BULLETIN DE L'ACADÉMIE
POLONAISE DES SCIENCES
Série des sciences techniques
Volume IX, No. 8 — 1961

Realization of Memory of Partial Results in Certain Parenthesis-free Formalisms

by

Z. PAWLAK

Presented by P. SZULKIN on February 25, 1961

The aim of this paper is to compare certain two parenthesis-free formalisms from the standpoint of their particular usefulness for mathematical machines. As criterion the way of localization of partial results in the memory was assumed. Obviously, this criterion does not exhaust all characteristics of formalisms, which decide of the usefulness of the latter to realization in mathematical machines. The assumed characterization seems, however, to have some practical advantages and the investigation of formal systems from this point of view may be justified.

1. Definition of languages L_1 and L_2

As arithmetical functions we will consider in this paper functions defined by two-argument initial functions, namely addition, subtraction, multiplication, division and, moreover the rule of substitution.

We shall call the tree T set $\delta(T)$, of symbols of dyadic operations $\delta_1, \delta_2, \ldots, \delta_n$ and the set $\alpha(T)$ of letters $a_1, a_2, \ldots, a_{2n+1}$ of a fixed alphabet A, such that:

i. For each $\delta_i \in \delta(T)$ there exist such $a_{i_1}, a_{i_2}, a_{i_3} \in \alpha(T)$ called left argument of operation δ_i, right argument of operation δ_i and result or output of operation, and denoted by $a_{i_1} = L(\delta_i)$, $a_{i_2} = R(\delta_i)$, $a_{i_3} = O(\delta_i)$ respectively.

ii. To each letter $a_i \in \alpha(T)$ corresponds at least one symbol of operation $\delta_j \in \delta(T)$, such that $a_i = L(\delta_j)$ or $a_i = R(\delta_j)$ or $a_i = O(\delta_j)$.

iii. There exists exactly one such $\delta_i \in \delta(T)$, that for none $a_j \in \alpha(T)$ does not hold: $a_j = L(\delta_i)$ and $a_j = R(\delta_i)$. a_j will be called either final result or output of the tree T, δ_i being called final operation of the tree T.

If $a_i = O(\delta_j)$, then we call a_i a distinguished symbol of alphabet A, and will be denoted by asterisk.

If $a = O(\delta_j)$ and $a = L(\delta_i)$, then we write $\delta_j \rightarrow \delta_i$.

If $a = O(\delta_j)$ and $a = R(\delta_i)$, then we write $\delta_j \Rightarrow \delta_i$.

The tree T' will be called "subtree" of the tree T:

i. if $\delta_i \in \delta(T')$, then $\delta_i \in \delta(T)$,

ii. if $a_i \in a(T')$, then $a_i \in a(T)$,

iii. if a_{i_1}, a_{i_2}, a_{i_3} are the left and right arguments and the results of operation δ_i in the tree T', then a_{i_1}, a_{i_2}, a_{i_3} are also the left and right arguments and the result of operation δ_i in the tree T.

If T' is the subtree of the tree T, then $T—T'$ is also a tree.

If T' is a subtree of the tree T, and δ_i — a final operation of the tree T', then we denote T' by $T(\delta_i)$.

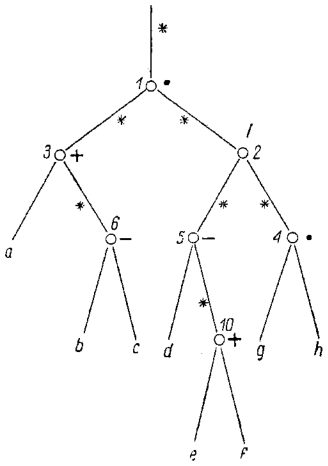
Let us define two systems of numerations N_1 and N_2, of operations of the tree.

Numeration N_1. To each symbol of operation δ_i we associate number $N_1(\delta_i)$, according to the following rule:

i. if δ_i is a final operation of the tree T, then $N_1(\delta_i) = 1$:

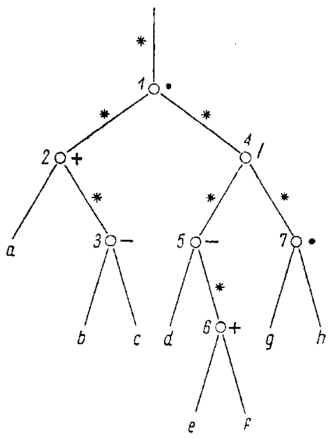
ii. if $\delta_i \Rightarrow \delta_j$, then $N_1(\delta_i) = 2N_1(\delta_j)$,

iii. if $\delta_i \rightarrow \delta_j$, then $N_1(\delta_i) = 2N_1(\delta_j)+1$.



$+ef — bc — d * \cdot gh + a */** \cdot ***$

Fig. 1. Language L_1



$\cdot gh + ef — d */** — bc + a* \cdot ***$

Fig. 2. Language L_2

Numeration N_2. This numeration will be also called the Łukasiewicz numeration:

i. if δ_i is a final operation of the tree T, then $N_2(\delta_i) = 1$.

ii. if $\delta_i \rightarrow \delta_j$, then $N_2(\delta_i) = N_2(\delta_j)+1$,

iii. if $\delta_i \Rightarrow \delta_j$, then $N_2(\delta_i) = \text{Max}\,(T(\delta_j) — T(\delta_i))+1$ *).

If $N_1(\delta_i) > N_1(\delta_j)$, then we shall write $\delta_i \succ \delta_j$.

If $N_2(\delta_i) > N_2(\delta_j)$, then we shall write $\delta_i \succ\succ \delta_j$. Sequence $\delta_n\, a_{2n+1}\, a_{2n}\, \delta_{n-1}\, a_{2n-1}\, a_{2n-2} \ldots \delta_1\, a_3\, a_2\, a_1$ will be called a formula in L_1, if for each $i \leqslant n$, $\delta_{i+1} \succ \delta_i$ and

*) Max (T) denotes the greatest number $N(\delta_i)$ of operation δ_i in the tree T.

a_{2i+1}, a_{2i} are appropriately the left or right argument of operation δ_i. Sequence $\delta_n\ a_{2n+1}\ a_{2n}\ \delta_{n-1}\ a_{2n-1}\ a_{2n-2} \dots \delta_1\ a_3\ a_2\ a_1$ will be called formula in L_2, if for each $i < n$, $\delta_{i+1} \succ\succ \delta_i$ and a_{2i+1}, a_{2i} are left and right arguments of δ_i.

The language L_2 may be considered as a certain modification of the known parenthesis-free Łukasiewicz's symbolism. Simple examples of trees and formulae in both these languages are given in Figs. 1 and 2.

It can be shown that for computation of an arbitrary arithmetic formula, containing n dyadic operations, written down in the language L_1, at least $E(n+1/2)$ locations in the memory for partial results are required, whereas computation of the same formula written down in the langugae L_2 needs at least $\ln_2(n+1)$ of memory locations. E denotes the whole part.

2. Realization of the memory of partial results by means of languages L_1, L_2

The memory of partial results may be considered as a generalization of an accumulator in a one-address machine. In both these languages, partial results may be automatically located into the memory of partial results and also taken up in the same way in the course of computing of the formula by means of the machine — setting of addresses, by a programmer is not needed here.

We may quote here several possibilities at design of the memory of partial results in both aforesaid languages.

In this paper we will discuss only these possibilities which seem to have certain practical applications. The choice of a proper organization of the memory depends, of course, not only upon the formalism used but also is determined by the kind of the memory (e.g. ferrite, drum or delay-line-memory).

In the language L_1 partial results can be placed into consecutive positions in the memory in the course of computing, and they are taken up in the same sequence, in order to perform computing. This property enables numerous, different solutions of the memory of partial results. Some of them are discussed in [1]—[3]. The solutions given require n locations in the memory of partial results, provided there are not more than $E(n+1/2)$ partial results. Thus half of the memory has not been utilized.

If the machine contains a great, and quick memory, ferrite memory for instance, where the whole formula is located in the course of computing, the utilization of the entire memory is not so important and a certain insignificant loss associated with it may well be disregarded.

Similarly, if the machine has only a drum memory, usually with high capacity, then a certain disregard of the memory for partial results is of no importance.

In a case when we deal with a machine having a drum memory and a small quick memory of partial results the proper capacity of the quick memory is of primary importance; in a case when the language L_1 is used it may amount to $E(n+1/2)$. The memory of partial results may then be built similarly to the shifting register of numbers under the condition that now the delay-lines will store not only one bit, as is done in the register, but the whole number (partial result).

In the language L_2 the least possible number of locations for partial results amounts to $\ln_2 (n+1)$. Then, the algorithm of localization of partial results is as follows:

i. Insert the first partial result in the location No.*i*.

ii. If not one of the arguments of operation executed is a partial result, then the result of operation should be put in the successive location.

iii. If one of the arguments of operation is a partial result, then take as its value the last number written in the partial results memory, and write the result of operation in the place of the number just taken.

iv. If both the arguments of operation are partial results as their values, then take two last numbers written in the partial results memory and write the result in the place of the last but one number in the memory of partial results; erase the last number.

The algorithm under consideration may be used in any memory. It is particularly advantageous for small machines with drum memory and with small quick memory for partial results. Realization of the memory of partial results by the algorithm given is not difficult.

Broadly speaking, the Łukasiewicz symbolism modified is more convenient in practical application for mathematical machines than the symbolism L_1, for it requires a smaller memory of partial results than does the latter, and it is also easier in use than the symbolism L_1. On the contrary, in less complex machines, namely, these similar in construction to B—100 (see [3]) the language L_1 is simpler in technical realization than the previous one.

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 with separate memory of partial results*, ibid., **9** (1961), 123.

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