Mathematics

1961

SovietRxiv

Abstract

Full Text

Mathematics

N. M. Nagornyi

On the Realization of Functions in Alphabets by Algorithms of Certain Classes

(Presented by Academician P. S. Novikov on 20 IV 1961)

In the paper ([3]) the question of finding a minimal alphabet of normal algorithms over a given alphabet was considered. It turned out that, whatever the alphabet A: 1) every normal algorithm over A is equivalent relative to A to some normal algorithm in a one-letter extension of A, and 2) one can specify a class of normal algorithms over A that are not equivalent relative to A to any normal algorithms in A.

In the present note an analogous question is considered for algorithms of type σ ([2]) and T-algorithms—algorithms defined by means of Turing machines. The results obtained are connected with the problem of classifying algorithms according to their complexity.

1. Let A be an arbitrary alphabet. By a function in the alphabet A we shall mean a constructively specified partial transformation of the set of words in the alphabet A.

Let f be a function in the alphabet A, and let $\mathfrak{A}$ be an algorithm over A. We shall say that the algorithm $\mathfrak{A}$ realizes the function f if, for every word P in A, the conditional equality ([1], p. 51) $\mathfrak{A}(P) \simeq f(P)$ holds.

1.1. We shall consider systems of words in the alphabet A. Two systems with the same number of terms will be called concordant.

Let P be a word in A, and let $U = (U_1, \dots, U_n)$ and $V = (V_1, \dots, V_n)$ be concordant systems of words in A. Put

$$P_0 = P, \qquad P_{i+1} = \Sigma(P_i, U_{i+1}, V_{i+1}) \quad (i = 0, 1, \dots, n-1)$$

(here $\Sigma(P, Q, R)$ denotes the result of substituting R for the first occurrence of Q in P, if Q occurs in P, and is undefined if Q does not occur in P; see [1], pp. 25, 34). We shall denote P_n by the symbol $\{P, U, V\}$.

Let P be a word in A, and let $U = (U_1, \dots, U_m)$ be a system of words in A. We shall say that P is a word of type U if there exist occurrences $W_1, \dots, W_m$ in the word P such that W_i is an occurrence of the word U_i, and for $i \neq j$, $W_i \neq W_j$.

We shall call a function f in the alphabet A covering if, for every system U of words in A, there exists a word Q in A such that $f(Q)$ is defined and is a word of type U.

Theorem 1. *Let f be a covering function in the alphabet A such that, whatever concordant systems U and V of words in A may be, there exists a word P in A such that $f(P)$ and $\{P, U, V\}$ are defined and $f(P) \neq \{P, U, V\}$. Then f cannot be realized by any algorithm of type σ in the alphabet A.*

The proof is based on reducing algorithms of type σ to a certain canonical form.

From Theorem 1 the following theorems follow:

Theorem 2. *A covering function f in the alphabet A such that, whatever word P in A may be, there exist words Q and R in A such that P occurs in Q, while $f(Q)$ and $f(QR)$ are defined and $f(QR) \neq f(Q)R$, cannot be realized by any algorithm of type σ in the alphabet A.*

Theorem 3. *A covering function f in the alphabet A such that, whatever an integer n and a word P in A may be, there exists a word Q in A such that*

that P occurs in Q, $f(Q)$ is defined, and $f(Q)^{\partial} \neq [Q^{\partial} + n$ ((1), p. 17), cannot be realized by any algorithm of type σ in the alphabet A.

From Theorem 2 it follows that

Theorem 4. If the alphabet A contains more than one letter, then a computing function f in the alphabet A, for any word P in A satisfying the condition $f(P) = [P^{\partial}$ ((1), p. 24), cannot be realized by any algorithm of type σ in the alphabet A.

We shall call a function f in the alphabet A *stretching* if one can indicate a constructive [(4)] real number k, greater than one, such that for every P in A for which $f(P)$ is defined, $[f(P)^{\partial} \geq k[P^{\partial}$.

From Theorem 3 it follows that

Theorem 5. A stretching covering function f in the alphabet A such that, for every word P in A, there is a word Q such that P occurs in Q and $f(Q)$ is defined, cannot be realized by any algorithm of type σ in the alphabet A.

Theorem 6. If $n \geq 2$, then the function f_n in the alphabet A, satisfying the condition

$$f_n(P) = \underbrace{P \ldots P}_{n \text{ times}}(P \text{ is a word in } A),$$

cannot be realized by any algorithm of type σ in the alphabet A.

Theorem 7. A stretching function defined on an infinite set of words in a one-letter alphabet cannot be realized by any algorithm of type σ in this alphabet.

1.2. Theorem 5 is an analogue of Corollary 1 to Theorem 3.1 of work [(3)]. However, the condition requiring that the function f be covering, which is absent in [(3)], is essential here. Namely:

Theorem 8. If the alphabet A contains at least two letters, one can indicate a stretching function in A, defined for every word in this alphabet, that is realized by some algorithm of type σ in A.

Hence it follows that

Theorem 9. If the alphabet A contains more than one letter, then one can construct an algorithm of type σ in A that is not equivalent relative to A to any normal algorithm in A.

It is curious, however, that

Theorem 10. Every algorithm of type σ in a one-letter alphabet Ψ is equivalent relative to Ψ to some normal algorithm in this alphabet.

Algorithms of type σ in a one-letter alphabet also possess other special features. In particular, for them the applicability and equivalence problems are decidable. The class of arithmetical functions generated by these algorithms coincides with the class of arithmetical functions f satisfying the condition

$$\forall x\,(!f(x)\vee\neg!f(x))\,\&\,\exists y\exists z(\forall x(x\geq y\supset f(x)\simeq f(x+z))\vee\forall x(x\geq y\supset f(x)=x+z)\vee(y\geq z\,\&\,\forall x(x\geq y\supset f(x)=$$

(Here $!f(x)$ means "$f(x)$ is defined" ; x, y, z are variables for natural numbers.)

1.3. Comparing Theorem 6 with the result of § 3 of work [2] and Theorem 2.1 of work [3], we obtain:

Theorem 11. The minimal alphabet of algorithms of type σ over the alphabet A is the one-letter extension of A.

Of course, in consequence of V. A. Uspenskii' s theorem [6] on the impossibility of recognizing invariant properties of algorithms, we have:

Theorem 12. There is no algorithm recognizing those algorithms of type σ (or normal algorithms) over the alphabet A for which there exist algorithms of type σ in the alphabet A equivalent to them relative to A.

2. We now pass to consideration of the analogous question for algorif-

···defined by means of Turing machines. As the basis for defining algorithms we shall use Turing machines with extendable finite tapes.

2.1. Let A be an arbitrary alphabet consisting of letters $a_1, \ldots, a_n$. Words in the alphabet A will be written on tapes, which we shall regard as finite and divided into a finite number of cells, in each of which any letter of the alphabet A may be written. Of an empty cell we shall say that the empty sign is written in it. In what follows, the letters of the alphabet A and the empty sign will, for brevity, be called signs.

Let P be a word in the alphabet A. We shall say that the standard record of the word P is printed on a tape L if one of the following two conditions is satisfied:

a) $P = a_{i_1} a_{i_2} \ldots a_{i_k}$ $(k \geqslant 1)$; the tape L consists of k cells, and in the j-th cell from the left the letter a_{i_j} is written;

b) P is the empty word and the tape L consists of one cell, in which the empty sign is written.

We shall say that a record of the word P is printed on the tape if on some segment of the tape the standard record of the word P is printed, while the cells of the tape not belonging to this segment are empty.

2.2. We shall now describe a Turing machine in the alphabet A. A Turing machine in the alphabet A has a finite number of states, among which there are distinguished states: the initial and the final state. A Turing machine is capable of establishing whether a tape is within its field of vision, and, if it is, then it perceives one cell of the tape and reads the sign written in it. The machine is capable of writing a new sign in the cell under consideration in place of the sign under consideration and of shifting the tape by one cell to the right or to the left, or of leaving the tape in place. In necessary cases (to be discussed below), the machine is capable of attaching one empty cell to one of the ends of the tape.

2.3. The operation of a Turing machine consists in the successive execution of elementary steps. It is convenient to regard these steps as occurring at integral moments of time. They may be of two kinds, depending on whether at the given moment the tape is within the machine' s field of vision. A step of the first kind consists in the following: the machine, being at some moment of time in a nonfinal state q and perceiving a sign ξ, at the next moment prints in the cell under consideration the sign η, passes into a new state r, and shifts the tape or leaves it in place, with the new sign η, the new state r, and the movement of the tape being uniquely determined by the preceding state q and the sign ξ. The initial step in the operation of a Turing machine will always be a step of this kind. An elementary step of the second kind is performed by a Turing machine when the preceding step has caused the tape to leave the machine' s field of vision, i.e. when the machine, having printed a sign in the extreme left cell, shifted the tape to the right, or, having printed a sign in the extreme right cell, shifted the tape to the left. An elementary step of the second kind consists in the following: the machine, being at some moment of time in a nonfinal state q and having established that the tape is not within the machine' s field of vision, at the next moment attaches one empty cell to the left end of the tape, if in performing the preceding step the tape shifted to the right, and to the right end if in performing the preceding step the tape shifted to the left, and then, having set this cell in the machine' s field of vision, prints on it the sign η, passes into a new state r, and shifts the tape or leaves it in place, with the sign η, the state r, and the movement of the tape being uniquely determined by the state q. A complete description of all possible elementary steps of the machine can be given by a certain table—the control scheme of the machine. Being in the final state, the Turing machine performs no step. An elementary step after whose

performance the machine passes into the final state will be called final.

2.4. By a T-algorithm in the alphabet A we shall mean any algorithm $\mathfrak{A}$ in A defined as follows. Some Turing machine is fixed···

Turing machine $\mathfrak{M}$ in the alphabet A. Let P be the word accepted as the input datum. We print on the tape the standard notation of P, put the machine $\mathfrak{M}$ into its initial state, and in its field of vision set the leftmost cell of the tape containing the standard notation of P. Then $\mathfrak{M}$ will begin to perform elementary steps in accordance with its control scheme until the final step occurs, after which the operation of the machine stops. If in this process a notation of some word Q in the alphabet A (not necessarily standard) is printed on the tape, then we shall say that the T-algorithm under consideration $\mathfrak{A}$ is applicable to the word P and that the result of the application is the word Q. To express this circumstance we shall also use the notation $\mathfrak{A}(P) = Q$. Every T-algorithm is completely determined by specifying the alphabet in which it acts—the alphabet of this algorithm—and some Turing machine in this alphabet. Of course, such a refinement of the concept of an algorithm in an alphabet is equivalent to other known refinements (for example, by means of the concept of a normal algorithm).

Theorem 13. *Every T-algorithm over an alphabet A is equivalent, relative to A, to some T-algorithm in A.*

Thus, the minimal alphabet of T-algorithms over an alphabet A is the alphabet A itself. The proof of Theorem 13 uses, as is usual in proofs of this kind, the method of coding. Taking into account the special features of Turing machines, it is convenient, as C. E. Shannon did in ([5]), to choose the coding to be uniform.

Computing Center
Academy of Sciences of the USSR

Received
18 IV 1961

CITED LITERATURE

1. A. A. Markov, *Tr. Matem. inst. im. V. A. Steklova AN SSSR*, **42** (1954).

2. N. M. Nagornyi, *Tr. Matem. inst. im. V. A. Steklova AN SSSR*, **52**, 7 (1958).

3. N. M. Nagornyi, *Tr. Matem. inst. im. V. A. Steklova AN SSSR*, **52**, 66 (1958).

4. H. A. Shannon, *Zs. f. math. Logik u. Grundlagen d. Math.*, **2**, 1956, S. 27.

5. C. E. Shannon, *V sborn. Avtomaty*, Moscow, 1956, p. 213.

6. V. A. Uspenskii, *UMN*, **11**, 7 (70), 172 (1956).

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

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