BULLETIN DE L'ACADÉMIE
POLONAISE DES SCIENCES
Série des sciences techniques
Volume IX, No. 5 — 1961

Some Remarks on Automatic Programming of Arithmetical Formulae

by

Z. PAWLAK

Presented by P. SZULKIN on February 20, 1961

Each digital computer realizes certain formal system. This system we will call "internal language" of the machine or briefly "language of the machine". Languages of digital computers existing differ pretty much from those being in use in computations executed by means of "paper and pensil". Reasons of such a state are obvious. Namely, structure of the language of machine being dependent on the technical possibilities of its realization, whereas the current language was fitted to making of use of the latter in manual computations with ease.

In order to avoid a troublesome use of the language of the machine, for some time already an automatic programming has been applied, which is nothing else but only an interpretation of current language (mathematical language) in the language of the machine.

Automatic programming in a sense of the above conception has several imperfections, namely, it requires a lot of commands, which in consequence lower pretty much the effective speed of operation of the machine. In other words the machine is applied in order to perform activities, to execution of which the construction of the machine is not fitted. Thus the question arises: cannot the problem of "easiness" of the machine service be realized in another way? It seems that there exist two possibilities to solve this problem.

The first is the working out of such mathematical machines, the language of which would be better suited to direct manipulation. Of course, it would be most convenient to construct machines working immediately in the mathematical language being commonly in use. Such a construction is quite possible, however from the technical point of view it would be so inadequate that there are slight chances for building these machines on a large scale.

The idea of working out such mathematical language, which would be both easy in manipulation and also facilitate the construction of a machine working in this language seems to be reasonable.

Certain, elementary steps have already been taken in Poland in this domain (see [1], [2]).

As far as I know this question has not been "touched" abroad up to date.

The second way to solve the problem set in this paper necessitates working out such a language, which would be simply interpreted in the language of the machine and also easy in use.

As a whole the problem is rather difficult.

The present paper gives a certain proposal of such a language for a narrow class of functions, called here arithmetical functions.

1. Determination of parenthesis-free symbolism

In this section of the paper we will discuss a certain parenthesis-free symbolism convenient for recording arithmetical functions in reference to mathematical machines.

As arithmetical functions we will consider functions defined by dyadic output functions: addition, subtraction multiplication, division and the rule of substitution.

We will call the "tree" the set D of dyadic operations $\delta_1, \delta_2, ..., \delta_n$ and set N, of $2n+1$ numbers $a_1, a_2, ..., a_{2n+1}$, such as:

i. For each operation δ_i, which belongs to D, there exists an ordered set of numbers, $a_{i_1}, a_{i_2}, a_{i_3}$, belonging to set N; we will call a_{i_1} the left argument of operation δ_i, and then denote $L(\delta_i)$; a_{i_2} will be called the right argument of operation δ_i and denoted by $R(\delta_i)$; a_{i_3} we will call the result of operation δ_i and denote $0(\delta_i)$.

ii. For each number a_i, which belongs to N, there exists exactly one such an arithmetical operation δ_j, which belongs to D, that $a_i = L(\delta_j)$ or $a_i = R(\delta_j)$ or $a_i = 0(\delta_j)$.

iii. There exists exactly one such a number a_i, which belongs to N, that for no one δ_j, which belongs to D, takes place neither $a_i = L(\delta_j)$ nor $a_i = R(\delta_j)$. This number we will call final result.

Each result of operation we denote by asterisk (*). An argument, which is simultaneously the result of operation is the dependent argument; other arguments are independent ones.

To each element δ_i of the set D we associate the number $A(\delta_i)$, and to each element of the set N we associate the number $A(a_i)$.

The following rule is assumed to carrying out the numeration of elements of the set N:

i. If a_i is the final result, $A(a_i) = 1$,

ii. $A(L(\delta_i)) = 2A(0(\delta_i))+1$,

iii. $A(R(\delta_i)) = 2A(0(\delta_i))$.

The operations will be numerated as follows:

$$A(\delta_i) = A(0(\delta_i)).$$

If $A(a_i) > A(a_j)$, then we say that a_j is a successor of a_j, and we write $a_i \succ a_j$.

If $A(\delta_i) > A(\delta_j)$, we say that δ_i is "later" than δ_j, and we write $\delta_i \succ \delta_j$.

The sequence $\delta_n\ a_{2n+1}\ a_{2n}\ \delta_{n-1}\ a_{2n-1}\ a_{2n-2}, \ldots, \delta_1\ a_3\ a_2\ a_1$ of all elements of the tree we call a formula, if for each $i \leqslant 2n+1$, $a_{i+1} \succ a_i$ and $\delta_{i+1} \succ \delta_i$. Namely, the formulae of the trees, presented in Figs. 1 and 2 have the following form:

$$- cd \cdot b : \cdot ef - a : : g + ::: \text{ and } \cdot cd + ab - : e : ::: .$$

We may easily notice that in the formalism given, the result of i-th operation is i-th dependent argument. This feature enables to locate in a very simple way the results of operations into the memory of the machine.

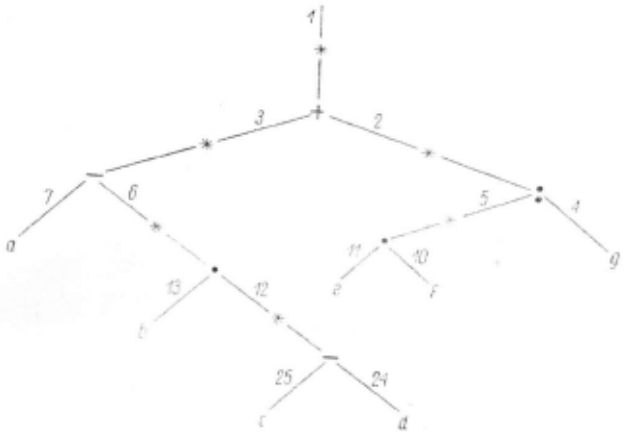

Fig. 1

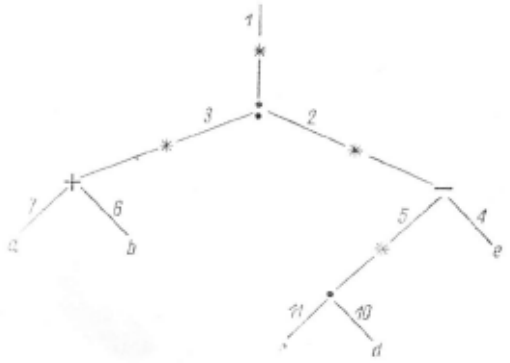

Fig. 2

The formalism under consideration is very comprehensive and, moreover, from the point of practical application to mathematical machines has two qualities:

1) the operations are performed in the order of their appearance in the formula;

2) as has just been mentioned, it allows to locate without any "hindrances" the results of operations.

In the further section of the paper, a simplified programme of one-address machine, interpreting the formalism discussed will be given.

2. Interpretation of parenthesis-free symbolism in one-address machine

Assume one-address machine with the list of commands:

1. *Ra* denotes transfer of the content of address *a* to accumulator,
2. *Aa* — addition of the content of address *a* to accumulator,
3. *Sa* — subtraction of the content of address *a* to accumulator,
4. *Ma* — multiplication of the content of address *a* by the content of the accumulator,
5. *Da* — division of the content of address *a* by the content of the accumulator,
6. *Ta* — transfer of the content of accumulator to the memory *a*.

Expression (a) denotes the address, the address of which is placed at the address of a.

For instance, $T(a)$ denotes the transfer of the number from the accumulator into the memory to the address given at the address *a*.

If a is an address, then a' is the address modified by 1.

The programme brings about the "taking" consecutively: respective arguments, performing of operations and transfer of the results of operations at the appropriate locations in the memory in the following way:

If a letter is the argument of the operation performed in the formula, the number from the address corresponding to the given letter is taken.

If an asterisk (*) is the argument, then the number from address (i) is taken, and after performing of command address (i) is modified.

On performing of each operation, the command $T(j)'$ is carried out, causing the transfer of successive partial results into the memory. For instance, the aforementioned examples will have the following programmes:

$$Re,\ Sd,\ T(j)',\ Rb,\ M(i)',\ T(j)',\ Re,\ Mf,\ T(j)',\ Ra,\ S(i)',\ T(j)',$$

$$R(i)',\ Dg,\ T(j)',\ R(i)',\ A(i)',\ T(j)',$$

$$Rc,\ Md,\ T(j)',\ Ra,\ Ab,\ T(j)',\ R(i)',\ Se,\ T(j)',\ R(i)',\ D(i)',\ T(j)'.$$

The initial value of i and j is the same. Of course, the programme given may be somewhat simplified or modified, in relation to the destination in detail.

The rule and the examples given in the paper are an illustration of the general method of realization of arithmetical formulae, and first of all show how to locate partial results.

The programmes for a three-address machine would be simpler, of course.

INSTITUTE OF MATHEMATICS, POLISH ACADEMY OF SCIENCES
(INSTYTUT MATEMATYCZNY, PAN)

REFERENCES

[1] Z. Pawlak, *Organization of the address-free digital computer for calculating simple arithmetical expressions*, Bull. Acad. Polon. Sci., Sér. sci. techn., **8** (1960), 193.

[2] — , *Organization of address-free computer B-100*, ibid., **9** (1961), 229.