SovietRxiv

Soviet-era science, translated into English

A. P. ERSHOV

1958

SovietRxiv

View the original and related papers at https://sovietrxiv.org/items/ru-195801.97783

Source: Math-Net.Ru and CyberLeninka. Machine translation. Verify with the original.

SovietRxiv

Abstract

Full Text

A. P. ERSHOV

ON OPERATOR ALGORITHMS

(Presented by Academician S. L. Sobolev on 7 VI 1958)

The article proposes a new method of formally describing algorithms, on the basis of which, in the author' s opinion, one can develop a mathematical apparatus for solving a number of problems in theoretical programming. In connection with this, a definition of an "operator algorithm" is given, and the connection of operator algorithms with partial recursive functions and normal algorithms is established.

1°. Preliminary constructions. Each concrete class of operator algorithms is constructed starting from a certain potentially infinite set V of **variables** and a certain list S of **operations**. A variable is represented by a word in some alphabet. For each variable $x \in V$, all those constructive objects (c.o.) which may be values of the variable x are described. Each k-ary ($k > 0$) operation is represented by a word $A_0(\ )A_1(\ )\dots(\ )A_k$, where $A_0, A_1, \dots, A_k$ are words in some alphabet. The i-th pair of parentheses from the left ( ) is called the i-th **place** of the operation $A_0(\ )A_1(\ )\dots(\ )A_k$. Each k-ary operation specifies, for certain sets of c.o. $\mu_1, \dots, \mu_k$, an effective method of obtaining a new c.o. ν, called the **result of applying the operation to the set** of c.o. $\mu_1, \dots, \mu_k$. The result of applying the operation $A_0(\ )A_1(\ )\dots(\ )A_k$ to the set $\mu_1, \dots, \mu_k$ will be denoted by $A_0(\mu_1)A_1(\mu_2)\dots(\mu_k)A_k$.

Below, by simultaneous induction, the following notions are defined: **expression**, **arguments of an expression**, **value of an expression for given values of the arguments**.

a) **Basis of induction.** The word x (where x is any variable) is an expression. The argument of the expression x is x. For any value x of the argument x, the expression x has a value equal to x.

Let $T(x_1, \dots, x_n)$ denote an expression T with arguments $x_1, \dots, x_n$, and let $T(x_1, \dots, x_n)$ denote the value of the expression T, defined for the values $x_1, \dots, x_n$ of the arguments $x_1, \dots, x_n$.

b) **Induction step.** Let $A_0(\ )\dots(\ )A_k$ be a k-ary operation and $T_1(x_{11}, \dots, x_{1n_1}), \dots, T_k(x_{k1}, \dots, x_{kn_k})$ expressions. Then the word T, obtained by substituting into the i-th place of the operation $A_0(\ )\dots(\ )A_k$ the expression $T_i(x_{i1}, \dots, x_{in_i})$, and having the form

$$A_0(T_1(x_{11}, \dots, x_{1n_1}))\dots(T_k(x_{k1}, \dots, x_{kn_k}))A_k,$$

SovietRxiv

is an expression with arguments $x_1, \dots, x_n$, where $x_1, \dots, x_n$ are all those variables which are arguments of the expressions $T_1, \dots, T_k$.

Let $x_1, \dots, x_n$ be values of the arguments of the expression T. Then, if $T_1(x_{11}, \dots, x_{1n_1}) = y_1, \dots, T_k(x_{k1}, \dots, x_{kn_k}) = y_k$ and $A_0(y_1) \dots (y_k)A_k = y$, then $T(x_1, \dots, x_n) = y$.

A **formula** is any word of the form $T(x_1, \dots, x_n) \Rightarrow y$, where $T(x_1, \dots, x_n)$ is any expression and y is any variable. An **operator** is any word of the form $T_1 \Rightarrow y_1 * \dots * T_m \Rightarrow y_m$, where $m > 0$, $T_i \Rightarrow y_i$ is a formula $(i = 1, \dots, m)$. The alphabets in which variables and operations are written are chosen so that every operator is uniquely decomposed into uniquely readable formulas.

2°. **Operator algorithms.** Let variables V and operations S be given. Fix some c.o. p_0, which we shall call the **stop**. Take arbitrary $r + s$ variables from V. Declare r variables $p_1, \dots, p_r$ to be the **parameters** of the operator algorithm and assign them certain values $p_1, \dots, p_r$. Among the parameters single out one variable p^*, which we shall call the **beginning** of the operator algorithm. Declare the remaining s variables to be the **functional variables** of the operator algorithm. Any collection of functional variables $x_1, \dots, x_s$ and parameters $p_1, \dots, p_r$, with a distinguished beginning p^* and with assigned values $p_1, \dots, p_r$, will be called an **operator algorithm** $\mathfrak{A}$ from the class $\mathfrak{A}(V, S)$ of operator algorithms over the variables V and operations S.

The execution of the algorithm* $\mathfrak{A}$ is preceded by assigning arbitrary values $x_1, \dots, x_s$, which are **initial data** for the algorithm $\mathfrak{A}$, to the functional variables $x_1, \dots, x_s$. After the initial data have been chosen, any distinction between parameters and functional variables is erased.

The execution of the algorithm $\mathfrak{A}$ consists in the successive transformation of its **domain of action**—a certain list of variables to which certain values have been assigned. At the initial moment the domain of action of the algorithm $\mathfrak{A}$ is formed by the variables $p_1, \dots, p_r, x_1, \dots, x_s$, having the values $p_1, \dots, p_r, x_1, \dots, x_s$, respectively. Transformation of the domain of action consists in changing the variables belonging to the domain of action and in adding new variables to it.

The execution of the algorithm $\mathfrak{A}$ is composed of the following elementary acts: α) transition to a variable; β) execution of an operator; γ) natural stop; δ) stop without result.

The initial step in the execution of the algorithm $\mathfrak{A}$ consists in a transition to the beginning p^*. The further execution of the algorithm proceeds cyclically.

Suppose that a transition has occurred to some variable $\tilde{y}$. The value $\tilde{y}$ of this variable is examined. If $\tilde{y}$ is not an operator, a stop without result occurs. Suppose $\tilde{y}$ is an operator equal to $T_1 \Rightarrow y_1 * \dots * T_m \Rightarrow y_m$. In this case, successively for all i $(i = 1, \dots, m)$, computations are carried out according to the i-th formula of the operator. Let $T_i(x_1, \dots, x_n) \Rightarrow y_i$ be the i-th formula of the operator. If not all of the variables $x_1, \dots, x_n$ belong to the domain of action

SovietRxiv

of $\mathfrak{A}$, a stop without result occurs. Suppose $x_1, \dots, x_n$ belong to the domain of action of $\mathfrak{A}$ with values $x_1, \dots, x_n$. If $T_i(x_1, \dots, x_n) = v_i$ and v_i can be the value of the variable y_i, then the value v_i is assigned to the variable y_i, and the variable y_i with this value is included in the domain of action of the algorithm $\mathfrak{A}$**. In the contrary case a stop without result occurs.

After the computation according to the last formula of the operator $\tilde{y}$, the computed value v_m of the variable y_m in the last formula is examined. If $v_m = p_0$, a natural stop occurs, indicating the end of the execution of the algorithm $\mathfrak{A}$. The result of the execution of $\mathfrak{A}$ is considered to be the values of all variables that by this time belong to the domain of action of $\mathfrak{A}$. If v_m is a c.o. that is not a variable belonging to the domain of action of $\mathfrak{A}$, there occurs–

---

* Here and everywhere below, "algorithm" means "operator algorithm."

** If y previously belonged to the domain of action of $\mathfrak{A}$ with value y, then simply y is replaced by v. At the same time, if it turns out that $y = \tilde{y}$, i.e., that y is precisely the operator which at the given moment is being executed, then it is considered that the replacement of y by v occurs after the computation according to all formulas entering into the operator $\tilde{y}$.

entails an unproductive stop. If, however, $v_m = \tilde{y}$ and $\tilde{y}$ lies in the domain of action $\mathfrak{A}$, then there is a transition to the variable y, etc.

3^0. **Some classes of operator algorithms.**

I. **The class $\mathfrak{A}(V_1, S_1)$.** The variables from V_1 are the notations of all natural numbers in the decimal system, except for the number 2, which denotes the stop n_0. All even variables x ($x \neq 0$) are called **numerical variables**, whose values are natural numbers. All odd variables and the variable 0 are called **operator variables**, whose values are words in the alphabet
$\{0, \dots, 9, +, \mathrm{P}, (,), *, \Rightarrow\}$. Let μ, μ_1, μ_2 be arbitrary natural numbers.

Operations S_1:
1) the operation $(\ ) + 1$, for which $(\mu) + 1 = \mu + 1$;
2) the operation $\mathrm{P}((\ )(\ ))$, for which

$$\mathrm{P}((\mu_1)(\mu_2)) = \begin{cases} 1, & \text{if } \mu_1 \leqslant \mu_2; \\ 0, & \text{if } \mu_1 > \mu_2. \end{cases}$$

II. **The class $\mathfrak{A}(V_2, S_2)$.** The variables are all words in the alphabet $\{|\}$, except for the word $||$, which denotes the stop n_0. The values of the variables are words in some alphabet $A = \{a_1, \dots, a_t\}$, which includes the letters $\mathrm{T}, \mathrm{R}, \mathrm{S}, |, (,), *, \Rightarrow$.*

Let P be any word in the alphabet A.

SovietRxiv

Operations S_2:
1) the operation T( ), for which

$$\mathrm{T}(P) = \begin{cases} P', & \text{if } P = \xi P' \ (\xi \in A); \\ \Lambda, & \text{if } P = \Lambda; \end{cases}$$

2) the operations $\mathrm{R}|^i(\ )$ $(i = 1, \dots, t)$, for which $\mathrm{R}|^i(P) = a_{iP}$;

3) the operations $\mathrm{S}|^i(\ )$, for which

$$\mathrm{S}|^i(P) = \begin{cases} |, & \text{if } P = a'_{iP}; \\ \Lambda, & \text{if } P \neq a'_{iP}. \end{cases}$$

III. **The class $\mathfrak{A}(V_3, S_3)$.** $V_3 = V_2$. Let P_1, P_2 be arbitrary words in the alphabet A.

Operations S_3:
1) the operation R(( )( )), for which $\mathrm{R}((P_1)(P_2)) = P_1P_2$;
2) the operation S(( )( )), for which

$$\mathrm{S}((P_1)(P_2)) = \begin{cases} |, & \text{if } P_2 \text{ ends with the word } P_1; \\ \Lambda, & \text{if } P_2 \text{ does not end with the word } P_1. \end{cases}$$

Suppose an algorithm $\mathfrak{A}$ from the class $\mathfrak{A}(V, S)$ is given. To each variable $y \in V$, with the help of the algorithm $\mathfrak{A}$, we assign a certain function $y_{\mathfrak{A}}(x_1, \dots, x_s)$, whose arguments are the initial data $x_1, \dots, x_s$ of the algorithm $\mathfrak{A}$, and which is computed according to the following rule: suppose that the execution of $\mathfrak{A}$ for the initial data $x_1, \dots, x_s$ terminates with a natural stop and that, at the moment when the execution of $\mathfrak{A}$ ends, the variable y lies in the domain of action $\mathfrak{A}$ with value y. Then $y_{\mathfrak{A}}(x_1, \dots, x_s) = y$. We shall say that the function $y_{\mathfrak{A}}(x_1, \dots, x_s)$ is **realized** by the algorithm $\mathfrak{A}$.

* The restriction on the alphabet A is not essential. If A contains at least two letters, all the listed letters used for representing operators can be translated into words in the alphabet A without loss of unambiguity in reading an operator ("(|)" p. 40).

§4. **Operator algorithms and partial-recursive functions** ((2), p. 294).

Theorem 1. For any partial-recursive function p.r.f. $\varphi(x_1, \dots, x_n)$ there exists an operator algorithm of the class $\mathfrak{A}(V_1, S_1)$ with n functional variables, realizing $\varphi(x_1, \dots, x_n)$.

Idea of the proof. Individual algorithms are constructed that realize the basic functions of counting, argument selection, and constants. Then a method is given for constructing, from algorithms realizing certain initial p.r.f.' s, new

SovietRxiv

algorithms realizing p.r.f.' s obtained by applying to the initial p.r.f.' s the substitution operator, the primitive recursion operator, and the μ-operator.

Theorem 2. Any function realized by any operator algorithm of the class $\mathfrak{A}(V_1, S_1)$ is a partial-recursive function.

Idea of the proof. An inductive method is found for constructing an arbitrary algorithm from $\mathfrak{A}(V_1, S_1)$, starting from the simplest algorithms containing only one executed operator, by successive application of the operations of **looping** and **joining**. The operation of joining consists in constructing from the algorithms $\mathfrak{A}^*, \mathfrak{A}_1, \dots, \mathfrak{A}_n$ a new algorithm $\mathfrak{A}$, realizing the instruction: execute the algorithm $\mathfrak{A}^*$, and then one of the algorithms $\mathfrak{A}_1, \dots, \mathfrak{A}_n$ depending on the results obtained. The operation of looping consists in constructing from the algorithm $\mathfrak{A}^*$ a new algorithm $\mathfrak{A}$, realizing the instruction: execute the algorithm $\mathfrak{A}^*$ repeatedly until a natural or resultless stop occurs. It is proved that the simplest algorithms realize only p.r.f.' s. Proceeding from the assumption that the theorem is valid for certain algorithms, it is proved that the algorithms obtained from them by the operations of joining and looping also satisfy the theorem.

§5. **Operator algorithms and normal algorithms** ((1), p. 54).

Theorem 3. For any normal algorithm $\mathfrak{A}_n$ in the alphabet A there exists an operator algorithm $\mathfrak{A}$ of the class $\mathfrak{A}(V_2, S_2)$ and a variable y such that for any word P in the alphabet A the conditional equality* holds

$$\mathfrak{A}_n(P) \simeq y_{\mathfrak{A}}(P).$$

Idea of the proof. For any substitution formula ((1), p. 54)

$$R = P \to \sigma Q \quad (\sigma \text{ is either } \cdot \text{ or } \Lambda)$$

an algorithm $\mathfrak{A}^*$ from $\mathfrak{A}(V_2, S_2)$ is constructed, realizing the application of R. For any normal algorithm $\mathfrak{A}_n$ with substitution formulas $R_1, \dots, R_n$, a method is given for composing the algorithm $\mathfrak{A}$ from algorithms $\mathfrak{A}_1^*, \dots, \mathfrak{A}_n^*$, realizing the application of the substitution formulas $R_1, \dots, R_n$, respectively.

Theorem 4. Theorem 3 is also valid for the class of operator algorithms $\mathfrak{A}(V_3, S_3)$.

Idea of the proof. Individual algorithms of the class $\mathfrak{A}(V_3, S_3)$ are constructed which realize the operations S_2.

Computing Center
Academy of Sciences of the USSR

Received
30 V 1958

SovietRxiv

CITED LITERATURE

1. A. A. Markov, *Tr. Matem. inst. im. V. A. Steklova AN SSSR*, **42** (1954).
   S. C. Kleene, *Introduction to Metamathematics*, Moscow, 1956.

* Concerning conditional equality, see (1), p. 51.

Note: Figure translations are in progress. See original paper for figures.

Source: Math-Net.Ru and CyberLeninka. Machine translation. Verify with the original.