Mathematical Problems of Computer Science 43, 32--41, 2015.

On Strongly Positive Multidimensional Arithmetical Sets[1]

Seda N. Manukian

Institute for Informatics and Automation Problems of NAS RA
e-mail: zaslav@ipia.sci.am

Abstract

The notion of positive arithmetical formula in the signature $(0,=,S)$, where $S(x)=x+1$, is defined and investigated in [1] and [2]. A multidimensional arithmetical set is said to be positive if it is determined by a positive formula. Some subclass of the class of positive sets, namely, the class of strongly positive sets, is considered. It is proved that for any $n \geq 3$ there exists a $2n$-dimensional strongly positive set such that its transitive closure is non-recursive. On the other side, it is noted that the transitive closure of any 2-dimensional strongly positive set is primitive recursive.

Keywords: Arithmetical formula, Transitive closure, Recursive set, Signature.

1. Introduction

The classes of recursive sets having in general non-recursive transitive closures have been investigated in the theory of algorithms since the first steps of this theory ([3]-[8]). The works [9]-[13] are dedicated mainly to algebraic problems, however, some examples of recursive sets having non-recursive transitive closures are actually given also in these works. In [14] it is noted that there exists a two-dimensional arithmetical set belonging to the class Σ_4 and having a non-recursive transitive closure (the classes Σ_n for $n \geq 0$ are defined in [14] as some classes of arithmetical sets determined by formulas in M. Presburger's system ([4], [15], [16])). Below the class of strongly positive arithmetical sets is considered (the definition will be given in Section 2) such that the sets belonging to this class have a more simple structure than the sets noted above, and have the following properties: (1) for any $n \geq 3$ there exists a $2n$-dimensional strongly positive set such that its transitive closure is non-recursive; (2) any 2-dimensional strongly positive set has a primitive recursive transitive closure (see below, Theorem 1 and Theorem 2).

[1] This work was supported by State Committee of Science, MES RA, in frame of the research project №SCL 13-1B321.

2. Main Definitions and Results

By N we denote the set of all non-negative integers, $N = \{0,1,2,...\}$. By N^n we denote the set of n-tuples $(x_1, x_2, ..., x_n)$, where $n \geq 1$, $x_i \in N$ for $1 \leq i \leq n$.

An n-dimensional arithmetical set, where $n \geq 1$, is defined as any subset of N^n.

An n-dimensional arithmetical predicate P is defined as a predicate which is true on some set $A \subseteq N^n$ and false out of it; in this case we say that A is the set of truth for P, and P is the representing predicate for A.

The notions of primitive recursive function, general recursive function, partially recursive function, primitive recursive set, recursive set are defined in a usual way ([3]-[8]). The corresponding terms will be shortly denoted below by PmRF, GRF, PtRF, PmRS, RS.

We will consider arithmetical formulas in the signature $(0,=,S)$, where $S(x) = x+1$, for $x \in N$ (see [1]-[8]). Any term included in a formula of the mentioned kind has the form $S(S(...S(x)...))$ or $S(S(...S(0)...))$, where x is a variable. Such terms we will denote correspondingly by $S^k(x)$ and $S^k(0)$, where k is the quantity of symbols S contained in the considered term. We replace $S^0(x)$ and $S^0(0)$ with x and 0. Any elementary subformula of a formula of this kind has the form $t_1 = t_2$, where t_1 and t_2 are terms. Any arithmetical formula of this kind is obtained by the logical operations $\&, \vee, \supset, \neg, \forall, \exists$ from elementary formulas. We say that a formula is semi-elementary if it has the form $t_1 = t_2$ or $\neg(t_1 = t_2)$, where t_1 and t_2 are terms.

The deductive system of formal arithmetic in the signature $(0,=,S)$ is defined as in [4], [6]; we will denote this system by $\mathrm{Ded_S}$ (cf. [1], [2]). As it is proved in [4], this system is complete. We say that formulas F and G in the signature $(0,=,S)$ are $\mathrm{Ded_S}$-equivalent (or simply equivalent) if the formula $(F \supset G) \& (G \supset F)$ is deducible in $\mathrm{Ded_S}$. Below we consider formulas of the mentioned kind up to their $\mathrm{Ded_S}$-equivalence.

An arithmetical formula of the mentioned kind is said to be positive if it contains no other symbols of logical operations except $\exists, \&, \vee, \neg$, and all the symbols $\neg$ of negation relate to elementary subformulas containing no more than one variable (see [1], [2]). An arithmetical formula of this kind is said to be strongly positive if it can be obtained by the logical operations $\&$ and $\vee$ from semi-elementary formulas of the following forms: $x = a$, where x is a variable, a is a constant, $a \in N$; $x = y$, where x and y are variables; $x = S(y)$, where x and y are variables; $\neg(x = 0)$, where x is a variable. An arithmetical predicate is said to be positive (correspondingly, strongly positive), if it can be expressed by a positive (correspondingly, strongly positive) formula. An arithmetical set is said to be positive (correspondingly, strongly positive) if its representing predicate is positive (correspondingly, strongly positive).

The notion of one-dimensional creative set is given in a usual way ([3], [5], [7], [8]). We will slightly generalize this notion. We use a PmRF $c_n(x_1, x_2, ..., x_n)$, where $n \geq 2$, establishing a one-to-one correspondence between N^n and N (for example, $c_n(x_1, x_2, ..., x_n) = c_2(c_2(...c_2(c_2(x_1, x_2), x_3)..., x_{n-1}), x_n)$, where $c_2(x, y) = 2^x \cdot (2y+1) - 1$). We say that a set $B \subseteq N^n$ is an n-dimensional image of a set $A \subseteq N$ when $c_n(x_1, x_2, ..., x_n) \in A$ if and only if $(x_1, x_2, ..., x_n) \in B$. The set $B \in N^n$ is said to be creative in the generalized sense if it is an n-dimensional image of some one-dimensional creative set. Clearly, the properties of creative sets in the generalized sense are similar to the properties of one-dimensional creative sets (for example, all sets creative in the generalized sense are non-recursive).

Transitive closure A^* of an arithmetical set A having an even dimension $2k$ is defined in a usual way by the following generating rules (cf. [1], [2], [13]): (1) if $(x_1, x_2, ..., x_{2k}) \in A$, then $(x_1, x_2, ..., x_{2k}) \in A^*$, (2) if $(x_1, x_2, ..., x_k, y_1, y_2, ... y_k) \in A^*$, and $(y_1, y_2, ..., y_k, z_1, z_2, ... z_k) \in A^*$, then $(x_1, x_2, ..., x_k, z_1, z_2, ... z_k) \in A^*$.

Theorem 1: *For any $n \geq 3$ there exists a $2n$-dimensional strongly positive set such that its transitive closure is creative in the generalized sense.*

Theorem 2: *Transitive closure of any 2-dimensional strongly positive set is primitive recursive.*

The proof of Theorem 1 will be given below. The proof of Theorem 2 will be published later.

3. Auxiliary Notions and Statements

We will use some class of operator algorithms ([8], [17]) having a special structure. The algorithms belonging to this class we will call Ω-algorithms. Any Ω-algorithm consists of finite number of elementary Ω-algorithms, which will be called below "Ω-operators". The set of all Ω-operators included in the considered Ω-algorithm we call "scheme" of this Ω-algorithm. We suppose that some non-negative integer is attached to any Ω-operator in the scheme of a given Ω-algorithm in such a way, that different integers are attached to different Ω-operators. The integer attached to some Ω-operator we call "an identifier" of this Ω-operator. In this case we say that this Ω-operator has the mentioned identifier. Any Ω-operator implements one step of the process of computation realized by the considered Ω-algorithm. The objects transformed in the process of computation are non-negative integers. The state of the mentioned computation process is defined as a pair (α, w), where α is the identifier attached to the Ω-operator which is working on the considered step of the process, and w is the number obtained by the previous steps of the process. Ω-operators are algorithms having one of the following forms (where α is the identifier attached to the considered Ω-operator, β and γ are identifiers attached to Ω-operators which should work after the working of this Ω-operator):

(1) (α, end). This Ω-operator is called below "a final operator"; it finishes the process of computation.
(2) $(\alpha, \times 2, \beta)$. This Ω-operator transforms the state (α, w) to the state $(\beta, 2w)$.
(3) $(\alpha, \times 3, \beta)$. This Ω-operator transforms the state (α, w) to the state $(\beta, 3w)$.
(4) $(\alpha, :6, \beta, \gamma)$. This Ω-operator transforms the state (α, w) to the state $(\beta, \frac{w}{6})$ if the number w is divisible by 6; in the opposite case it transforms the state (α, w) to the state (γ, w).

Note that such forms of operators are considered actually in [17] (see also [8], p. 292, p. 312).

We suppose that any scheme of Ω-algorithm contains only a single final Ω-operator which has the identifier $\alpha = 0$. Among the operators contained in the scheme of the considered Ω-algorithm we distinguish the initial Ω-operator having the identifier $\alpha = 1$; the working of this operator begins the process of computation. The whole process of working of the given Ω-algorithm is described by the sequence of states (α_1, w_1), (α_2, w_2),…, (α_k, w_k),…,(where

$\alpha_1 = 1$) obtained during the working of this Ω-algorithm. The process is described by a finite sequence $(1, w_1)$, (α_2, w_2),…, $(0, w_m)$ if it is finished by the working of the final Ω-operator.

In this case we say that the considered Ω-algorithm transforms the state $(1, w_1)$ to the state $(0, w_m)$, and is applicable to the state $(1, w_1)$. If the final Ω-operator does not work during the process of computation, then the mentioned sequence $(1, w_1)$, (α_2, w_2),… is infinite. In this case we say that the considered Ω-algorithm is not applicable to the state $(1, w_1)$.

The following theorem is proved in [17] (see also [8], pp. 312-315) in some other terms.

Theorem 3 ([17]): *For any PtRF $f(x)$ there exists an Ω-algorithm which transforms the state $(1,2^{2^x})$ to the state $(0,2^{2^{f(x)}})$ when the value $f(x)$ is defined, and is not applicable to the state $(1,2^{2^x})$ in the opposite case.*

If some Ω-algorithm has the property described in Theorem 3, then we say that this Ω-algorithm realizes the PtRF $f(x)$. For example, the following Ω-algorithm:

$$(0, end),\ (1, \times 3, 2),\ (2, : 6, 1, 3),\ (3, \times 2, 0)$$

realizes the GRF $f(x) = 0$.

We will use also another classes of algorithms, namely, Γ_n-algorithms for $n \geq 1$.

These algorithms are actually special cases of graph-schemes with memory ([18]), though they will be described below in some other terms than the descriptions in [18].

Any Γ_n-algorithm consists of finite number of Γ_n-operators. The set of all Γ_n-operators included in the considered Γ_n-algorithm we call “scheme” of this Γ_n-algorithm. The index n in the notation Γ_n denotes that the objects transformed by the considered Γ_n-algorithm are n-tuples $(x_1, x_2, ..., x_n)$, where $x_i \in N$ for $1 \leq i \leq n$. The notion of identifier attached to the considered Γ_n-operator is defined similarly to the notion of “identifier attached to the considered Ω-operator” which is given above; we suppose that different Γ_n-operators have different identifiers attached to them. If some identifier is attached to a Γ_n-operator, we will say that this Γ_n-operator has the mentioned identifier.

The state of the computation process realized by a Γ_n-algorithm is defined as an $(n+1)$-tuple $(\alpha, x_2, x_3, ..., x_{n+1})$, where α is the identifier attached to the Γ_n-operator which is working on the considered step of the process, and $(x_2, x_3, ..., x_{n+1})$ is the n-tuple of numbers obtained by the previous steps of the process. Γ_n-operators are algorithms having one of the following forms (where the notations α, β, γ have the same sense as α, β, γ in the description of Ω-operators given above):

(1) (α, end). This Γ_n-operator we call “a final operator”; it finishes the process of computation.
(2) $(\alpha, x_i + 1, \beta)$, where $2 \leq i \leq n+1$. This Γ_n-operator transforms the state $(\alpha, x_2, x_3, ..., x_{i-1}, x_i, x_{i+1}, ..., x_{n+1})$ to the state $(\beta, x_2, x_3, ..., x_{i-1}, x_i + 1, x_{i+1}, ..., x_{n+1})$.

(3) $(\alpha, x_i \dot{-} 1, \beta)$, where $2 \le i \le n+1$; we denote by the symbol $\dot{-}$ the PmRF such that $x \dot{-} y = x - y$ when $x \ge y$, and $x \dot{-} y = 0$ when $x < y$ (cf. [3]-[8]). This Γ_n-operator transforms the state $(\alpha, x_2, x_3, ..., x_{i-1}, x_i, x_{i+1}, ..., x_{n+1})$ to the state $(\beta, x_2, x_3, ..., x_{i-1}, x_i \dot{-} 1, x_{i+1}, ..., x_{n+1})$.

(4) $(\alpha, x_i = 0, \beta, \gamma)$, where $2 \le i \le n+1$. This Γ_n-operator transforms the state $(\alpha, x_2, x_3, ..., x_{n+1})$ to the state $(\beta, x_2, x_3, ..., x_{n+1})$ when $x_i = 0$, and to the state $(\gamma, x_2, x_3, ..., x_{n+1})$ when $x_i \ne 0$.

We suppose that any scheme of Γ_n-algorithm contains only a single final Γ_n-operator which has the identifier $\alpha = 0$. Among the Γ_n-operators contained in the scheme of the considered Γ_n-algorithm we distinguish the <u>initial</u> Γ_n-operator having the identifier $\alpha = 1$; the working of this operator begins the process of computation. This process is described by a sequence of states (α_1, Q_1), (α_2, Q_2),…, (α_k, Q_k),… where $\alpha_1 = 1$, and any Q_i is an n-tuple $(x_2^{(i)}, x_3^{(i)}, ..., x_{n+1}^{(i)})$. Such a sequence is finite if the final Γ_n-operator works during the mentioned process, and is infinite in the opposite case. If the sequence of states is finite, then we say that the considered Γ_n-algorithm is <u>applicable</u> to the state $(1, Q_1)$; in this case we say also that Γ_n-algorithm <u>transforms</u> the state $(1, Q_1)$ to the state $(0, Q_m)$, where $(0, Q_m)$ is the last state in the considered sequence. If the sequence of states $(1, Q_1)$, $(2, Q_2)$,… is infinite, then we say that the considered Γ_n-algorithm is <u>not applicable</u> to the state $(1, Q_1)$.

We say that a Γ_n-algorithm (where $n \ge 2$) <u>realizes</u> a PtRF $f(x)$, if for any $x \in N$ it transforms the state $(1, 2^x, 0, 0, ..., 0)$ to the state $(0, 2^{f(x)}, 0, 0, ..., 0)$ when the value $f(x)$ is defined, and is not applicable to the state $(1, 2^x, 0, 0, ..., 0)$ when the value $f(x)$ is not defined. For example, the following Γ_n-algorithm realizes the PtRF $f(x)$ which is nowhere defined:
$(0, end)$, $(1, x_2 \dot{-} 1, 1)$.

Lemma 3.1: *If the initial state in the process of computation realized by some Ω-algorithm has the form $(1, 2^u, 3^v)$, where $u \in N$, $v \in N$, then any state (α_m, w_m) included in this process satisfies the condition $w_m = 2^t \cdot 3^s$, where $t, s \in N$.*

The proof is easily obtained from the definitions.

Lemma 3.2: *For any Ω-algorithm φ realizing some PtRF $f(x)$ there exists a Γ_2-algorithm ψ realizing the same PtRF $f(x)$.*

Proof: We will consider the process of computation realized by the Ω-algorithm φ. Any initial state in such a process has the form $(1, 2^{2^x})$ that is $(1, 2^{2^x} \cdot 3^0)$. As it is proved in Lemma 3.1 any state included in such a process has the form $(\alpha_m, 2^t \cdot 3^s)$ where $t, s \in N$. For any Ω-operator included in the scheme of Ω-algorithm φ we will construct some subscheme of the supposed Γ_2-algorithm ψ which has the following property: if the considered Ω-operator transforms the state $(\alpha, 2^u \cdot 3^v)$ to the state $(\beta, 2^t \cdot 3^s)$ then the corresponding subscheme of the supposed Γ_2-

algorithm ψ transforms the state (α, u, v) of Γ_2-algorithm ψ to the state (β, t, s). We will consider the following cases.

Case 1. The considered Ω-operator has the form $(\alpha, \times 2, \beta)$. In this case the required subscheme of the supposed Γ_2-algorithm ψ consists of the single Γ_2-operator $(\alpha, x_2 + 1, \beta)$.

Case 2. The considered Ω-operator has the form $(\alpha, \times 3, \beta)$. In this case the required subscheme of the supposed Γ_2-algorithm ψ consists of the single Γ_2-operator $(\alpha, x_3 + 1, \beta)$.

Case 3. The considered Ω-operator has the form $(\alpha, :6, \beta, \gamma)$. In this case the required subscheme of the supposed Γ_2-algorithm ψ consists of the following Γ_2-operators: $(\alpha, x_2 = 0, \gamma, \delta_1)$, $(\delta_1, x_3 = 0, \gamma, \delta_2)$, $(\delta_2, x_2 \dot{-} 1, \delta_3)$, $(\delta_3, x_3 \dot{-} 1, \beta)$. Here δ_1, δ_2, δ_3 are identifiers attached to additional Γ_2-operators which are included in the scheme of the supposed Γ_2-algorithm for modeling the working of the considered Ω-operator. Of course, these identifiers should be different in different subschemes of this kind.

Case 4. The considered Ω-operator has the form $(0, end)$. This Ω-operator does not transform the states of Ω-algorithm. So, the corresponding Γ_2-operator has the same form $(0, end)$.

The scheme of the supposed Γ_2-algorithm is obtained as the union of subschemes of the mentioned forms constructed for all Ω-operators included in the scheme of the given Ω-algorithm. It is easily seen that such Γ_2-algorithm satisfies the conditions of Lemma 3.2. This completes the proof.

Corollary 1: *For any PtRF $f(x)$ and any $n \geq 2$ there exists a Γ_n-algorithm realizing the PtRF $f(x)$.*

The proof is based on Theorem 3 and is similar to that of Lemma 3.2.

Note: *The statements established in Lemma 3.2 and in its Corollary 1 are similar to Theorem 7.1 in [18], where it is proved that any PtRF may be realized by some graph-scheme with memory constructed on the base of the functions $x+1$, $x \dot{-} 1$ and of the predicate $x = 0$. However, graph-schemes with memory corresponding to Γ_n-algorithms are essentially simpler than the graph-schemes considered in Theorem 7.1 in [18]. Besides, the definition of realizability of PtRf by Γ_n-algorithm differs from the corresponding definition in [18].*

Now let us define for any Γ_n-algorithm, where $n \geq 1$, the predicate describing one step of computation process realized by this Γ_n-algorithm. Such a predicate we will call “a step describing predicate”, or, shortly, “SD-predicate” for a given Γ_n-algorithm. Namely, if η is the SD-predicate for a given Γ_n-algorithm, then $\eta(x_1, x_2, ..., x_{2n+2})$ is true if and only if the given Γ_n-algorithm transforms the state $(x_1, x_2, ..., x_{n+1})$ to the state $(x_{n+2}, x_{n+3}, ..., x_{2n+2})$ by one step of the corresponding computation process. Let us note the following property of the predicate η: if $(x_1, x_2, ..., x_{n+1})$ is a state of the computational process realized by the considered Γ_n-algorithm, such that $x_1 \neq 0$, then there exists a single $(n+1)$-tuple $(x_{n+2}, x_{n+3},..., x_{2n+2})$ such that $\eta(x_1, x_2,..., x_{2n+2})$ is true.

The set of truth for the mentioned predicate η we will call "SD-set" for the considered Γ_n-algorithm. Clearly, such a set π has the following property: $(x_1, x_2,..., x_{2n+2}) \in \pi$ if and only if $(x_1, x_2,..., x_{n+1})$ is a state of computation process realized by the considered Γ_n-algorithm, and this Γ_n-algorithm transforms the state $(x_1, x_2,..., x_{n+1})$ to the state $(x_{n+2}, x_{n+3},..., x_{2n+2})$ by one step of the computation process.

Now let us define the forms of SD-predicates and SD-sets for Γ_n-algorithms. We suppose that some Γ_n-algorithm ψ, where $n \geq 1$ is fixed. We will define the forms of SD-predicates for any Γ_n-operator included in the scheme of ψ.

Case 1. The considered Γ_n-operator has the form $(\alpha, x_i + 1, \beta)$. Such Γ_n-operator transforms the state $(\alpha, x_2, x_3,..., x_{i-1}, x_i, x_{i+1},..., x_{n+1})$ to the state $(\beta, x_{n+3}, x_{n+4},..., x_{n+i}, x_{n+i+1}, x_{n+i+2},..., x_{2n+2})$, where $x_{n+3} = x_2$, $x_{n+4} = x_3, \ldots, x_{n+i} = x_{i-1}$, $x_{n+i+1} = x_i + 1$, $x_{n+i+2} = x_{i+1}, \ldots,$ $x_{2n+2} = x_{n+1}$.

The SD-predicate for such a Γ_n-operator is expressed by the following formula: $(x_1 = \alpha) \,\&\, (x_{n+2} = \beta) \,\&\, (x_{n+3} = x_2) \,\&\, (x_{n+4} = x_3) \,\&\, ... \,\&\, (x_{n+i} = x_{i-1}) \,\&\, (x_{n+i+1} = S(x_i)) \,\&$
$\&(x_{n+i+2} = x_{i+1}) \,\&\, ... \,\&\, (x_{2n+2} = x_{n+1})$.

Case 2. The considered Γ_n-operator has the form $(\alpha, x_i \dot{-} 1, \beta)$. Such Γ_n-operator transforms the state $(\alpha, x_2, x_3,..., x_{i-1}, x_i, x_{i+1},..., x_{n+1})$ to the state $(\beta, x_{n+3}, x_{n+4},..., x_{n+i}, x_{n+i+1}, x_{n+i+2},..., x_{2n+2})$, where $x_{n+3} = x_2$, $x_{n+4} = x_3, \ldots, x_{n+i} = x_{i-1}$, $x_{n+i+1} = x_i \dot{-} 1$, $x_{n+i+2} = x_{i+1}, \ldots,$ $x_{2n+2} = x_{n+1}$.

The SD-predicate for such a Γ_n-operator is expressed by the following formula: $(x_1 = \alpha) \,\&\, (x_{n+2} = \beta) \,\&\, (x_{n+3} = x_2) \,\&\, (x_{n+4} = x_3) \,\&\, ... \,\&\, (x_{n+i} = x_{i-1}) \,\&\, (x_{n+i+2} = x_{i+1}) \,\&\, ...$
$\&\, (x_{2n+2} = x_{n+1}) \,\&\, (((x_{n+i+1} = 0) \,\&\, (x_i = 0)) \vee (\neg(x_i = 0) \,\&\, (x_i = S(x_{n+i+1}))))$.

Case 3. The considered Γ_n-operator has the form $(\alpha, x_i = 0, \beta, \gamma)$. Such Γ_n-operator transforms the state $(\alpha, x_2, x_3,..., x_{n+1})$ to the states $(\beta, x_{n+3}, x_{n+4},..., x_{2n+2})$ or $(\gamma, x_{n+3}, x_{n+4},..., x_{2n+2})$ (where $x_{n+3} = x_2$, $x_{n+4} = x_3, \ldots, x_{2n+2} = x_{n+1}$) in the cases, when, correspondingly, $x_i = 0$ or $x_i \neq 0$. The SD-predicate for such a Γ_n-operator is expressed by the following formula: $(x_1 = \alpha) \,\&\, (x_{n+3} = x_2) \,\&\, (x_{n+4} = x_3) \,\&\, ... \,\&\, (x_{2n+2} = x_{n+1}) \,\&\, (((x_{n+2} = \beta) \,\&\, (x_i = 0)) \vee$
$((x_{n+2} = \gamma) \,\&\, \neg(x_i = 0)))$.

Case 4. The considered Γ_n-operator has the form $(0, end)$. Such Γ_n-operator does not transform the states of Γ_n-algorithm, so, an SD-predicate is not considered for such Γ_n-operator.

The SD-predicate for Γ_n-algorithm ψ is expressed by the formula obtained as the disjunction of formulas expressing SD-predicates constructed above for all Γ_n-operators contained in the scheme of ψ and different from the operator $(0, end)$. The SD-set for Γ_n-algorithm ψ is obtained as the set of truth for the corresponding SD-predicate. Clearly, such SD-set is a $(2n+2)$-dimensional arithmetical set.

Lemma 3.3: *SD-predicate and SD-set constructed for any* Γ_n*-algorithm, where* $n \geq 1$*, are strongly positive.*

The proof is obtained evidently from the definitions.

Lemma 3.4: (cf. [13], p.72). *If* A *is a* $2k$*-dimensional set,* $A \subseteq N^{2k}$*, then* $2k$*-tuple* $(x_1, x_2, ..., x_k, y_1, y_2, ..., y_k)$ *belongs to the transitive closure* A^* *of the set* A *if and only if there exists a sequence* $(Q_1, Q_2, ..., Q_m)$ *of* k*-tuples, such that* $m \geq 2$*,* $Q_1 = (x_1, x_2, ..., x_k)$*,* $Q_m = (y_1, y_2, ..., y_k)$ *and any* $2k$*-tuple* (Q_i, Q_{i+1}) *for* $1 \leq i \leq m-1$ *belongs to* A.

The proof is easily obtained using the definition of the transitive closure A^*.

4. Proof of Theorem 1

Let M be any one-dimensional creative set ([3], [5], [7], [8]). We consider the PtRF $f(x)$ such that $f(x) = 0$ when $x \in M$, and the value $f(x)$ is indefined when $x \notin M$. For any fixed $n \geq 2$ we construct (using Corollary of Lemma 3.2) a Γ_n-algorithm ψ realizing the PtRF $f(x)$; clearly, ψ transforms the state $(1,2^x,0,0,...,0)$ to the state $(0,1,0,0,...,0)$ when $x \in M$ and is not applicable to the state $(1,2^x,0,0,...,0)$ when $x \notin M$. Now, let us consider the SD-predicate η and SD-set π for ψ. Clearly, η is true for $(2n+2)$-tuple $(x_1, x_2, ..., x_{n+1}, y_1, y_2, ..., y_{n+1})$ (and the statement $(x_1, x_2, ..., x_{n+1}, y_1, y_2, ..., y_{n+1}) \in \pi$ holds) if and only if ψ transforms the state $(x_1, x_2, ..., x_{n+1})$ to the state $(y_1, y_2, ..., y_{n+1})$ by one step of the process of computation. Let us consider the transitive closure π^* of the SD-set π.

Using Lemma 3.4 we conclude that $(x_1, x_2, ..., x_{n+1}, y_1, y_2, ..., y_{n+1}) \in \pi^*$ if and only if there exists a sequence $(Q_1, Q_2, ..., Q_m)$ of $(n+1)$-tuples such that $Q_1 = (x_1, x_2, ..., x_{n+1})$, $Q_m = (y_1, y_2, ..., y_{n+1})$, and $(Q_i, Q_{i+1}) \in \pi$ for any i such that $1 \leq i < m$. But in this case the sequence $(Q_1, Q_2, ..., Q_m)$ is a sequence of states of the Γ_n-algorithm ψ which describes some part of a process of computation implemented by the Γ_n-algorithm ψ.

Hence, the $(2n+2)$-tuple $(1,2^x,0,0,...,0,0,1,0,0,...,0)$ belongs to π^* if $x \in M$. It is easily seen that the mentioned $(2n+2)$-tuple does not belong to π^* if $x \notin M$. Let us consider the set $\pi^{**} \in N$ such that its $(2n+2)$-dimensional image is π^*. Then $c_{2n+2}(1,2^x,0,0,...,0,0,1,00,...,0) \in \pi^{**}$ if and only if $x \in M$. So the set M is m-reducible to the set π^{**}. Using the corresponding theorem concerning m-reducibility (see, for example, [8], p. 161), we conclude that the set π^{**} is creative, the set π^* is creative in the generalized sense, and the set π is strongly positive (see Lemma 3.3). This completes the proof.

Note: *It is seen from Theorem 1 that the transitive closures of some strongly positive sets having the dimensions 6, 8, 10, … are creative in the generalized sense. On the other side (Theorem 2) the transitive closure of any 2-dimensional strongly positive set is primitive recursive. Similar problem concerning 4-dimensional strongly positive sets remains open.*

References

[1] S. N. Manukian, “On the representation of recursively enumerable sets in weak arithmetics”, *Transactions of the IIAP of NAS RA, Mathematical Problems of Computer Science*, vol. 27, pp. 90--110, 2006.

[2] S. N. Manukian, “On an algebraic classification of multidimensional recursively enumerable sets expressible in formal arithmetical systems”, *Transactions of the IIAP of NAS RA, Mathematical Problems of Computer Science*, vol. 41, pp. 103-113, 2014.

[3] S. C. Kleene, *Introduction to Metamathematics*, D, Van Nostrand Comp., Inc., New York-Toronto, 1952.

[4] H. B. Enderton, *A Mathematical Introduction to Logic*, 2nd edition, San Diego, Harcourt, Academic Press, 2001.

[5] E. Mendelson, *Introduction to Mathematical Logic*, D, Van Nostrand Comp., Inc., Princeton-Toronto-New York-London, 1964.

[6] D. Hilbert and P. Bernays, *Grundlagen der Mathematik*, Band1, Zweite Auflage, Berlin-Heidelberg-New York, Springer Verlag, 1968.

[7] H. Rogers, *Theory of Recursive Functions and Effective Computability*, McGraw Hill Book Comp., New York-St. Louis-San Francisco-Toronto-London-Sydney, 1967.

[8] A. I. Malcev, *Algorithms and Recursive Functions*, 2nd edition, (in Russian), 1986.

[9] A. A. Markov, “Impossibility of some algorithms in the theory of associative systems”, *Reports of the Acad. Sci. USSR*, (in Russian), vol. 55, no. 7, pp. 587-590, 1947.

[10] E. L. Post, “Recursive unsolvability of a problem of Thue”, *Journ. of Symb. Logic*, vol. 12, pp. 1-11, 1947.

[11] P. S. Novikov, “On the algorithmic unsolvability of identity problem in the group theory”,*Transactions of Steklov Institute of the Acad. Sci. USSR*, (in Russian), vol. 44, 1955.

[12] G. S. Tseytin, “Associative calculus with the unsolvable problem of equivalence”, *Transactions of Steklov Institute of the Acad. Sci. USSR*, (in Russian), vol. 52, pp. 172-189, 1958.

[13] G. S. Tseytin, “One method of representation for the theory of algorithms and enumerable sets”, *Transactions of Steklov Institute of the Acad. Sci. USSR*, (in Russian), vol. 72, pp. 69-98, 1964.

[14] S. N. Manukian,. “Classification of many-dimensional arithmetical sets represented in M. Presburger’s system”, *Reports of the National Acad. Sci. of Armenia*, (in Russian), vol. 111, no. 2, pp. 114--120, 2011.

[15] M. Presburger, “Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt”, *Comptes Rendu du I Congres des Mathematiciens des Pays Slaves*, Warszawa, pp. 92-101, 1930.

[16] R. Stransifer, *Presburger’s Article on Integer Arithmetics: Remarks and Translation*, Department of Computer Science, CornellUniversity, Ithaca, New York, 1984.

[17] M. L. Minsky, “Recursive unsolvability of Post’s problem of “Tag” and topics in theory of Turing machines”, *Ann. Math.*, vol. 74, pp. 437-455, 1961.

[18] I.D. Zaslavsky, “Graph-schemes with memory”, *Transactions of Steklov Institute of Acad. Sci. USSR*, (in Russian), vol. 72, pp. 99-192, 1964.

Submitted 25.11.2014, accepted 27.01.2015.

Խիստ պոզիտիվ բազմաչափ թվաբանական բազմությունների մասին

Ս. Մանուկյան

Ամփոփում

[1]-ում և [2]-ում սահմանվում և հետազոտվում է պոզիտիվ թվաբանական բանաձևի գաղափարը $(0,=,S)$ սիգնատուրայում, (որտեղ $S(x)=x+1$): Բազմաչափթվաբանական բազմությունը կոչվում է պոզիտիվ, եթե այն որոշվում է որևէ պոզիտիվ բանաձևի միջոցով: Դիտարկվում է պոզիտիվ բազմությունների դասի որևէ ենթադաս, այսինքն՝ խիստ պոզիտիվ բազմությունների դասը: Ապացուցվում է, որ ցանկացած n-ի համար, որտեղ $n \geq 3$, գոյություն ունի $2n$-չափանի խիստ պոզիտիվ բազմություն, որի տրանզիտիվ փակումը ռեկուրսիվ չէ: Մյուս կողմից նշվում է, որ ցանկացած 2-չափանի խիստ պոզիտիվ բազմություն ունի պարզագույն ռեկուրսիվ տրանզիտիվ փակում:

О строго позитивных многомерных арифметических множествах

С. Манукян

Аннотация

Понятие позитивной арифметической формулы в сигнатуре $(0,=,S)$, где $S(x)=x+1$, определено и исследовано в [1] и [2]. Многомерное арифметическое множество называем позитивным, если оно задаётся позитивной формулой. Рассматривается подкласс класса позитивных множеств, а именно, класс строго позитивных множеств. Доказывается, что для всякого $n \geq 3$ существует строго позитивное множество размерности $2n$, такое, что его транзитивное замыкание нерекурсивно. С другой стороны, указывается, что транзитивное замыкание всякого строго позитивного множества размерности 2 примитивно рекурсивно.