Aplikace matematiky

Karel Čulík

Syntactical definitions of program and flow diagram

Aplikace matematiky, Vol. 18 (1973), No. 4, 280–301

Persistent URL: http://dml.cz/dmlcz/103479

SYNTACTICAL DEFINITIONS OF PROGRAM AND FLOW DIAGRAM

KAREL ČULÍK

(Received October 19, 1972)

Weak and strong definitions of the program and the flow diagram are introduced, studied and compared using the branches of programs which correspond to the maximal pathes in flow diagrams. An algebraic characterization of flow diagrams is presented. The set of all possible branches is a regular event.

0. MOTIVATION OF PROBLEMS AND NOTATION

With respect to the execution of a program, which is considered as a finite sequence of commands, and with respect to the input data the unique sequence of commands from the program is determined, which will be called a *branch of the program*. In order to study all possible branches of a program it is sufficient to distinguish two main sorts of commands: let *Seq* be the set of all possible *sequential commands* which are characterized by the fact that within the program their right neighbouring commands (if any) will be executed as the next ones, and let *Dec* be the set of all possible *decision commands* characterized by a decision which command should be executed as the next one.

To be more concrete (see [1]) let each sequential command be a string of the form:

$$f^{(n)}(x_1, x_2, \ldots, x_n) =: x_0 \,, \tag{0.1}$$

where "$f^{(n)}$" is a symbol of an n-ary operation (either basic with respect to the computer under the consideration, or a composed one, which is determined by another program, i.e. a subprogram, or a procedure call, or a macro-command; this is allowed according to the sort of programming language we have in mind,) and x_i are proper individual variables or symbolic addresses for $i = 0, 1, \ldots, n$.

Further let ***START***, ***STOP*** be the well known special commands which are called the *starting* and the *stopping command*, respectively.

This paper was presented at the conference on Graph Theory, held in Štiřín in May 1972.

Two types of decision command are distinguished: a *branching command* has the form:

$$g^{(n)}_{[a_1,\ldots,a_k]}(x_1, x_2, \ldots, x_n)\,, \tag{0.2}$$

where "$g^{(n)}_{[a_1,\ldots,a_k]}$" is the symbol for an n-ary k-valued condition, which is a function assigning one from the set of labels $\{a_1, \ldots, a_k\}$ to an n-tuple of objects, which the computer is dealing with, and x_i are proper variables again for $i = 1, 2, \ldots, n$. It is assumed that $k \geqq 2$ and that "$g^{(n)}_{(k)}$" is the symbol of an n-ary relation in the k-valued logic, i.e., Field $g^{(n)}_{(k)} \subset Obj^n$ (or even Field $g^{(n)}_{(k)} = Obj^n$, if no partial relations are admitted) where Obj is the set of all (basic) objects the computer is dealing with, and further that the label a_i (which is a symbolic address as well, but distinguished from the proper variables) corresponds to the truth value $\boldsymbol{i}$ if the k values are $\{\boldsymbol{1}, \boldsymbol{2}, \ldots, \boldsymbol{k}\}$. Thus if $\boldsymbol{1} \equiv$ true, $\boldsymbol{2} \equiv$ false, $<^{(2)}_{(2)} \equiv <$ and Obj is the set of real numbers, then $<_{[a,b]}(x, y)$ is the function assigning the label b to the pair (3, 2), i.e. for $x = 3$, $y = 2$, because it is false that $3 < 2$, etc.

The second type of the decision command, called the *unconditional jump*, has the form:

$$a, \text{ where } a \text{ is a label}\,. \tag{0.3}$$

Now a *labelled command* is a pair $\langle b, C\rangle$ where b is a label and C a command.

To be quite exact (see [4]) let us assume that Lab, $PVar$ are sets of labels and proper variables, respectively, such that $Lab \cap PVar = \emptyset$; let $SymbOpr$, $SymbRel$ be the set of symbols of operations and relations in the Obj, respectively, and finally let $Sep = \{,\ ;\ =:\ (\ )\ [\ ]\ <>\ \boldsymbol{START}\ \boldsymbol{STOP}\}$ be the set of certain auxiliary symbols. Then all commands are strings over the alphabet $SymbOpr \cup SymbRel \cup Lab \cup PVar \cup Sep$ defined according to (0.1), (0.2) and (0.3). Therefore the set of all commands Com over this alphabet is defined perfectly.

The execution of a command C from Com is defined in [1, 4] with respect to a given state of storage σ, which is a function from the set of functions $(Obj \cup PVar \cup Com)^{Lab}$ in a natural way. The new state $\sigma^* = C\sigma$ is determined by $C = f^{(n)}(x_1, \ldots, x_n) =: x_0$ by the following requirement:

$$\begin{aligned} &\sigma^*(x_0) = f^{(n)}(\sigma(x_1)\,,\ \ \sigma(x_2), \ldots, \sigma(x_n))\,, \\ &\sigma^*(y) = \sigma(y) \quad \text{for each } y \in PVar \text{ such that } y \neq x_0\,, \end{aligned} \tag{0.4}$$

where, obviously, $f^{(n)}$ denotes an n-ary operation such that Doman $f^{(n)} \subset Obj^n$ and Range $f^{(n)} \subset Obj$.

The decision commands do not change the state; they determine only the next command which should be executed. This command depends on the given state if a branching command is executed. It is not necessary to go in all details in the definition of semantics here. It should be only stressed that all possible interpretations of symbols of operations and relations may be taken into account.

1. REQUIREMENTS CONCERNING SYNTACTICAL DEFINITIONS OF PROGRAM

The main aim of a program is to determine the detailed prescription of the computing process, which includes the information whether or not the intended computation has been completed, or in other words, whether or not the computing process has been interrupted before its completion for some reasons. Therefore the following (*weak*) *definition of program over* $\langle Com, Lab\rangle$ should be accepted:

A finite ordered set, which we consider as a sequence $P = (K^{(0)}, K^{(1)}, \ldots, K^{(N)})$ of labelled commands $K^{(i)} = \langle b^{(i)}, C^{(i)}\rangle$ where $b^{(i)} \in Lab$ and $C^{(i)} \in Com$ for $i = 0, 1, \ldots, N$, is called a *program* if the following requirements are satisfied:

(1.1) (i) $C^{(0)} = \boldsymbol{START}$ and $C^{(i)} \neq \boldsymbol{START}$ for all $i = 1, 2, \ldots, N$;

(ii) there exists i, where $1 \leqq i \leqq N$, such that $C^{(i)} = \boldsymbol{STOP}$

(iii) there exists at least one labelled branch of the program P, which is defined in (1.2);

(iv) $b^{(i)} \neq b^{(j)}$ where $i \neq j$ for each $i, j = 0, 1, \ldots, N$.

A finite sequence $LB = (K_0, K_1, \ldots, K_q)$ of labelled commands $K_i = \langle b_i, C_i\rangle$ where $K_i = K^{(j_i)}$ and $0 \leqq j_i \leqq N$ for each $i = 0, 1, \ldots, q$, is called a *labelled branch* of the program $P = (K^{(0)}, \ldots, K^{(N)})$, if the following requirements are satisfied:

(1.2) (i) $K_0 = K^{(0)}$,

(ii) $K_q = K^{(i)}$ where $1 \leqq i \leqq N$ and $C^{(i)} = \boldsymbol{STOP}$;

(iii) if $0 \leqq i \leqq q$ and $K_i = K^{(h)}$ where $0 \leqq h \leqq N$, then there occurs one of the following three possibilities:

a) $C^{(h)}$ is a sequential or starting command, and then $K_{i+1} = K^{(h+1)}$;

b) $C^{(h)}$ is a branching command, i.e. $C^{(h)} = g^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$, and there exist j, $1 \leqq j \leqq k$, and p, $1 \leqq p \leqq N$, such that $a_j = b^{(p)}$ and then $K_{i+1} = K^{(p)}$;

c) $C^{(h)}$ is an unconditional jump, i.e. $C^{(h)} = a$, and there exists p, $1 \leqq p \leqq N$, such that $a = b^{(p)}$ and then $K_{i+1} = K^{(p)}$.

The four requirements (1·1 i–iv) need no special clarification. The requirement (i) is a formal one, the requirement (ii) is necessary in order to have a possibility to find whether or not the program was completed. The requirement (iii) avoids certain programs which never can be completed, and the requirement (iv) follows naturally if we admit the following interpretation of a labelled command $\langle b, C\rangle$: the command C is stored at the memory cell with the abstract address b.

Besides the main aim of a program the further requirements concerning programs may be classified in the following three sorts:

(1.3) an interruption of the computing process (not caused by an error of the computer itself) before its completion should be caused only by the fact that some

operations and relations are partial, i.e. they are defined only for some n-tuples of objects;

(1.4) the computing process, which would be infinite, should be excluded in advance (when it is possible at all);

(1.5) there are no superfluous commands in the program, where a command is called *superfluous* if it is never executed with respect to all possible interpretations.

The requirement (1.3) is guaranteed by the following three requirements:

(1.1) (v) if $C^{(i)} = a$, where $1 \leqq i \leqq N$, then there exists j, $1 \leqq j \leqq N$, such that $b^{(i)} = a$;

(vi) if $C^{(i)} = g^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$, where $1 \leqq i \leqq N$, then for each integer j, $1 \leqq j \leqq k$, there exists an index h_j, $1 \leqq h_j \leqq N$, such that $b^{(h_j)} = a_j$;

(vii) either $C^{(N)} = \boldsymbol{STOP}$ or $C^{(N)}$ is a decision command.

The requirement (1.4) is guaranteed by the following requirement only partly:

(1.1) (viii) there is no ordered set of decision commands $\{C^{(i_1)}, C^{(i_2)}, \ldots, C^{(i_p)}\}$, where $p \geqq 1$ and $1 \leqq i_j \leqq N$ for $j = 1, 2, \ldots, p$, such that the following assertion is true for each j (mod p):
if $C^{(i_j)} = g^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$ then there exists and index h, $1 \leqq h \leqq k$, such that $a_h = b^{(i_{j+1})}$, and if $C^{(i_j)} = a$ then $a = b^{(i_{j+1})}$.

Finally the requirement (1.5) is guaranteed by the following requirement:

(1.1) (ix) each labelled command of the program belongs to at least one of its labelled branches.

The (*strong*) *definition of program* includes all nine requirements (1.1 i–ix) instead of the four requirements (1.1 i–iv) in the (weak) definition above. Throughout the paper, this definition of program is assumed.

Let LBr_P be the set (which may be infinite) of all labelled branches of the program P and let two programs P and Q be called *LBr-equivalent* if $LBr_P = LBr_Q$.

By any program $P = (K^{(0)}, K^{(1)}, \ldots, K^{(N)})$ the finite system $\bar{P} = \{P_1, P_2, \ldots, P_p\}$ of all subsequences P_i of P is determined uniquely by the following requirements:

(1.6) (i) each $P_i \in \bar{P}$ contains at least one $K^{(j)}$ such that $C^{(j)}$ is either a sequential or the starting command;

(ii) if $K^{(j)} \in P_i$, where $P_i \in \bar{P}$, and $C^{(j)}$ is either a sequential or the starting command, then $K^{(j+1)} \in P_i$ as well;

(iii) the length of each subsequence $P_i \in \bar{P}$ is the largest possible;

(iv) if $K^{(j)} \in P_i$, where $P_i \in \bar{P}$, and $C^{(j)}$ is neither a sequential nor the starting command, then $K^{(j+1)} \notin P_i$.

In addition, the fact that the program P is an ordered set implies

(1.6) (v) $P_i \cap P_j = \emptyset$ for $i \neq j$ where $i, j = 1, 2, \ldots, p$.

Theorem 1.1. *Two (strong) programs* $P = (K^{(0)}, \ldots, K^{(N)})$ *and* $Q = (L^{(0)}, \ldots, L^{(M)})$ *are LBr-equivalent if and only if* $N = M$ *and if there exists a permutation* π *of integers* $\{0, 1, \ldots, N\}$ *such that the following requirements are satisfied:*

(1.7) (i) $K^{(i)} \in P_h$, *where* $P_h \in \bar{P}$ *if and only if* $L^{(\pi(i))} \in Q_j$, *where* $Q_j \in \bar{Q}$;

(ii) *if* $(K_i, K_{i+1}, \ldots, K_{i+r}) \in \bar{P}$ and $K_i = K^{(h)}$ *then* $\pi(h + j) = \pi(h) + j$ for *each* $j = 1, 2, \ldots, r$;

(iii) $\pi(0) = 0$;

(iv) $K^{(i)} = L^{(\pi(i))}$ *for each* $i = 0, 1, \ldots, N$.

Proof. Let $N = M$ and let π be a permutation satisfying (1.7). We want to prove $LBr_P = LBr_Q$ and therefore, first of all, let us take an arbitrary labelled branch $LB = (K_0, K_1, \ldots, K_q) \in LBr_P$. We need to prove $LB \in LBr_Q$, i.e. that LB satisfies (1.2) with respect to Q.

According to (1.7 iv) $K_i = L^{(j_i)}$ where $0 \leqq j_i \leqq N$ for each $i = 0, 1, \ldots, N$, and it remains to verify (1.2 i–iii).

By (1.2 i) with respect to P it holds $K_0 = K^{(0)}$, by (1.7 iv) $K^{(0)} = L^{(\pi(0))}$ and therefore from (1.7 iii) it follows $K_0 = L^{(0)}$ which proves (1.2 i) with respect to Q.

By (1.2 ii) $K_q = K^{(i)}$, where $1 \leqq i \leqq N$ and $C^{(i)} = \textbf{\textit{STOP}}$, and using (1·7 iv) $K^{(i)} = L^{(\pi(i))}$ we get $K_q = L^{(\pi(i))}$ such that (1.2 ii) with respect to Q is satisfied.

If $0 \leqq i \leqq q$ and $K_i = K^{(h)}$, then again by (1.7 iv) one has $K_i = L^{(\pi(h))}$. If $C^{(h)}$ is a sequential or the starting command then $K_{i+1} = K^{(h+1)}$. In this case $K^{(h)} \in P_s$, where $P_s \in \bar{P}$ for some index s, and by (1.6 ii) also $K^{(h+1)} \in P_s$. Further by (1·7 ii) $\pi(h + 1) = \pi(h) + 1$ and therefore using (1.7 iv) again one has $K_{i+1} = L^{(\pi(h)+1)}$, which is the possibility a) in (1.2 iii) with respect to Q.

If $C^{(h)}$ is a decision command then $K_{i+1} = K^{(p)}$, where $K^{(p)}$ satisfies either b) or c) of (1.2 iii) with respect to P, then by (1.7 iv) $K^{(p)} = L^{(\pi(p))}$, which means that either b) or c) of (1.2 iii) is also satisfied with respect to Q.

Thus we have proved $LB \in LBr_Q$, and therefore $LBr_P \subset LBr_Q$. The remaining inclusion $LBr_Q \subset LBr_P$ will follow from the previous part of the proof for the permutation π^{-1} provided it is shown that also π^{-1} satisfies (1.7). This is immediately clear for (i), (iii) and (iv). Thus let us show it for (ii). Let us assume that $(L_j, L_{j+1}, \ldots, L_{j+r}) \in \bar{Q}$ and $L_j = L^{(s)}$. Then according to (1.7 i) and (1.7 iv) there is precisely one $(K^{(h)}, K^{(h+1)}, \ldots, K^{(h+r)}) \in \bar{P}$, where $h = \pi^{-1}(s)$. According to (1.7 ii) for π it is true that $\pi(h + t) = \pi(h) + t$ for $t = 1, 2, \ldots, r$, and according to the defining requirement (1.6) $L_{j+t} = L^{(s+t)}$ for $t = 1, 2, \ldots, r$, i.e., $\pi(h) = s$ and $\pi(h + t) = s + t$ for $t = 1, 2, \ldots, r$. Then $h + t = \pi^{-1}(\pi(h + t)) = \pi^{-1}(s + t)$ and $h + t = \pi^{-1}(s) + t$, i.e., $\pi^{-1}(a + t) = \pi^{-1}(s) + t$ for each $t = 1, 2, \ldots, r$, which is the requirement (1.7 ii) for π^{-1}. Consequently, $LBr_Q \subset LBr_P$ is proved.

Now it remains to assume $LBr_P = LBr_Q$ and then to prove $N = M$ and to find the permutation π which satisfies (1.7). If $N \neq M$ then we may assume that there exists a labelled command $L^{(j)} \in Q$ such that $L^{(j)} \notin P$ (and similarly in the contrary case if $P - Q \neq \emptyset$) and by (1.1 ix) there exists a labelled branch $LB \in LBr_Q$ which contains $L^{(j)}$. Therefore $LB \notin LBr_P$ and we obtained a contradiction $LBr_P \neq LBr_Q$.

Thus it must be $N = M$ and, moreover, by the previous argument P and Q must be set theoretically equal, i.e., there must exist a permutation π of integers $\{0, 1, \ldots, N\}$ such that (1.7 iv) is satisfied. With respect to this requirement and by the requirement (1.1 i), which is fulfilled by P and Q, it follows that (1.7 iii) is satisfied, too.

Further let us assume that (1·7 i) is not satisfied, i.e., there exist $P_h \in \bar{P}$ and $K^{(i)} \in P_h$ such that $L^{(\pi(i))}$ does not belong to any $Q_j \in \bar{Q}$. According to (1.6) $L^{(\pi(i))}$ must be a decision command and therefore $K^{(i)}$ must be the last element of the subsequence P_h (because by (1.7 iv) $K^{(i)} = L^{(\pi(i))}$). By (1·1 ix) there exists at least one labelled branch $LB \in LBr_P$ which contains $K^{(i)}$. If $K^{(i)} = K_s$ then $s > 0$ and according to (1.6) K_{s-1} must be a sequential or the starting command. On the other hand the command $L^{(\pi(i))}$ must be preceded in each branch from LBr_Q by another decision command, which means that $LB \notin LBr_Q$, and therefore $LBr \neq LBr_Q$. Thus (1.7 i) must be valid.

Finally let us assume that (1.7 ii) is not satisfied, i.e. there exists a subsequence $(K_i, K_{i+1}, \ldots, K_{i+r}) \in \bar{P}$, where $K_i = K^{(h)}$, and an integer, t, $1 \leqq t \leqq r$ such that $\pi(h + t) \neq \pi(h) + t$. We may assume that $\pi(h + t - 1) = \pi(h) + t - 1$ (i.e., t is the smallest possible). Using (1.7 iv) $K^{(h+t-1)} = L^{\pi(h+t-1)} = L^{\pi(h)+t-1}$, but $K^{(h+t)} = L^{\pi(h+t)} = L^{\pi(h)+t}$. By (1.1 ix) there exists at least one branch LB in LBr_P which contains the two commands $K^{(h+t-1)} = K_{i+t-1}$, $K_{i+t} = K^{(h+t)}$, but no branch in LBr_Q can contain the two commands $L^{\pi(h+t-1)} = L_s$, $L_{s+1} = L^{\pi(h+t)}$, because according to (1.6) the command L_s must be followed by $L^{\pi(h+t-1)+1} = L^{\pi(h)+t} \neq L^{\pi(h+t)}$. Thus $LB \notin LBr_Q$ and therefore $LBr_P \neq LBr_Q$, which is the required contradiction proving that (1.7 ii) must be satisfied. Now the proof is complete.

If $P = (K^{(0)}, \ldots, K^{(N)})$ is an arbitrary program then let S_P be the set of all integers j such that $K^{(j)}$ is the first (or leading) member of a subsequence $P_h \in \bar{P}$ (defined by (1.6)) and such that $1 \leqq j \leqq N$ (thus the subsequence with the leading member $K^{(0)}$ is not taken into account and therefore $|S_P|$ may be equal to zero). Further let R_P be the set of all integers j such that $K^{(j)}$ does not belong to any subsequence $P_h \in \bar{P}$ (thus $|R_P|$ may be equal to zero, too).

If π_S, π_R is an arbitrary permutation of S_P, R_P respectively, then the following permutation π of $\{0, 1, \ldots, N\}$ is determined uniquely as follows:

(1.8)
(i) $\pi(0) = 0$;
(ii) if $j \in S_P$ then $\pi(j) = \pi_S(j)$;
(iii) if $j \in R_P$ then $\pi(j) = \pi_R(j)$;
(iv) if $j \in \{1, 2, \ldots, N\} - (S_P \cup R_P)$ then there exists the largest integer $i \in S_P \cup \{0\}$ such that $i < j$ and $\pi(j) = \pi(i) + j - i$.

Theorem 1.2. *If* $P = (K^{(0)}, \ldots, K^{(N)})$ *is an arbitrary program then each program* Q *which is LBr-equivalent to* P *arises by reordering of* P *when a permutation* π *is used, i.e.,* $Q = (K^{\pi(0)}, K^{\pi(1)}, \ldots, K^{\pi(N)})$, *where* π *is defined by* (1.8) *from the both permutations* π_S, π_R *which are chosen arbitrarily. Thus there are* $(|S_P|!)$. $(|R_P|!)$ *different programs* Q, *each of which is LBr-equivalent to* P.

Proof follows easily from Theorem 1.1.

2. WEAKENING OF CONDITIONS

We intend to compare the weak and strong definition of the program.

Lemma 2.1. *Let* P *be a weak program and let* Q *arise from* P *by ommitting all labelled commands which are not contained in any labelled branch of* P. *Then* $LBr_Q = LBr_P$ *and the weak program* Q *satisfies the requirements* (1.1 ix), (1.1 v), (1.1 vii) *and the following assumption analogous to, but weaker than,* (1.1 vi):

(2.1) *if* $C^{(i)} = g^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$, *where* $1 \leqq i \leqq N$, *then there exist an integer* j, $1 \leqq j \leqq k$, *and an index* h_j, $1 \leqq h_j \leqq N$ *such that* $b^{(h_j)} = a_j$.

Proof is obvious.

If $C^{(i)} = g^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$ is a decision command of a weak program $P = (K^{(0)}, \ldots, K^{(N)})$ which satisfies (2.1) but, in general, not (1.1 vi), then the n-ary relation in the k-valued logic $g^{(n)}_{(k)}$, required by an interpretation of P, cannot be employed fully within P, and the same results will be obtained using another n-ary relation in the r-valued logic $g^{(n)}_{(r)}$, where $r < k$, which is defined as follows: let j_i be all integers such that $1 \leqq j_1 < j_2 < \ldots < j_r \leqq k$ and that there exists an index h_i, $1 \leqq h_i \leqq N$, which satisfies $b^{(h_i)} = a_{j_i}$. Now let a new condition be defined:

$$\tilde{g}^{(n)}_{[a_{j_1},a_{j_2},\ldots,a_{j_r}]}(x_1, \ldots, x_n) = {}_{\mathrm{df}} g^{(n)}_{[a_1,a_2,\ldots,a_k]}(x_1, \ldots, x_n)\,, \tag{2.2}$$

in every interpretation of the symbol of relation $g^{(n)}_{(k)}$. It is clear that $\tilde{g}^{(n)}_{(r)}$ is a partialization of $g^{(n)}_{(k)}$ which is determined by the prescription of a smaller range $\{a_{j_1}, a_{j_2}, \ldots, a_{j_r}\} \subset \{a_1, a_2, \ldots, a_k\}$ (instead of the more usual prescription of a smaller domain).

Lemma 2.2. *Let* P *be a weak program which satisfies* (1.1 ix), (1.1 v), (1.1 vii) *and* (2.1), *and let* Q *arise from* P *by replacing each condition, which does not satisfy* (1.1 vi), *by its partialization defined in* (2.2). *Then* Q *satisfies* (1.1 vi).

The proof is obvious.

It is easy to see that the requirement (1.1 viii) is independent of all the others.

Lemma 2.3. *A weak program P satisfies* (1.1 viii) *if and only if each labelled branch $LB \in LBr_P$ satisfies the following requirement*:

(2.3) *if $LB = (K_0, K_1, \ldots, K_q)$ then there exists an integer j, $0 < j < q$, such that $K_j = K_{j+p}$, where $p \geqq 1$, and K_{j+t} is a decision command for each $t = 1, 2, \ldots, p$.*

The proof is obvious.

It remains an open problem whether or not to each weak program P, which satisfies (1.1 vi), there exists a strong program Q such that the following assertion is valid:

(2.4) $LBr_Q = \{LB;\ LB \in LBr_P$ and LB satisfies (2.3)$\}$.

Lemma 2.4. *A strong program $P = (K^{(0)}, \ldots, K^{(N)})$ satisfies the following requirements*:

(2.5) *if $C^{(i)}$ is a decision command, where $1 \leqq i < N$, then there exists a decision command $C^{(h)}$, $1 \leqq h \leqq N$, such that $h \neq i + 1$, and if $C^{(i)} = g^{(n)}_{a[1,\ldots,a_k]}(x_1, \ldots, x_n)$, then there exists an index j, $1 \leqq j \leqq k$, such that $a_j = b^{(i+1)}$, and if $C^{(i)} = a$, then $a = b^{(i+1)}$;*

(1.1v*) *is identical with* (1.1 v), *where in addition $j \neq i$ is required*;

(1.1vi*) *is identical with* (1.1 vi), *where in addition $h_j \neq i$ is required.*

The proof is obvious.

3. FLOW DIAGRAM OF A PROGRAM

The flow-diagram of a (strong) program $P = (K^{(0)}, \ldots, K^{(N)})$ is an oriented graph $FD_P = \langle V, \varrho, \lambda, \Lambda \rangle$ with labelled vertices and edges, which is defined as follows:

(3.1) (i) $V = \{b^{(0)}, b^{(1)}, \ldots, b^{(N)}\}$;

(ii) a) if $C^{(i)}$ is a sequential or the starting or the stopping command, then $\lambda(b^{(i)}) = C^{(i)}$;

b) if $C^{(i)} = g^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$ then $\lambda(b^{(i)}) = g^{(n)}_{(k)}(x_1, \ldots, x_n)$;

c) if $C^{(i)} = a$ then $\lambda(b^{(i)}) = \boldsymbol{GOTO}$, where "***GOTO***" is a new symbol (corresponding to "go to" in programming languages) and $0 \leqq i \leqq N$;

(iii) a) if $C^{(i)} = \boldsymbol{STOP}$ then there is no edge starting at $b^{(i)}$;

b) if $C^{(i)} \neq \boldsymbol{STOP}$ is a sequential command or $C^{(i)} = \boldsymbol{START}$ then there is a unique edge $(b^{(i)}, b^{(i+1)})$ starting at $b^{(i)}$;

c) if $C^{(i)} = g^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$ then there are as many edges starting at $b^{(i)}$ as there are different labels a_j, viz. the edges $(b^{(i)}, a_j)$ for each $j = 1, 2, \ldots, k$;

d) if $C^{(i)} = a$ then there is the unique edge $(b^{(i)}, a)$ starting at $b^{(i)}$, where $1 \leqq i \leqq N$, and there are no other edges in ϱ than those defined above;

(iv) if $C^{(i)} = g^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$, where $1 \leqq i \leqq N$, then $\Lambda(b^{(i)}, a_j) = \{h;\ h \text{ is an integer such that } 1 \leqq h \leqq k \text{ and } a_h = a_j\}$ for each $j = 1, 2, \ldots, k$ and for each of the remaining edges (x, y) one defines $\Lambda(x, y) = \{1\}$.

According to (3.1 ii) the range of the labelling λ of vertices of FD_P is the set consisting of certain sequential commands of the form "$f^{(n)}(x_1, \ldots, x_n) =: x_0$", of symbols of certain n-ary relations in the k-valued logic $g^{(n)}_{(k)}(x_1, \ldots, x_n)$, where x_i is a proper variable for $i = 0, 1, \ldots, n$, and of symbols ***START***, ***STOP*** and ***GOTO***.

According to (3.1 iv) the range of the labelling Λ of edges of FD_P is the set consisting of finite sets of integers. Only in the special case when the following condition is satisfied:

(3.2) if $C^{(i)} = g^{(n)}_{[a_1,\ldots,a_k]}(x_1, \ldots, x_n)$, where $1 \leqq i \leqq N$, then $a_j \neq a_h$ for $j \neq h$ and for all $j, h = 1, 2, \ldots, k$, where always $k \geqq 2$,

the range of Λ is the set of particular integers only (i.e. no sets of integers are necessary).

Without a loss of generality the requirement (3.2) will be accepted in the sequel. This requirement only means that some new relations and symbols of relations are introduced in advance. E.g. $g^{(2)}_{[a,a,b]}$ is a binary relation only in the 2-valued logic (although it arised from $g^{(2)}_{(3)}$) which does not satisfy (3.2) and therefore one assumes that a new symbol "$h^{(2)}_{[a,b]}$" is available which satisfies (3.2) and which is defined as follows: $h^{(2)}_{[a,b]}(x, y) = {}_{\mathrm{df}} g^{(2)}_{[a,a,b]}(x, y)$. Obviously a different relation is defined by $k^{(2)}_{[a,b]}(x, y) = {}_{\mathrm{df}} g^{(2)}_{[a,b,a]}(x, y)$, etc. If $k = 2$ and (3·2) is not satisfied, e.g. "$\langle_{[a,a]}(x, y)$" then it is clear that this degenerated decision command may be replaced by the unconditional jump "a". Therefore, if the requirement (3.2) is not satisfied then the notion of unconditional jump is superfluous.

It is useful to call a vertex of FD_P *sequential vertex* or a *decision vertex*, respectively if its label (in the labelling λ) is a sequential or a decision command.

Theorem 3.1. *An oriented graph $G = \langle V, \varrho, \lambda, \Lambda\rangle$ which is isomorphic with respect to the labellings λ and Λ to a flow diagram of a program satisfying* (3.2), *satisfies the following requirements* (3.3) *and* (3.4):

(3.3) (i) *there exists exactly one input vertex $w \in V$; $\lambda(w) =$ **START**; there exists exactly one edge (w, x) starting at w and $\Lambda(w, x) = 1$;*

(ii) *there exists at least one output vertex*; if $w \in V$ *is an output vertex then $\lambda(w) =$ **STOP** and there is no edge starting at w;*

(iii) *if $w \in V$ is an inner vertex and $od(w) = 1$ then either $\lambda(w) =$ **GOTO** or $\lambda(w)$ is a sequential command and $\Lambda(w, x) = 1$ holds for the unique edge (w, x) starting in w;*

(iv) *if* $w \in V$ *is an inner vertex and* $od(w) = k > 1$ *then* $\lambda(w) = g_{(k)}^{(n)}(x_1, \ldots, x_n)$, *where* $g_{(k)}^{(n)}$ *is the symbol of a relation in the k-valued logic and all k edges starting at w are labelled by integers* $1, 2, \ldots, k$ *in the labelling* Λ;

(viii) *each cycle in G contains at least one sequential vertex and there are no slings*;

(ix) *each vertex belongs at least to one* (monotonic) *path which starts at the input vertex and terminates at an output vertex*;

(3.4) *if* $w \in V$ *then there exists at most one vertex* $v \in V$ *such that* $(v, w) \in \varrho$ *and either* $\lambda(v) =$ ***START*** *or* $\lambda(v)$ *is a sequential vertex.*

Proof. We need to prove that every flow diagram satisfies (3.3) and (3.4) if the labels are considered as arbitary abstract symbols. Let us show it consecutively (although not in all details):

(3.3) (i) follows from (1.1 i), (3.1 ii-a), (3.1 iii-b) and (3.1 iv);
(ii) follows from (1.1 ii), (3.1 ii-a), (3.1 iii-a) and (3.1 iv);
(iii) follows from (3.1 iii-d), (3.1 iii-b), (3.1 ii-c), (3.1 ii-a) and (3.1 iv);
(iv) follows from (3.1 ii-b), (3.1 iii-c), (3.2) and (3.1 iv);
(viii) follows from (1.1 viii),

which also implies that there are no slings at the decision vertices (according to (1.1 v*) and (1.1 vi*)); for the remaining inner vertices this follows by the definition of the labelled branch (1.2) and from (3.1 ii-a) and (3.1 iii-b);

(3.3) (ix) follows from (1.1 ix);

(3.4) follows from the assertion (1.6 iv) which concerns the subsequences $P_h \in \bar{P}$.

Before showing the synthesis of a program for a prescribed flow diagram let us describe a certain construction on a graph $G = \langle V, \varrho, \lambda, \Lambda \rangle$ which corresponds to the construction of the finite system $\bar{P}$ of subsequences of the program P (see the requirement (1.6)).

If G satisfies (3.3) and (3.4) then let $\bar{G}$ be the set of all subgraphs $G_i = \langle V_i, \varrho_i, \lambda_i, \Lambda_i \rangle$ of G for $i = 1, 2, \ldots, p$, which are uniquely determined by the following requirements:

(3.5) (i) at least one vertex $w \in V_i$ satisfies the requirement that $\lambda(w)$ is either a sequential command or the starting command;
(ii) if $w \in V_i$ and $\lambda(w)$ is either a sequential or the starting command then there exists just one vertex $v \in V$ such that $(w, v) \in \varrho$ and also $v \in V_i$;
(iii) the number of vertices in V_i is the largest possible;
(iv) if $w \in V_i$ and $\lambda(w)$ is neither a sequential nor the starting command, then there exists no $v \in V$ such that $v \in V_i$ and $(w, v) \in \varrho$;
(v) $\varrho_i = V_i^2 \cap \varrho$; $\lambda_i(w) = \lambda(w)$ for each $w \in V_i$ and $\Lambda_i(w, v) = \Lambda(w, v)$ for each $(w, v) \in \varrho_i$.

Lemma 3.2. *The subgraphs G_i of $\bar{G}$ defined by* (3.5) *satisfy the following requirements:*

(3.6) (i) *G_i is a simple path, the length of which is at least one;*

(ii) *the output vertex in G_i is either a decision vertex or it is labelled by the stopping command;*

(iii) *if $w \in V$ and $\lambda(w)$ is either a sequential command or* ***START****, then there exists G_i such that $w \in V_i$;*

(iv) *$p \geqq 1$ and $V_i \cap V_j = \emptyset$ for $i \neq j$ and each $i, j = 1, 2, \ldots, p$.*

Proof. The assertion (3.6 i) follows from (3.5 i), (3.5 ii) and from (3.3). The assertion (3.6 ii) follows from (3.5 iii). The assertion (3.6 iii) follows from the definition of $\bar{G}$. Finally, let us assume $V_i \cap V_j \neq \emptyset$ and $V_i \neq V_j$ for some $i \neq j$, and let $(v_{h,1}, v_{h,2}, \ldots, v_{h,q_h})$ be the path G_h for $h = i, j$, i.e. there exist $v_{i,r} = v_{j,s}$ for some $1 \leqq r \leqq q_i$, $1 \leqq s \leqq q_j$ such that r and s are the smallest possible, which means that $\{v_{i,1}, \ldots, v_{i,r-1}\} \cap \{v_{j,1}, \ldots, v_{j,s-1}\} = \emptyset$. If $r < q_i$, then $\lambda(v_{i,r})$ is either a sequential or the starting command. If, in addition, $r = s = 1$, then by (3.3 i), (3.3 iii) and (3.5 ii) it must be $V_i = V_j$, which contradicts our assumption. If $r = 1 < s$ then V_i does not satisfy (3.5 iii), which is a contradiction again. Finally if $1 < r$, $1 < s$ then (3.4) cannot be valid which is the required contradiction which completes the proof.

Algorithm 3.3. (Synthesis of a program). *If a graph $G = \langle V, \varrho, \lambda, A \rangle$ satisfying* (3.3) *and* (3.4) *is prescribed then a program P such that FD_P is isomorphic to G may be constructed as follows:*

1) *One takes the set $\bar{G}$ of G defined by* (3.5) *and for each $G_i \in G$ one chooses such ordering of its vertices $v_{i,j} \in V_i$ for $j = 1, 2, \ldots, q_i$ that $(v_{i,1}, v_{i,2}, \ldots, v_{i,p_i})$ is a simple path of all of them, where $i = 1, 2, \ldots, p$. Further let $V_{p+1} = V - \bigcup_{i=1}^{p} V_i$ and let $(v_{p+1}, v_{p+1,2}, \ldots, v_{p+1,q_{p+1}})$ be an arbitrary ordering of V_{p+1}. Let us assume that the vertex $v_{1,1}$ is the input vertex of G.*

2) *The following ordering of all vertices from V may be chosen:*

$$(3.7)\quad (v_{1,1}, \ldots, v_{1,q_1}, v_{2,1}, \ldots, v_{2,q_2}, \ldots, v_{p,1}, \ldots, v_{p,q_p}, v_{p+1,1}, \ldots, v_{p+1,q_{p+1}}) .$$

Further if $|V| = \sum_{i=1}^{p+1} q_i = N + 1$ then $N + 1$ different labels may be chosen arbitrarily. If the labels are denoted by $b^{(0)}, b^{(1)}, \ldots, b^{(N)}$, then their ordering with respect to their upperscripts is determined by the following one-to-one mapping φ:

(3.8) $\varphi(v_{i,j}) = b^{(h)}$ *where* $h = q_1 + q_2 + \ldots + q_{i-1} + j$ $(q_0 = 0)$ *for each* $1 \leqq i \leqq p$ *and* $1 \leqq j \leqq q_i$.

3) *The required commands of the program are now defined uniquely as follows: if* $v \in V$ *and* $\lambda(v) =$ ***GOTO*** *then one takes the unique vertex* $w \in V$ *such that* $(v, w) \in \varrho$ *and defines a new function* $\psi(v) = \varphi(w)$; *if* $v \in V$ *and* $\lambda(v) = g_{(k)}^{(n)}(x_1, \ldots, x_n)$ *then (with respect to* (3.2)*) for each* j, $1 \leqq j \leqq k$, *there is a unique vertex* $w_j \in V$ *such that* $(v, w_j) \in \varrho$ *and* $\Lambda(v, w_j) = j$, *and one defines the new function* $\psi(v) = g_{[\varphi(w_1), \varphi(w_2), \ldots \varphi(w_k)]}^{(n)}(x_1, \ldots, x_n)$. *In all other cases of* $v \in V$ *one defines* $\psi(v) = \lambda(v)$.

4) *The required program* $P = (K^{(0)}, \ldots, K^{(N)})$ *is defined by determining* $K^{(h)} = \langle \varphi(v_{i,j}), \psi(v_{i,j}) \rangle$ *where* h *is defined by* (3.8) *for each* i j.

The proof of correctness. Let $FD_P = \langle V', \varrho', \lambda', \Lambda' \rangle$ be the flow diagram of the program P. It is sufficient to prove that the one-to-one mapping φ is an isomorphism of G onto FD_p. If we recall that there is a one-to-one correspondence between the finite system $\overline{P}$ of P defined by (1.6) and the set $\overline{G}$ of G defined by (3.6), then with respect to (3.1 ii) one easily recognises that φ preserves the labelling of vertices. It remains to consider consecutively all four possible cases a), b), c) and d) of (3.1 iii) which concern the edges of FD_P and to show that the main requirement of isomorphism: if $(x, y) \in \varrho$ then $(\varphi(x), \varphi(y)) \in \varrho'$, is satisfied. Finally, with respect to (3.1 iv) one immediately sees that also the labelling of edges is preserved by φ.

Theorem 3.4. *Each program* Q *such that* FD_Q *is isomorphic to* FD_P, *where* P *is a prescribed program, may be received as follows: using a one-to-one mapping* τ *each label* b *of* P *is replaced by* $\tau(b)$ *every time it appears in* P *and then a reordering from Theorem* 1.2 *is carried out.*

Proof. The first part follows from Theorem 3.1 and the second from Theorem 1.2.

A (*non-labelled*) *branch* of a program P is a sequence $B = (D_0, D_1, \ldots, D_q)$ which arises of a labelled branch $LB = (K_0, K_1, \ldots, K_q) \in LBr_P$ as follows:

(3.9) if $K_i = \langle b_i, C_i \rangle$ where $1 \leqq i \leqq q$ then either a) C_i is a sequential command and then $D_i = {}_{\mathrm{df}}[C_i, 1]$, or b) $C_i =$ ***STOP*** and then $D_i = [$***STOP***$, 0]$, or c) $C_i \in \{$***START***, ***GOTO***$\}$ and then $D_i = [C_i, 1]$ or d) $C_i = g_{[a_1, \ldots, a_k]}^{(n)}(x_1, \ldots, x_n)$ and then $D_i = [g_{(k)}^{(n)}(x_1, \ldots, x_n), h]$ where h is an integer such that $1 \leqq h \leqq k$ and $a_h = b_{i+1}$.

Let Br_P be the set of all (non-labelled) branches of P.

Lemma 3.5. *If* $LB_1, LB_2 \in LBr_P$, $LB_1 \neq LB_2$ *and* $B_i \in Br_P$ *arised from* LB_i *by* (3.9) *for* $i = 1, 2$, *then* $B_1 \neq B_2$. *There fore there is a one-to-one map* β *of* LBr_P *onto* Br_P *which assigns to the labelled branch* $LB \in LBr_P$ *such a branch* $B \in Br_P$ *which arised from* LB *by* (3.9).

Proof. Let $LB_i = (K_{0,i}, K_{1,i}, \ldots, K_{q_i,i})$ for $i = 1, 2$. Then it follows from $LB_1 \neq LB_2$ that there exists an integer j such that $1 \leqq j \leqq \min(q_1, q_2)$ and $K_{h,1} = K_{h,2}$ for $0 \leqq h \leqq j$, but $K_{j,1} \neq K_{j,2}$. Therefore $K_{j-1,1}$ must be a decision command and it follows from (3·1 iv) that if $D_{j,i} = [C_{j,i}, h_{j,i}]$ arised from $K_{j,i}$ for $i = 1, 2$, then $h_{j,1} \neq h_{j,2}$ and therefore $B_1 \neq B_2$.

Further, if $G = \langle V, \varrho, \lambda, \Lambda \rangle$ is a graph isomorphic to a flow diagram of a program then a *branch of* G is a sequence $B = (D_0, D_1, \ldots, D_q)$ determined by a monotonic path $(v_0, e_1, v_1, \ldots, e_q, v_q)$ of vertices and edges in G such that v_0 is the input vertex and v_q is an output vertex, as follows:

$$(3.10)\quad D_i = [\lambda(v_i), \Lambda(e_{i+1})] \quad \text{for} \quad i = 0, 1, \ldots, q-1 \quad \text{and} \quad D_q = [\lambda(v_q), 0].$$

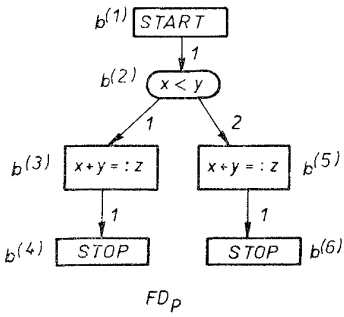
Let Br_G be the set of all branches of G and let two programs P and Q be *Br-equivalent* if $Br_P = Br_Q$. In addition one can define the *Br-equivalence* also *for one program* P and *one graph* G by the requirement $Br_P = Br_G$, which allows to compare the programs and the flow diagrams.

Theorem 3.6. *$Br_{FD_P} = Br_P$ for every program P, i.e., the program and its flow diagram are Br-equivalent.*

Proof. Using the definition (3.1) a one-to-one correspondence is determined between the set LBr_P and the set of all monotonic pathes in FD_P which start at the input vertex and terminate at an output vertex of FD_P. The required equality follows by (3.9) and (3.10).

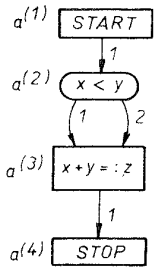
Theorem 3.7. *If G is isomorphic to FD_P for a program P then $Br_G = Br_{FD_P}$.*

The proof follows from Theorems 3.4 and 3.6.



FD_P

Fig. 1



FD_Q

Fig. 2

On the other hand, it must be pointed out that the equality $Br_{FD_Q} = Br_{FD_P}$ (or $Br_Q = Br_P$) may hold also in the case when FD_Q is not isomorphic to FD_P, which is proved by the two flow diagrams in Fig. 1 and in Fig. 2. In Fig. 1 the condition "$x < y$" is superfluous in the sense that the branching, which it allows, leads to the same remaining parts of branches. The flow diagram FD_P in Fig. 2 of another program, which does not satisfy (3.2), is not isomorphic to FD_Q in Fig. 1, although $Br_{FD_P} = Br_{FD_Q} = \{(\boldsymbol{START}, x < y, x + y =: z, \boldsymbol{STOP}\}$.

Using the synthesis algorithm 3.3 one obtains:

$$Q = (\langle b^{(1)}, \boldsymbol{START}\rangle\ ;\ \langle b^{(2)}, x < y_{[b^{(3)}, b^{(5)}]}\rangle\ ;\ \langle b^{(3)}, x + y =: z\rangle\ ;$$

$$\langle b^{(4)}, \boldsymbol{STOP}\rangle\ ;\ \langle b^{(5)}, x + y =: z\rangle\ ;\ \langle b^{(6)}, \boldsymbol{STOP}\rangle \quad \text{and} \quad P = (\langle a^{(1)}, \boldsymbol{START}\rangle$$

$$\langle a^{(2)}, x < y_{[a_3, a_3]}\rangle\ ;\ \langle a^{(3)}, x + y =: z\rangle\ ;\ \langle a^{(4)}, \boldsymbol{STOP}\rangle).$$

As it was mentioned the decision command of Fig. 1 and of Fig. 2 is superfluous, and actually only the program $R = (\langle c^{(1)}, \boldsymbol{START}\rangle; \langle c^{(2)}, x + y =: z\rangle; \langle c^{(3)}, \boldsymbol{STOP}\rangle)$, the flow diagram FD_R of which is in Fig. 3, is sufficient for the required

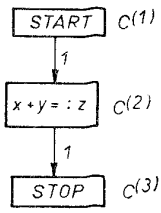


Fig. 3 FD_R

algorithm. In order to be able to compare also such flow diagrams and the corresponding programs, the concept of the operational branch should be introduced as follows.

An *operational branch of a program* P is a sequence $OB = (E_1, E_2, \ldots, E_p)$ of sequential=operational commands only, which arises from a (non-labelled) branch $B = (D_0, D_1, \ldots, D_q)$ of P (or of G) by omitting all commands which are not sequential commands, and all the second members of pairs D_j (see (3.9) or (3·10)), i.e. the integers r_j from $D_j = [C_j, r_j]$. Let OBr_P be the set of all operational branches of P and, similarly, let OBr_G be the set of all operational branches of G, where G is isomorphic to the flow diagram of a program. The programs P and Q are called *OBr-equivalent* if $OBr_P = OBr_Q$.

Thus one may write $OBr_P = OBr_Q = OBr_R$. The characterization of the *Br*-equivalency and the *OBr*-equivalency of programs is clarified as follows.

It can be proved (but it is beyond the frame of this paper) that the *Br*-equivalence of two programs is a sufficient condition for their functional equivalence, i.e. in the same computer the same result is computed by two *Br*-equivalent programs.

Further, it is clear (by the definitions of non-labelled and operational branches) that if $Br_P = Br_Q$ then also $OBr_P = OBr_Q$ for all programs P and Q. The converse assertion is not valid, which is shown by Fig. 4 and 5 where flow diagrams of two programs P_1 and Q_1 are shown such that $OBr_{P_1} = OBr_{P_2}$ but $BR_{P_1} \neq Br_{Q_1}$. Moreover it is immediately seen that P_1 and Q_1 compute different functions, viz P_1 computes $|x|$ and Q_1 computes $-|x|$.

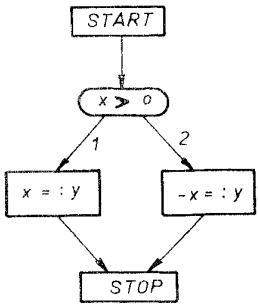


Fig. 4

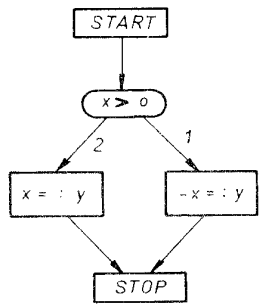


Fig 5

4. FLOW DIAGRAM AND ITS EXECUTION

The execution of a program requires the determination of the next state and of the next command, i.e. which command should be executed as the next one (see Sect. 0). The same may be determined also for an oriented graph $G = \langle V, \varrho, \lambda, \Lambda \rangle$ which satisfies (3.3) independently of the requirement (3.4). The current state is changed only by a sequential command which is the label in λ of a vertex w, while the command which should be executed next is that one by which the unique vertex v, such that $(w, v) \in \varrho$, is labelled in λ. If a decision command, which is the label of the vertex w, is considered and executed, then with respect to the current state its value is a truth value i and therefore the unique vertex v is determined such that $(w, v) \in \varrho$ and $\Lambda(w, v) = i$. Hence $\lambda(v)$ should be executed as the next command.

Construction (Linearization) **4.1.** If an oriented graph $G = \langle V, \varrho, \lambda, \Lambda \rangle$ satisfies (3.3) but does not satisfy (3.4), then an oriented graph $G' = \langle V', \varrho', \lambda', \Lambda' \rangle$ may be constructed as follows: consecutively (in an arbitrary order) each vertex $w \in V$ is considered such that if $w_1, w_2, \ldots, w_n \in V$ are all those vertices such that $(w_1, w) \in \varrho$ and $\lambda(w_i, w)$ is either a sequential or the starting command for each $i = 1, 2, \ldots, n$, then $n \geqq 2$; if w is the vertex under the consideration, then $n - 1$ new vertices $t_1, t_2, \ldots, t_{n-1}$ are chosen and added to V, further the edge (w_i, w) is omitted from ϱ and replaced by two new edges (w_i, t_i) and (t_i, w), which are added to ϱ for $i =$

$= 1, 2, \ldots, n-1$ (which may be done in n different ways); and finally one defines $\lambda'(t_i) = \textbf{\textit{GOTO}}$, $\Lambda'(w_i, t_i) = \Lambda(w_i, t_i)$ and $\Lambda'(t_i, w) = 1$ for $i = 1, 2, \ldots, n-1$, while in all the remaining cases λ and Λ remain unchanged.

Let us write $Br_G = Br_{G'} \pmod{\textbf{\textit{GOTO}}}$ if ${}^{\#}\widetilde{Br_G} = {}^{\#}\widetilde{Br_{G'}}$ where ${}^{\#}\widetilde{Br_G}$ arises from Br_G as follows: each branch of ${}^{\#}\widetilde{Br_G}$ arises from a branch $(D_0, D_1, \ldots, D_q)$ of Br_G by omitting all pairs D_j, which contain "***GOTO***".

Theorem 4.2. *If G satisfies* (3.3) *and G' arises from G by the construction* 4.1, *then G' satisfies* (3.3) *and* (3.4), $Br_{G'} = Br_G \pmod{\textbf{\textit{GOTO}}}$, *and therefore* $OBr_{G'} = OBr_G$. *If G satisfies* (3.3) *then there exists G^* which satisfies* (3.3) *and* (4.1), $Br_{G^*} = Br_G \pmod{\textbf{\textit{GOTO}}}$, *and therefore* $OBr_{G^*} = OBr_G$, *where*:

(4.1) *there is no vertex labelled by "**GOTO**".*

Proof. The first part follows immediately by the construction 4.1, because it is sufficient to show it for the single vertex under the consideration. The second part requires a converse construction to the construction 4.1, i.e., the omission of vertices which are labelled by ***GOTO***, which is defined easily.

Now it is clear that to each oriented graph $G = \langle V, \varrho, \lambda, \Lambda \rangle$, which satisfies (3.3) and (4.1), a program P may be constructed such that $Br_G = Br_{FD_P} \pmod{\textbf{\textit{GOTO}}}$, and therefore it is possible to define this graph G as a flow diagram without any respect to the program as in Sect. 3. The omission of all unconditional jumps and the omission of (3.4) enables us to give the following definition of a flow diagram (being a reformulation of (3.3)) which is clearer and more suitable for our purpose:

(4.2) (i) there exists exactly one input vertex;
(ii) there exists at least one output vertex;
(iii) each vertex belongs at least to one (monotonic) path which starts at the input vertex and terminates at an output vertex;
(iv) each cycle contains at least one sequential vertex;

(4.3) by the labelling λ of vertices the input vertex is labelled by "***START***", each output vertex is labelled by "***STOP***", each sequential vertex is labelled by an operational command, and each decision vertex w with $\mathrm{od}(w) = k \geqq 2$ is labelled by a decision command in the k-valued logic;

(4.4) by the labelling Λ of edges all k edges starting at the vertex w such that $\mathrm{od}(w) = k \geqq 1$ are labelled by k integers $1, 2, \ldots, k$;

where the particular requirement (4.2), (4.3) and (4.4) are classified according to the graph structure $\langle V, \varrho \rangle$ itself, the labelling λ of vertices and the labelling Λ of edges, respectively.

It should be mentioned that (4.3) says that the types of commands uniquely correspond to the types of vertices.

5. REGULAR LANGUAGES OF LABELLED BRANCHES

Let $P = (K^{(0)}, K^{(1)}, \ldots, K^{(N)})$, where $K^{(i)} = \langle b^{(i)}, C^{(i)} \rangle$ be a program and $FD_P = \langle V, \varrho, \lambda, \Lambda \rangle$ its flow diagram. Let us define a new graph $G_P = \langle V, \varrho, \varkappa \rangle$ where $\varkappa$ is the following labelling of vertices in G_P: $\varkappa(b^{(i)}) = K^{(i)}$ for $i = 0, 1, \ldots, N$. Further let us shift the labels of vertices on those edges which terminate at them, i.e., let us define $G_P^* = \langle V, \varrho, \varkappa^* \rangle$ by the following requirement:

$$\varkappa^*(x, y) = {}_{\mathrm{df}}\varkappa(y) \quad \text{for each} \quad (x, y) \in \varrho \,. \tag{5.1}$$

If we assume that every monotonic path $(v_0, v_1, \ldots, v_q)$ in G_P where v_0, v_q is the input vertex and an output vertex, respectively, generates the labelled branch $(\varkappa(v_0), \varkappa(v_1), \ldots, \varkappa(v_q))$, then we may say that the same path in G_P^* generates the sequence of labelled commands $(\varkappa^*(v_0, v_1), \varkappa^*(v_1, v_2), \ldots, \varkappa^*(v_{q-1}, v_q))$. If LBr_{G_P} and $LBr_{G^*_P}$ are the sets of all sequences of labelled commands generated by G_P and G_P^*, respectively, then it is clear that the following assertion is valid:

$$LBr_P = LBr_{G_P} = K^{(0)} LBr_{G^*_P} \,, \tag{5.2}$$

where $K^{(0)} LBr_{G^*_P}$ means the concatenation of $K^{(0)}$ with the set of strings in $LBr_{G^*_P}$.

Theorem 5.1. *LBr_P is a regular language over the vocabulary of all commands for each program P.*

Proof. Using (5.2) it is sufficient to show that $LBr_{G^*_P}$ is a regular event, but this is an immediate consequence of Theorem 3.4 of [2], because if x is the input vertex of G_P^*, X is the set of all output vertices in G_P^* and $G_P^*[x, X]$ denotes the set of all labelled branches generated by all monotonic pathes $(v_0, v_1, \ldots, v_q)$ such that $v_0 = x$ and $v_q \in X$, then obviously $G_P^*[x, X] = LBr_{G^*_P}$, and Theorem 3.4 of [2] says that $G_P^*[x, X]$ is a regular event. See also [5].

6. FLOW NETS

With respect to the following algebraic investigations it is convenient to modify the concept of the flow diagram in an unessential but very useful way which consists in omitting the input vertex. In order to avoid any confusion the new term "flow net" will be used for this modified flow diagram.

A flow net without origin is an oriented graph $G = \langle V, \varrho, \lambda, \Lambda \rangle$ with two labellings λ and Λ such that the following requirements (being similar to (4·2–4·4)) are satisfied:

(6.1) (i) there exists at least one output vertex;
(ii) there exists at least one vertex, called a possible origin, such that each vertex belongs at least to one monotonic path which starts at the possible origin and terminates at an output vertex;
(iii) each cycle contains at least one sequential vertex, i.e., a vertex w such that $\mathrm{od}(w) = 1$;

(6.2) (i) w is an output vertex $\Leftrightarrow \lambda(w) = \boldsymbol{STOP}$;
(ii) w is a sequential vetex $\Leftrightarrow \lambda(w)$ is an operational command;
(iii) w is a decision vertex and $\mathrm{od}(w) = k \Leftrightarrow \lambda(w)$ is a decision command in the k-valued logic;

(6.3) $\equiv$ (4.4)

If a possible origin $v \in V$ is distinguished then $\langle V, \varrho, v, \lambda, \Lambda\rangle$ is called a flow net with the origin v. Obviously the origin should replace the input vertex of the flow diagram.

Lemma 6.1. *There is a one-to-one correspondence between flow diagrams G and those flow nets G^* with origin which arise from G by omitting the input vertex and the unique edge starting at it. The vertex at which the omitted edge terminates is the chosen origin. In addition $Br_G = [\boldsymbol{START}, 1]\, BR_{G^*}$, where Br_G and Br_{G^*} are considered as sets of strings.*

The proof follows from the definitions.

It should be mentioned that it is not possible to omit also the output vertices and to distinguish the last vertices in general, because a vertex which is connected by an edge with an output vertex may be a decision vertex, and therefore it can happen that sometimes it should be the last vertex, but sometimes not.

Lemma 6.2. *If $G = \langle V, \varrho, \lambda, \Lambda\rangle$ is a flow net and $x \in V$, then its subgraph $G_x = \langle V_x, \varrho_x, \lambda_x, \Lambda_x\rangle$ defined by the requirement* (6.4) *is a flow net again with the possible origin x where*:

(6.4) *let V_x, ϱ_x be the set of all vertices and edges, respectively, which belong at least to one monotonic path in G which starts at x and terminates at an output vertex of G; $\lambda_x =_{\mathrm{df}} \lambda|_{V_x}$ and $\Lambda_x =_{\mathrm{df}} \Lambda|_{\varrho_x}$.*

The proof is obvious.

Let us define a binary relation $\succ$ in the set of all nets G_x where $x \in V$ and $G = \langle V, \varrho, v, \lambda, \Lambda\rangle$ is a flow net, and simultaneously in the set V itself, as follows:

(6.5) $\quad G_x \succ G_y \Leftrightarrow_{\mathrm{df}} y \in V_x \quad$ and $\quad G_x \sim G_y \Leftrightarrow_{\mathrm{df}} (G_x \succ G_y)\,\&\,(G_y \succ G_x)$.

Theorem 6.3. *The binary relation* $\succ$ *is a quasi-ordering relation, i.e., it is reflexive and transitive, and therefore the binary relation* $\sim$ *is an equivalence relation.*

Proof. According to (6·4) $x \in V_x$ and therefore by (6·5) $G_x \succ G_x$ for each $x \in V$, which proves the reflexivity of $\succ$. Further if $G_x \succ G_y$ and $G_y \succ G_z$ then by (6.5) $y \in V_x$ and $z \in V_y$. Using (6.4) one sees that $y \in V_x$ implies $V_y \subset V_x$, and $z \in V_y$ implies $V_z \subset V_y$. Therefore $V_z \subset V_x$ is true and $z \in V_x$ which means by (6.5) that $G_x \succ G_z$, i.e., the transitivity of $\succ$ is proved. The equivalence of $\sim$ follows by well known theorems.

Let $\overline{V} = \{V_1, V_2, \ldots, V_M\}$ be the set of all equivalence classes defined in V with respect to the equivalence relation $\sim$. Then $\overline{V}$ is a decomposition of V, i.e., the following requirements are satisfied:

(6.6) $V_i \neq \emptyset$; $V_i \cap V_j = \emptyset$ where $i \neq j$ for all $i, j = 1, 2, \ldots, M$; $M \geqq 1$ and $\bigcup_{i=1}^{M} V_i = V$.

Further let $\bar{G} = \langle \overline{V}, \bar{\varrho} \rangle$ be the factor graph of G defined as follows:

(6.7) $(V_i, V_j) \in \bar{\varrho} \Leftrightarrow_{\mathrm{df}}$ there are $v_i \in V_i$ and $v_j \in V_j$ such that $(v_i, v_j) \in \varrho$ for all $i, j = 1, 2, \ldots, M$,

and finally let $G_i = \langle V_i, \varrho_i \rangle$ be defined for $i = 1, 2, \ldots, M$ as follows:

(6.8) $\varrho_i = \varrho \cap V_i^2$.

Lemma 6.4. *If* $G = \langle V, \varrho \rangle$ *is a finite oriented graph having at least two vertices which satisfies* (6.1), *then its factor graph* $\bar{G}$ *is a finite oriented graph having at least two vertices such that the following requirements are satisfied*:

(6.9) (i) *there exists exactly one input vertex in* $\bar{G}$;
(ii) *there are no cycles in* $\bar{G}$;
(iii) *there are no slings in* $\bar{G}$;
(iv) $\bar{G}$ *is connected*;

and each of the subgraphs G_i *is a finite oriented graph such that*:

(6.10) (i) G_i *is strongly connected*;
(ii) *each cycle in* G_i *contains at least one vertex* w *such that* $\mathrm{od}(w) = 1$;
(iii) *there are no slings in* G_i;
(iv) *if* V_i *contains an output vertex then* $|V_i| = 1$.

Proof. First of all let us prove (6.10). (i) follows directly by (6.5); (ii) and (iii) follow by the definition (6.8) and by the assumptions (6.1) concerning G. If V_i contains an output vertex then (iv) follows by (i).

Now let us prove (6.9). By (6.1 ii) it follows that in $\bar{G}$ there exists exactly one input vertex, i.e., (6.9 i) is satisfied, and for this vertex (6.9 iv) is also satisfied. According to (6.5) each two vertices of G which belong to the same cycle must be equivalent, which proves that $\bar{G}$ defined by (6.7) cannot contain any cycle, i.e., (6.9 ii) is satisfied. In G there are no slings, because this is excluded by (6·1 iii) for the decision vertices, and by (ii) for the sequential ones. Therefore (6.7) cannot cause a sling, which means that (6.9 iii) is satisfied.

Finally, $M \geqq 2$ follows by (6.10 iv) and (6.9 ii and iii).

Construction 6.5. *An arbitrary flow net without origin* $G = \langle V, \varrho, \lambda, \Lambda \rangle$ *may be constructed as follows*:

a) *one takes as the base an arbitrary finite oriented graph* $H = \langle W, \sigma \rangle$ *which has at least two vetrices and satisfies* (6.9);

b) *if* $|W| = M$ *and* S, $1 \leqq S < M$ *is the number of the output vertices of* H, *then one takes* M *finite oriented graphs* $G_i = \langle V_i, \varrho_i \rangle$, $1 \leqq i \leqq M$ *such that* (6.6) *is satisfied, each of them satisfies* (6·10 i-iii), *and, in addition, there are at least* S *of them such that* $|V_i| = 1$;

c) *one chooses a one-to-one mapping* φ *such that Domain* $\varphi = \mathrm{W}$ *and Range* $\varphi = \{G_1, \ldots, G_m\}$, *and if* $w \in W$ *is an output vertex in* $\bar{G}$ *then* $G_i = \varphi(w)$ *satisfies the requirement* $|V_i| = 1$;

d) *now one defines* $V = \bigcup_{i=1}^{M} V_i$ *and besides the edges from* $\bigcup_{i=1}^{M} \varrho_i$ *which must belong to* ϱ *there are the following further edges which should belong to* ϱ *as well*; *for each pair* $(w, w^*) \in \sigma$ *one takes* $\varphi(w) = G_i$, $\varphi(w^*) = G_j$ *and then one finds a pair* v_i, v_j *such that* $v_i \in V_i$, $v_j \in V_j$ *and the vertex* v_i *is not the only sequential vertex on a cycle in* G_i; *now one puts* $(v_i, v_j) \in \varrho$; *after defining* ϱ *in this way one may, but need not, stop. i.e., one may go on adding further edges in the following way: if* $(v_i, v_j) \in \varrho$ *then one may take an arbitrary vertex* $v_i^* \in V_i$ *and* $v_j^* \in V_j$ *such that* v_i^* *is not the only sequential vertex on a cycle in* $\langle V, \varrho \rangle$, *and defines* $\varrho^* = \varrho \cup \{(v_i^*, v_j^*)\}$; *this construction may be repeated until the requirement* $(v_i^*, v_j^*) \notin \varrho$ *is satisfied*;

e) *the two labellings* λ *and* Λ *are chosen arbitrarily but they must satisfy* (6.2) *and* (6.3), *respectively.*

Proof of correctness. We need to prove that the graph $G = \langle V, \varrho \rangle$ determined by a)–d) satisfies (6·1), because the other part of the assertion that each graph of this type may be constructed in this way, follows immediately by Lemma 6.4.

First of all it follows from (6.9) that there must exist at least one output vertex (because the graph H is finite, e.g. by Theorem 2.1 of [3]), and therefore $S \geqq 1$ and

after the choice in a) the choice in b) may be done; then also the choice in c) is possible and it remains to consider the determination of ϱ.

If G_i is an arbitrary graph which satisfies (6.10) then either $|V_i| = 1$, which means that v_i such that $V_i = \{v_i\}$ can not be the only sequential vertex which belongs to a cycle in G_i, and therefore v_i may (and must) be always taken in the step d), or $|V_i| > 1$ and the following possibilities must be distinguished: if all the vertices in V_i are sequential, then there always exists the required vertex because $|V_i| > 1$; if there exists at least one decision vertex in G_i then such a vertex may be taken as v_i in any case. Thus we have proved that also the step d) may be done and, if necessary, may be repeated many times.

At last let us prove that $G = \langle V, \varrho \rangle$ satisfies (6.1):

(i) in H there exists at least one output vertex $w \in W$ and by c) one obtains $\varphi(w) = G_i$ such that $|V_i| = 1$. Therefore from d), where ϱ is determined definitely, it follows that $v_i \in V$, where $V_i = \{v_i\}$ must be an output vertex in G;

(*ii*) in H there exists (by (6.9 i)) exactly one input vertex $w \in W$ and there exists (by (6.9)) $v \in V_i = \varphi(w)$; therefore, one immediately sees that v is a possible origin in G, because if v^* is an arbitrary other vertex in G then $v^* \in V_j = \varphi(w^*)$ for some $w^* \in W$, and therefore there is a monotonic path which starts at w, contains w^* and terminates at an output vertex w^{**} of H (see Theorem 2.1 [3]). With respect to the requirement (6·10 i) and to the construction step d) it is clear that there exists a monotonic path in G which starts at v, contains v^*, and terminates at an output vertex of G;

(iii) follows immediately by (6.10 ii) and by the construction step d).

References

[1] *Čulík K.:* Classifications of programming theories and languages, Information Processing Machines 17 (in print) .

[2] *Čulík K.:* Some notes on finite state languages and events represented by finite automata using labelled graphs, Čas. pro pěst. mat. 86 (1961), 43—55.

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

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

[5] *Engeler E.:* Algorithmic Approximations, Journal of Computer and System Sciences 5 (1971), 67—82.

Souhrn

SYNTAKTICKÉ DEFINICE PROGRAMU A BLOKOVÉHO DIAGRAMU

KAREL ČULÍK

Program je definován syntakticky jako uspořádaná, konečná množina značkovaných příkazů, což jsou jisté řetězy nad konečnou abecedou. Značkovanou větví programu se nazývá konečná posloupnost jeho značkovaných příkazů, která udává možné pořadí příkazů v nějakém dokončeném výpočtu. V silné definici programu je připojena řada syntaktických požadavků, motivovaných výpočetním procesem. Blokový diagram programu se zavádí jako orientovaný graf s ohodnocenými uzly i hranami a předkládá se algoritmus syntézy programu k danému blokovému diagramu. Neznačkovaná a operační větev se zavádí pro programy i blokové diagramy a uvádí se nutné a postačující podmínky, kdy dva programy mají tutéž množinu všech značkovaných nebo neznačkovaných větví, která je vždy regulární událostí. Přehled o všech možných blokových diagramech je získán algebraicky pomocí grafové faktorizace, kde faktor — graf je souvislý a acyklický graf s jediným vstupním uzlem, zatím co příslušné podgrafy jsou silně souvislé.

Author's address: Prof. Dr. *Karel Čulík*, Dr.Sc., Výzkumný ústav matematických strojů, Lužná 9, 160 00 Praha 6 - Vokovice.