = f^{(m+2)}(x_1, \ldots, x_m, y, f^{(m+1)}(x_1, \ldots, x_m, y)) , \end{cases}$$

is evaluated by the following algorithm over $Alra_1$:

$$F^{(m+1)} = (\langle l_1, F^{(m)}(x_1, \ldots, x_m) =: w \rangle ; \langle l_2, 0 =: t \rangle ; \langle l_3, t =: y_{[l_4, l_5]} \rangle ;$$
$$\langle l_4, \text{STOP} \rangle ; \langle l_5, F^{(m+2)}(x_1, \ldots, x_m, t, w) =: w \rangle ;$$
$$\langle l_6, Suc^{(1)}(t) =: t \rangle ; \langle l_7, t = y_{[l_8, l_5]} \rangle ; \langle l_8, \text{STOP} \rangle) ,$$

where $Inp_{F^{(m+1)}} = \{x_1, \ldots, x_m, y\}$, $Outp_{F^{(m+1)}} = \{w\}$ and $Par_{F^{(m+1)}} = \{t\}$;

e) μ -operator: if the function $f^{(n+1)}$ is evaluated by the algorithm $F^{(n+1)}$ over $Alra_1$, then the function

$$f^{(n)}(x_1,\dots,x_m) = (\mu y)[f^{(n+1)}(x_1,\dots,x_m,y) = 0],$$

where on the right hand side the smallest integer y is denoted such that the equality in square brackets is satisfied (if there is no such integer then $f^{(n)}$ remains undefined in this case), is evaluated by the following algorithm over $Alra_1$:

$$F^{(n)} = (\langle l_1, 0 =: y\rangle; \langle l_2, F^{(n+1)}(x_1,\dots,x_m,y) =: z\rangle; \langle l_3, y = : t\rangle;$$
$$\langle l_4, z = t_{[l_5,l_7]}\rangle; \langle l_5, z = : z\rangle; \langle l_6, STOP\rangle; \langle l_7, Suc^{(1)}(y)$$
$$=: y\rangle; \langle l_8, F^{(n+1)}(x_1,\dots,x_m,y) =: z\rangle; \langle l_9, z = t_{[l_5,l_7]}\rangle),$$

where $Inp_{F^{(n)}} = \{x_1,\dots,x_m\}$, $Outp_{F^{(n)}} = \{z\}$ and $Par_{F^{(n)}} = \{y\}$.

5. Equivalencies of algorithms over algebras

Let us consider an algorithm A over an algebra and let $Inp_A = \{x_1,\dots,x_k\}$ and $Outp_A = \{y_1\}$ (see the previous Sect.4). If $\{o_1,\dots,o_k\} \in Domain\ f_A$ then by the application of the algorithm A to an arbitrary input state σ_0 such that $\sigma_0(x_i) = o_i$ for $i = 1, 2, \dots, k$, the branch B of A is chosen uniquely (by the requirements a), b) and c)). We say that the k -tuple $(o_1,\dots,o_k)$ belongs to the branch B and let Dom_B be the set of

all κ-tuples from Domain f_A which belong to B .

The mapping $Part_A$ such that Domain $Part_A = \{OB ; B \in B\kappa_A\}$ and Range $Part_A = \{Dom_B ; B \in B\kappa_A\}$ is called partialization of the algorithm A if

(5.1) $\quad Part_A(OB) = Dom_B$ for each $B \in B\kappa_A$.

Obviously

(5.2) $\quad Dom_B \cap Dom_{B'} \neq \emptyset$ if $B\kappa \neq B'$ and $\bigcup_{B \in B\kappa_A} Dom_B = Domain\ f_A$.

Now let us consider two algorithms A_1 and A_2 with unique output variable over the algebras $Alra_1$ and $Alra_2$. We say that A_1 is weaker than A_2 if there exists a mapping Φ such that Domain $\Phi = B\kappa_{A_1}$ and Range $\Phi \subset B\kappa_{A_2}$ which satisfies the following two requirements

(5.4) $\quad Dom_B \subset Dom_{\Phi(B)}$ for each $B \in B\kappa_{A_1}$,

(5.5) $\quad OB$ and $O(\Phi(B))$ determine equivalent terms for each $B \in B\kappa_{A_1}$.

The algorithms A_1 and A_2 are called branch equivalent if A_1 is weaker than A_2 and simultaneously A_2 is weaker than A_1 .

Theorem 2. If A_1 and A_2 are two branch equivalent algorithms with unique output variable over some algebras, then $f_{A_1} = f_{A_2}$, i.e. there exists a one-to-one correspondence φ between Inp_{A_1} and Inp_{A_2} such that

$$f_{A_1}(x_1,\dots,x_n) = f_{A_2}(\varphi(x_1),\dots,\varphi(x_n)) \quad .$$

Proof follows immediately by the facts that $f_{OB}(o_1,\dots,o_n) = f_A(o_1,\dots,o_n)$ for each $(o_1,\dots,o_n) \in Dom_B$ and each $B \in Br_A$, and further, that two equivalent terms always determine the same function.

On the other hand there are terms T_1 and T_2 which are not branch equivalent although $f_{T_1} = f_{T_2}$. E.g. in a numerical algebra one may take $T_1 = ((x+y)z)$ and $T_2 = ((xz)+(yz))$ when the usual distributive law is assumed, or in the minimal boolean algebra one may take $T_1 = (\neg(X \wedge Y))$ and $T_2 = ((\neg X) \vee (\neg Y))$, etc.

If we admit infinite algorithms and relations in logic with infinite many truth values (which are the symbols $\underline{1}, \underline{2}, \dots, \underline{i}, \dots$) then the following definition and construction are possible for each algebra $Alra = \langle Obj, PVar, Oij, Rel_{[Slab]} \rangle$ and each algorithm A over $Alra$ such that $Outp_A = \{y\}$ and $Inp_A = \{x_1, x_2, \dots, x_n\}$.

If $Br_A = \{B_1, B_2, \dots\}$, i.e. a numbering of all branches of A is assumed, then the following n-ary relation ${}_A\varrho^{(n)}_{(k)}$, where $k = |Br_A|$ may be infinite, may be introduced:

$$(5.6) \qquad {}_A\varrho^{(n)}_{(k)}(o_1,\dots,o_n) = \underline{i} \Longleftrightarrow (o_1,\dots,o_n) \in Dom_{B_i}$$

for $i = 1, 2, \dots$.

If $OB_i = (C_1^{(i)}; \dots; C_{n_i}^{(i)})$ then let

$AB_i = (\langle \mathfrak{l}_1^{(i)}, C_1^{(i)} \rangle ; \dots ;$
$\langle \mathfrak{l}_{n_i}^{(i)}, C_{n_i}^{(i)} \rangle ; \langle \mathfrak{l}_{n_i+1}^{(i)}, STOP \rangle)$ for each $i = 1, 2, \dots$, where it is assumed that $\mathfrak{l}_h^{(i)} \neq \mathfrak{l}_q^{(j)}$ if either $i \neq j$ or $h \neq q$ and therefore the following (may be infinite) algorithm is defined:

(5.7) $A^* = (\langle \mathfrak{l}_0, A\mathfrak{y}^{(n)}_{[\mathfrak{l}_1^{(1)}, \mathfrak{l}_1^{(2)}, \dots, \mathfrak{l}_1^{(i)}, \dots]}(x_1, \dots, x_n) \rangle ;$

$\langle \mathfrak{l}_1^{(1)}, C_1^{(1)} \rangle ; \dots ; \langle \mathfrak{l}_{n_1}^{(1)}, C_{n_1}^{(1)} \rangle ; \langle \mathfrak{l}_{n_1+1}^{(1)}, STOP \rangle ;$

$\langle \mathfrak{l}_1^{(2)}, C_1^{(2)} \rangle ; \dots ; \langle \mathfrak{l}_{n_2+1}^{(2)}, STOP \rangle ; \dots ; \langle \mathfrak{l}_1^{(i)}, C_1^{(i)} \rangle ; \dots ;$

$\langle \mathfrak{l}_{n_i}^{(i)}, C_{n_i}^{(i)} \rangle ; \langle \mathfrak{l}_{n_i+1}^{(i)}, STOP \rangle ; \dots)$

over the algebra $\mathfrak{Alg}^*$ such that

(5.8) $\mathfrak{Obj}^* = \mathfrak{Obj}$, $PVar^* = PVar$, $\mathfrak{Opr}^* = \mathfrak{Opr}, Rel^*$ contains the only relation (5.6) and $\mathfrak{Lab}^*$ contains as many labels as necessary, i.e. $\mathfrak{Lab}^*$ may be infinite.

Thus the following theorem is proved:

Theorem 3. If A is an arbitrary algorithm with only one output variable over the algebra $\mathfrak{Alg}$ then there exists an other algebra $\mathfrak{Alg}^*$ satisfying (5.8) and (5.6), and an algorithm A^* over $\mathfrak{Alg}^*$ satisfying (5.7) and such that A^* and A are branch equivalent. Informally speaking Theorem 3 asserts that each algorithm may be replaced by an other one which is branch equivalent with it and which requires just one single relation being tested at the beginning.

References

[1] BURSTALL Rod.M.: An algebraic description of programs with assertions, verification and simulation, SIGPLAN Notices Vol.7,No.1,Jan.1972, 7-14.

[2] ČULÍK K.: On semantics of programming languages, Automatentheorie und Formale Sprachen, editors J.Dörr-G.Hotz,Bibliographisches Institut,Mannheim-Wien-Zürich 1970,291-303.

[3] 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).

Research Institute for Mathematical Machines

Lužná ul.

Prague 6

Československo

(Oblatum 17.5.1972)