On some generalizations of the primitive recursive arithmetic

I.D. Zaslavsky

Institute for Informatics and Automation Problems of the National Academy of Sciences of Armenia,
Parujr Sevak str. 1, Yerevan 375014, Armenia

Abstract

Formal arithmetical system PRAU is defined as an extension of R.L. Goodstein’s system PRA of the primitive recursive arithmetic; it is based on the consideration of functions similar to primitive recursive functions but in general not everywhere defined. It is proved that PRAU is a conservative extension of PRA. Some classes of the program schemes (PRA-schemes and PRAU-schemes) are introduced; it is proved that the classes of functions computable by such schemes coincide with the classes of functions taking part, correspondingly, in PRA and PRAU.

Keywords: Recursive; Branching; Superposition; Primitive; Memory; Loading operator; Logical operator

1. Introduction

The classification of arithmetical functions is considered in many aspects; but such considerations relate in most cases to everywhere defined functions (the class of partially recursive functions gives one of exceptions). However, partially defined functions play an important role in the theory of computation and it is natural to consider also classifications of arithmetical functions which are not necessarily everywhere defined. For example, such well-known functions of natural numbers as $x - y$ or $x/y$ (which are admitted to be undefined when $x < y$ or, correspondingly, $y$ is not a divisor of $x$) possess many properties of primitive recursive functions (for example, from the point of view of their computational complexity), and the description of properties of such functions leads to the consideration of a class of partially defined functions similar to that of primitive recursive functions. Below such a class of generalized primitive...
recursive functions (GPRF) is considered; some logical and computational systems based on GPRF are investigated. In Section 2, the formal system PRAU is defined; it is an extension of R.L. Goodstein’s system PRA of the primitive recursive arithmetic. It is proved (Theorem 2.1) that PRAU is a conservative extension of PRA, i.e. if some formula in the language of PRA is deducible in PRAU, then it is deducible also in PRA. In Section 3, two classes of program schemes (PRA-schemes and PRAU-schemes) are defined; the functions computable by PRA-schemes (correspondingly, PRAU-schemes) are primitive recursive functions (correspondingly, GPRF) and only such functions (Theorem 3.1). The formulations of the Theorems 2.1 and 3.1 were actually given in [9].

2. Generalized primitive recursive arithmetic

We shall consider multidimensional functions on the set \( N = \{0, 1, 2, \ldots \} \) of natural numbers, not necessarily defined everywhere. Such functions will be called, as usual, arithmetical functions. By \( \varphi(x_1, x_2, \ldots, x_n) \), where \( \varphi \) is an arithmetical function, we denote the statement: “the function \( \varphi \) is defined on the point \( (x_1, x_2, \ldots, x_n) \)”.

The notion of primitive recursive function (PRF) is introduced in the usual way [3–6]. The formal system of primitive recursive arithmetic (PRA) is defined as in [1,7,8]. We shall use the notations for individual PRF, for example, \( x+y, x \cdot y, \sigma(x) = x+1, x-y, \lfloor x-y \rfloor, s_g(x), \overline{s}_g(x) \) and so on [3–6]. Let us recall some definitions of operations on arithmetical functions; the corresponding notions are actually well-known [3–6], but we shall use the form of their definitions given below.

The operations of superposition and primitive recursion for arithmetical functions are defined as follows [3–6]. The function \( \varphi \) depending on \( n \) variables is said to be obtained by the operation of superposition from the functions \( \psi, \eta_1, \eta_2, \ldots, \eta_k \) depending, correspondingly, on \( k, n, n, \ldots, n \) variables, if the following equality holds:

\[
\varphi(x_1, x_2, \ldots, x_n) = \psi(\eta_1(x_1, x_2, \ldots, x_n), \eta_2(x_1, x_2, \ldots, x_n), \ldots, \eta_k(x_1, x_2, \ldots, x_n)),
\]

(2.1)

i.e. \( \varphi(x_1, x_2, \ldots, x_n) \) if and only if \( \psi(\eta_1(x_1, x_2, \ldots, x_n), \eta_2(x_1, x_2, \ldots, x_n), \ldots, \eta_k(x_1, x_2, \ldots, x_n)) \) and \( \psi(y_1, y_2, \ldots, y_k) \), where \( y_i = \eta_i(x_1, x_2, \ldots, x_n) \) for \( 1 \leq i \leq k \); in this case \( \varphi(x_1, x_2, \ldots, x_n) = \psi(y_1, y_2, \ldots, y_k) \). The function \( \varphi \) depending on \( n+1 \) variables is said to be obtained by the operation of primitive recursion from the functions \( \alpha \) and \( \beta \) depending, correspondingly, on \( n \) and \( n+2 \) variables, if the following equalities hold:

\[
\varphi(x_1, x_2, \ldots, x_n, 0) = \alpha(x_1, x_2, \ldots, x_n),
\]

\[
\varphi(x_1, x_2, \ldots, x_n, \sigma(y)) = \beta(x_1, x_2, \ldots, x_n, y, \varphi(x_1, x_2, \ldots, x_n, y)),
\]

(2.2)

i.e. \( \varphi(x_1, x_2, \ldots, x_n, y) \) if and only if there exists a finite sequence of natural numbers \( w_0, w_1, \ldots, w_y \) such that \( w_0 = \alpha(x_1, x_2, \ldots, x_n) \), \( w_{i+1} = \beta(x_1, x_2, \ldots, x_n, i, w_i) \) for \( 0 \leq i < y \); in this case \( \varphi(x_1, x_2, \ldots, x_n, y) = w_y \). The operations of branching and restricted branching are defined as follows. The function \( \varphi \) depending on \( n \) variables is said to be a
branching of the functions $\omega, \psi, \eta$ depending on $n$ variables if the value $\varphi(x_1, x_2, \ldots, x_n)$ is undefined in the points, where $\omega(x_1, x_2, \ldots, x_n)$ is undefined, and this value is equal to $\psi(x_1, x_2, \ldots, x_n)$ (correspondingly, $\eta(x_1, x_2, \ldots, x_n)$) in the points, where $\omega(x_1, x_2, \ldots, x_n) = 0$ (correspondingly, $\omega(x_1, x_2, \ldots, x_n)$ is defined and $\omega(x_1, x_2, \ldots, x_n) \neq 0$). Such a function $\varphi$ will be denoted by the following expression:

$$\varphi(x_1, x_2, \ldots, x_n) = \begin{cases} \omega(x_1, x_2, \ldots, x_n) = 0 & \text{then } \psi(x_1, x_2, \ldots, x_n) \\ \text{else } \eta(x_1, x_2, \ldots, x_n). \end{cases} \quad (2.3)$$

The restricted branching $\varphi_1$ of the functions $\omega, \psi, \eta$ is defined as follows:

$$\varphi_1(x_1, x_2, \ldots, x_n) = \begin{cases} \omega(x_1, x_2, \ldots, x_n) = 0 & \text{then } \psi(x_1, x_2, \ldots, x_n) \text{ else} \\ \eta(x_1, x_2, \ldots, x_n) = \overline{sg}(\omega(x_1, x_2, \ldots, x_n)) \cdot \psi(x_1, x_2, \ldots, x_n) \\ + sg(\omega(x_1, x_2, \ldots, x_n)) \cdot \eta(x_1, x_2, \ldots, x_n). \end{cases}$$

Below the particles “If” and “else” will be used for the notation of, correspondingly, branching and restricted branching.

Clearly, the branching and the restricted branching of $\omega, \psi, \eta$ coincide, if $\omega, \varphi, \eta$ are total functions, but in general it is not so. For example, if $\omega(x) = x - 3$, $\psi(x) = 3 - x$, $\eta(x) = x - 3$ (where the function $x - y$ is admitted to be undefined when $x < y$), then we have

$$\begin{array}{c}
\text{If } \omega(x) = 0 \text{ then } \psi(x) \text{ else } \eta(x) = |x - 3|,
\end{array}$$

but the restricted branching of $\omega, \psi, \varphi$ is equal to 0 when $x = 3$ and is undefined when $x \neq 3$.

Now let us give the definition of the formal system PRAU. An arithmetical function is said to be GPRF if it can be obtained from the basic functions $D(x) = 0$, $I^i_k(x_1, x_2, \ldots, x_n) = x_k$, $\sigma(x) = x + 1$, and $U(x)$, which is a function nowhere defined, using the operations of superposition and primitive recursion. So, the definition of GPRF differs from that of PRF only by introducing of nowhere defined function $U(x)$ in the list of basic functions. The language of the system PRAU contains the variables $x_1, x_2, \ldots, x_n$, the symbol of the constant 0, and the functional symbols $f_1, f_2, \ldots$ for all GPRF; the index $i$ of every functional symbol $f_i$ in PRAU is defined in such a way that it contains the complete information concerning the process of constructing the GPRF denoted by $f_i$ from the basic functions using the operations of superposition and primitive recursion. Obviously, every PRF is a GPRF, and for any functional symbol $f$ in PRA there is a functional symbol in PRAU expressing the same function as $f$; we shall denote such a functional symbol in PRAU by $f'$. The notion of the term is given in a usual way [3–6]. By $\text{Subst}(t, x, s)$, where $t$ and $s$ are terms, $x$ is a variable, we denote the term obtained by the substitution of $s$ for all occurrences of $x$ in $t$. For any term $t$ in PRA, we define the term $t'$ in PRAU obtained by replacing every functional symbol $f$ in $t$ by the corresponding functional symbol $f'$. Clearly, $t$ and $t'$ express the same PRF. The formulas in PRAU are defined as formal expressions having the form $t = s$ and $!t$, where $t$ and $s$ are any terms. The formula $!t$ is said to be true, if
the function expressed by the term \( t \) is total. The formula \( t = s \) is said to be true, if the functions expressed by the terms \( t \) and \( s \) are equal (i.e. they are simultaneously defined or simultaneously undefined on every point and have equal values on any point where they are defined). The axioms in the system PRAU are introduced as follows (where \( t \) is any term, and \( x_1, x_2, \ldots, x_n, \ldots \) are any variables):

1. \( t = t \);
2. \( D(x) = 0 \);
3. \( \delta^k (x_1, x_2, \ldots, x_n) = x_k \);
4. all equalities having the form (2.1) and (2.2) for all GPRF obtained, correspondingly, by the operations of superposition and primitive recursion;
5. all equalities having the form \( r = U(x) \), where \( r \) is any term containing the functional symbol \( U \);
6. all formulas having the form \( !r \), where \( r \) is any term in PRA.

The rules of inference in PRAU are introduced as follows (in these definitions \( t, s, r \) are arbitrary terms, \( x \) and \( y \) are arbitrary variables, \( u \) and \( v \) are any terms containing the variable \( x \)):

- (Symm) \( \frac{t = s}{s = t} \)
- (Trans) \( \frac{t = s; s = r}{t = r} \)
- (Sbu1) \( \frac{\text{Subst}(t, x, r) = \text{Subst}(s, x, r)}{t = x \longrightarrow \text{Subst}(t, x, r) \text{ and } s = x \longrightarrow \text{Subst}(s, x, r)} \)
- (Sbu2) \( \frac{\text{Subst}(u, x, r) = \text{Subst}(v, x, r)}{s = x \longrightarrow \text{Subst}(u, x, r) \text{ and } s = x \longrightarrow \text{Subst}(v, x, r)} \)
- (PR) \( \frac{\text{Subst}(t, x, y) = \text{Subst}(s, x, y) \text{ and } \text{Subst}(r, x, 0) = \text{Subst}(s, x, 0) \text{ and } \text{Subst}(r, x, \sigma(x)) = \text{Subst}(s, x, \sigma(x)) \text{ and } \text{Subst}(r, y, t) = \text{Subst}(s, y, t)}{t = x \longrightarrow \text{Subst}(t, x, y) \text{ and } s = x \longrightarrow \text{Subst}(s, x, y) \text{ and } r = x \longrightarrow \text{Subst}(r, x, y) \text{ and } s = x \longrightarrow \text{Subst}(s, x, y)} \)

The notion of a formula deducible in PRAU is defined in the usual way on the base of introduced axioms and rules of inference. It is easily seen that every formula deducible in PRAU is true, but the reverse in general does not hold (sf. [1]). This is so also in PRA.

Let us note that the definition of PRA may be given in the form similar to the definition of PRAU given above. In such a form of PRA the language consists only in formulas having the form \( t = s \), the list of axioms consists only in axioms of the kind (1)–(4) given above, and the list of inference rules includes the rules (Symm), (Trans), (Sb2), (PR), and besides, the following rule:

- (Sb1) \( \text{Subst}(t, x, r) = \text{Subst}(s, x, r) \).

Below we shall admit, that the system PRA is given in the form described here; it is easily seen that it is equivalent to the forms of PRA usually considered [1,7,8]. Let us note that the rule (Sb1) is not valid in PRAU. For example, the formula \( 0 \cdot x = 0 \) is true, but the substitution of \( U(x) \) for \( x \) gives the formula which is not true.

Obviously, PRAU can be considered as an extension of PRA. We shall prove that PRAU is a conservative extension of PRA. More precisely, the following theorem holds:

**Theorem 2.1.** If \( t \) and \( s \) are terms in PRA such that the formula \( t' = s' \) is deducible in PRAU, then \( t = s \) is deducible in PRA.

For the proof of this theorem we shall introduce some auxiliary notions and prove the Lemmas 2.1–2.4.
By $Br(x, y)$ ("Branching function") we denote a GPRF defined as follows by the operation of primitive recursion:

$$Br(x, 0) = 0;$$

$$Br(x, \sigma(y)) = U(I^*_1(x, y, Br(x, y))).$$

So, $Br(x, y) = x$ when $y = 0$, and $Br(x, y)$ is undefined when $y > 0$.

If $f$ is an arithmetical function, then the standard image or $S$-image of $f$ is defined as a total function $f^*$ such that

$$f^*(x_1, x_2, \ldots, x_n) = \begin{cases} \sigma(f(x_1, x_2, \ldots, x_n)), & \text{if } f(x_1, x_2, \ldots, x_n); \\ 0, & \text{otherwise.} \end{cases}$$

Lemma 2.1. An arithmetical function is a GPRF if and only if its $S$-image is a PRF.

Proof. Using the induction on the process of generating a considered GPRF we shall prove that the $S$-image of any GPRF is a PRF. Obviously, $S$-images of basic functions in PRAU are PRF. Further, if a function $f$ is obtained from the GPRF $g_1, h_1, h_2, \ldots, h_k$ by the operation of superposition, i.e.

$$f(x_1, x_2, \ldots, x_n) = g(h_1(x_1, x_2, \ldots, x_n), h_2(x_1, x_2, \ldots, x_n), \ldots, h_k(x_1, x_2, \ldots, x_n))$$

then, as it is easily seen, the $S$-image $f^*$ of the function $f$ satisfies the equation

$$f^*(x_1, x_2, \ldots, x_n) = g^*(h_1^*(x_1, x_2, \ldots, x_n) \cdot 1, h_2^*(x_1, x_2, \ldots, x_n) \cdot 1, \ldots, h_k^*(x_1, x_2, \ldots, x_n) \cdot 1) \cdot \prod_{i=1}^k \sigma(g_i^*(x_1, x_2, \ldots, x_n)), \quad (2.4)$$

where $g^*, h_1^*, h_2^*, \ldots, h_k^*$ are $S$-images of $g, h_1, h_2, \ldots, h_k$. By induction we have that $g^*, h_1^*, h_2^*, \ldots, h_k^*$ are PRF, hence $f^*$ is also a PRF. If a function $f$ is obtained from the GPRF $g$ and $h$ by the operation of primitive recursion, i.e.

$$f(x_1, x_2, \ldots, x_n, 0) = g(x_1, x_2, \ldots, x_n);$$

$$f(x_1, x_2, \ldots, x_n, \sigma(y)) = h(x_1, x_2, \ldots, x_n, y, f(x_1, x_2, \ldots, x_n, y)),$$

then, as it is easily seen, the $S$-image $f^*$ of the function $f$ satisfies the equations

$$f^*(x_1, x_2, \ldots, x_n, 0) = g^*(x_1, x_2, \ldots, x_n),$$

$$f^*(x_1, x_2, \ldots, x_n, \sigma(y)) = h^*(x_1, x_2, \ldots, x_n, y, f^*(x_1, x_2, \ldots, x_n, y)), \quad (2.5)$$

where

$$h^*(x_1, x_2, \ldots, x_n, y, w) = h^*(x_1, x_2, \ldots, x_n, y, w \cdot 1) \cdot \sigma(w),$$

and $g^*, h^*$ are $S$-images of $g, h$. So, we have proved that the $S$-image of every GPRF is a PRF. Now let us suppose that the $S$-image $f^*$ of some arithmetical function $f$ is a PRF. Clearly, $f^*$ is a GPRF. Then the function $f$ satisfies the equality

$$f(x_1, x_2, \ldots, x_n) = Br(f^*(x_1, x_2, \ldots, x_n) \cdot 1, \sigma(f^*(x_1, x_2, \ldots, x_n, y))).$$
Lemma 2.2. If \( \omega, \psi, \eta \) are GPRF, then the functions If \( \omega(x_1, x_2, \ldots, x_n) \) then \( \psi(x_1, x_2, \ldots, x_n) \) else \( \eta(x_1, x_2, \ldots, x_n) \) and if \( \omega(x_1, x_2, \ldots, x_n) \) then \( \psi(x_1, x_2, \ldots, x_n) \) else \( \eta(x_1, x_2, \ldots, x_n) \) are GPRF.

Proof. For the restricted branching of \( \omega, \psi, \eta \) the statement of the lemma is obvious. Let \( \varphi(x_1, x_2, \ldots, x_n) \) be the branching of \( \omega, \psi, \eta \). Let \( \varphi^*, \omega^*, \psi^*, \eta^* \) be \( S \)-images of, correspondingly, \( \varphi, \omega, \psi, \eta \). Then, as it easily seen, the following equality holds:

\[
\varphi^*(x_1, x_2, \ldots, x_n) = \begin{cases} 
0 & \text{if } \omega^*(x_1, x_2, \ldots, x_n) = 0 \\
\varphi^*(x_1, x_2, \ldots, x_n) & \text{else}
\end{cases}
\]

\[
(\text{if } |\omega^*(x_1, x_2, \ldots, x_n) - 1| = 0 \text{ then } \psi^*(x_1, x_2, \ldots, x_n) \text{ else } \eta^*(x_1, x_2, \ldots, x_n)).
\]

Hence \( \varphi^* \) is a PRF, and \( \varphi \) is a GPRF. This completes the proof. \( \square \)

For every term \( t \) in PRAU let us define now its \( S \)-image \( t^* \) in PRA expressing the \( S \)-image of GPRF described by the term \( t \). Namely, for every functional symbol \( f \) in PRAU let us define the functional symbol \( f^* \) in PRA expressing the \( S \)-image of the function described by \( f \). The symbol \( f^* \) in the case when \( f \) expresses a basic function in PRAU is defined as follows: if \( f \) expresses \( \sigma(x), I^k_n(x_1, x_2, \ldots, x_n), D(x), U(x) \), then \( f^* \) is the functional symbol, correspondingly, for \( \sigma(\sigma(x)), \sigma(I^k_n(x_1, x_2, \ldots, x_n)), \sigma(D(x)), D(x) \). If the function described by \( f \) is obtained in PRAU by the operations of superposition or primitive recursion, then the function described by \( f^* \) is obtained in PRAU by Eqs. (2.4) and (2.5). These conditions define the symbol \( f^* \) in PRA for every symbol \( f \) in PRAU. Let us define also for every \( f \) in PRAU the functional symbol \( \overline{f} \) in PRA such that the following equality holds:

\[
\overline{f}(x_1, x_2, \ldots, x_n) = f^*(x_1 \div 1, x_2 \div 1, \ldots, x_n \div 1) \cdot \prod_{i=1}^{n} s g(x_i).
\]

According to (2.4) we can conclude that if

\[
f(x_1, x_2, \ldots, x_n) = g(h_1(x_1, x_2, \ldots, x_n), h_2(x_1, x_2, \ldots, x_n), \ldots, h_k(x_1, x_2, \ldots, x_n))
\]

in PRAU, then

\[
f^*(x_1, x_2, \ldots, x_n) = \overline{g}(h_1^*(x_1, x_2, \ldots, x_n), h_2^*(x_1, x_2, \ldots, x_n), \ldots, h_k^*(x_1, x_2, \ldots, x_n))
\]

in PRA.

We define now the term \( t^* \) in PRA for every term \( t \) in PRAU as follows. If \( t \) is a variable \( x_i \) or a constant 0, then \( t^* \) is \( \sigma(t) \). If \( t \) is a term in PRAU having the form \( f(t_1, t_2, \ldots, t_n) \) then we define \( t^* \) by induction as \( \overline{f}(t_1^*, t_2^*, \ldots, t_n^*) \). It follows from the consideration above that every term \( t^* \) expresses in PRA the \( S \)-image of the function described by the term \( t \) in PRAU.
Lemma 2.3. For every term \( t \) in PRA the equality \( t^* = \sigma(t) \) is deducible in PRA.

Proof. We shall prove at first the equality \( t^* = \sigma(t) \) for the case, when the term \( t' \) has the form \( f(x_1, x_2, \ldots, x_n) \). We use the induction on the process of constructing a PRF denoted by \( f \). If the PRF denoted by \( f \) is a basic function then the equality \( t^* = \sigma(t) \) is obtained directly from the definition of \( t^* \). If the mentioned function is obtained by the superposition or primitive recursion then the equality \( f^*(x_1, x_2, \ldots, x_n) = \sigma(f(x_1, x_2, \ldots, x_n)) \) is easily obtained from (2.4) and (2.5) using the equalities \( sg(\sigma(x)) = 1 \) and \( \sigma(x) - 1 = x \) which are easily obtained in PRA; in the case of primitive recursion the rule (PR) is used.

Now in the general case we use the induction on the process of constructing the term \( t \). If \( t \) is a variable or a constant 0, then the required equality is obtained immediately from the definitions. If the term \( t \) has the form \( f(t_1, t_2, \ldots, t_n) \), then \( t^* = f^*(t_1^*, t_2^*, \ldots, t_n^*) \), and the required equality is easily obtained using equalities \( t_i^* = \sigma(t_i) \) for \( 1 \leq i \leq n \) and the definition of \( f^* \). This completes the proof. \( \square \)

Lemma 2.4. If \( t \) and \( s \) are terms in PRAU such that \( t = s \) is deducible in PRAU, then \( t^* = s^* \) is deducible in PRA.

The proof is similar to the proof of the preceding lemma using the induction on the process of the deduction of \( t = s \) in PRAU.

Proof of the Theorem 2.1. Let us suppose that the formula \( t' = s' \) is deducible in PRAU. Then it follows from Lemma 2.4 that \( t^* = s^* \) is deducible in PRA. Now we conclude from Lemma 2.3 that \( \sigma(t) = \sigma(s) \) is deducible in PRA. Hence \( t = s \) is also deducible in PRA. This completes the proof. \( \square \)

3. Program schemes computing PRF and GPRF

In this section some classes of program schemes computing PRF and GPRF are considered. Classes of program schemes computing PRF are well-known (see, for example, [2]); but the classes of programs considered below have special features giving the possibility for computing PRF and GPRF in a similar way.

The \( n \)-dimensional memory is a set of variables \( (x_1, x_2, \ldots, x_n) \) whose values are any natural numbers \( 0, 1, 2, \ldots \). The notions of a state of memory and of a transformation of memory are given in a natural way; we consider transformations of memory which are in general not everywhere defined. The components of a transformation \( \Phi \) of a memory \( (x_1, x_2, \ldots, x_n) \) are arithmetical functions \( \varphi_1, \varphi_2, \ldots, \varphi_n \) of \( n \) variables such that if the transformation \( \Phi \) is defined on a state \( K = (k_1, k_2, \ldots, k_n) \), then the state of memory \( \Phi(K) = (\varphi_1(k_1, k_2, \ldots, k_n), \varphi_2(k_1, k_2, \ldots, k_n), \ldots, \varphi_n(k_1, k_2, \ldots, k_n)) \); the functions \( \varphi_1, \varphi_2, \ldots, \varphi_n \) are defined on \( (k_1, k_2, \ldots, k_n) \) if and only if \( \Phi \) is defined on the state \( K = (k_1, k_2, \ldots, k_n) \).

Let us give the definition of PRA-scheme on a memory \( (x_1, x_2, \ldots, x_n) \). Elementary PRA-schemes on a memory \( (x_1, x_2, \ldots, x_n) \) are formal expressions having one of the
forms \( x_i := x_j, \) \( x_i := 0, \) \( x_i := x_j + 1, \) where \( 1 \leq i, j \leq n. \) These expressions are interpreted in a natural way as transformations of memory \((x_1, x_2, \ldots, x_n)\). They will be called, as usual, loading operators; the parts of such expression from the left and right side of the sign \( := \) will be called the left side and the right side of this expression. PRA-schemes on the memory \((x_1, x_2, \ldots, x_n)\) are defined inductively as formal expressions obtained by the following generating rules (1), (2), (3):

1. Every elementary PRA-scheme on a memory \((x_1, x_2, \ldots, x_n)\) is a PRA-scheme on the same memory.
2. If \( \Omega_1, \Omega_2, \ldots, \Omega_m \) are formal expressions such that every \( \Omega_i \) is either an already constructed PRA-scheme on the memory \((x_1, x_2, \ldots, x_n)\) or an expression having the form

\[
\text{if } x_i = x_j \text{ then go to } r \text{ else go to } s, \tag{3.1}
\]

where \( 1 \leq i, j \leq n, \) and \( r, s \) are natural numbers such that \( r > t, s > t, \) then the expression

\[
\text{begin } \Omega_1; \Omega_2; \ldots; \Omega_m \text{ end} \tag{3.2}
\]

is also a PRA-scheme on the memory \((x_1, x_2, \ldots, x_n)\).
3. If \( \Omega \) is an already constructed PRA-scheme on the memory \((x_1, x_2, \ldots, x_n)\), and the variables \( x_i \) and \( x_j \), where \( 1 \leq i, j \leq n, \) are not contained in the left sides of loading operators in \( \Omega \), then the expression

\[
\text{for } x_i := 0 \text{ when } x_i < x_j \text{ do } \Omega
\]

is also a PRA-scheme on the memory \((x_1, x_2, \ldots, x_n)\).

Expressions having the form (3.1) will be called, as usual, logical operators.

Let \( \Sigma \) be a PRA-scheme on some memory; the process of its working originating from a state \( K \) is defined, as usual, as the sequence of states \( K_1, K_2, \ldots, K_l \) obtained from the initial state \( K = K_1 \) by the consequent implementation of operators in \( \Sigma \). Namely, if \( \Sigma \) is an elementary PRA-scheme then its process of working consists of two states (initial and final). If in a scheme having the form (3.2) an operator \( \Omega_i \) which is to be implemented is a logical operator having the form (3.1), then this operator prescribes to pass to the working of the operator \( \Omega_r \) or \( \Omega_s \) in the cases when the values of \( x_i \) and \( x_j \) are, correspondingly, equal or not; if \( r > m \) (or \( s > m \)) then the corresponding passing is interpreted as the finishing of the working of the scheme (3.2). The working of a PRA-scheme described by expression (3) is defined as follows: the implementation of the scheme \( \Omega \) is repeated \( w \) times, where \( w \) is the value of the variable \( x_j \) in the state of memory in the process of working arising at the beginning of the working of our scheme; before every implementation of \( \Omega \) the variable \( x_i \) obtains the values, correspondingly, 0, 1, \ldots, \((w - 1)\). If \( w = 0 \) then the scheme \( \Omega \) does not work and only the operator \( x_i := 0 \) is implemented.

It is easily seen that the process of working of any PRA-scheme \( \Sigma \), originating from any memory state \( K \), is finite. The memory transformation \( \Phi \) defined by \( \Sigma \) is given as follows: if \( K \) is any state of memory, and \( K_1, K_2, \ldots, K_l \) is the process of working of \( \Sigma \) originating from the initial state \( K_1 = K \), then \( \Phi(K) \) is the final state \( K_l \) in
this process. Clearly, every transformation of memory defined by any PRA-scheme, is total.

The class of functions computable by PRA-schemes is defined as follows: If $\Psi$ is an arithmetical function, and $\Sigma$ is a PRA-scheme, then we say that the scheme $\Sigma$ computes the function $\Psi$ if the following condition holds: either $\Psi$ is equal to some component $\omega$ of the memory transformation $\Phi$ defined by $\Sigma$, or $\Psi$ can be obtained by a substitution of zeros for some variables in a component $\omega$ of $\Phi$. An arithmetical function is said to be PRA-computable if there exists a PRA-scheme computing $\Psi$.

Let us note that the given definition corresponds to the usual procedure of computing the values of functions by use of program schemes when we mark in the memory the so-called “input variables” and an “output variable”.

The notion of PRAU-scheme on a memory $(x_1, x_2, \ldots, x_n)$ is defined similarly to the definition of PRA-scheme with the only difference that the inequalities $r > t$ and $s > t$ in the point (2) of this definition are replaced by $r \geq t$ and $s \geq t$. Clearly, every PRA-scheme on $(x_1, x_2, \ldots, x_n)$ is a PRAU-scheme on the same memory. The process of working of PRAU-scheme is defined similarly to the process of working of PRA-scheme with the following differences. If in the scheme having the form (3.2) the operator $\Omega_t$ which is to be implemented has the form (3.1), where $r = t$ or $s = t$, then the operator $\Omega_t$ in the corresponding cases prescribes to return infinitely to the implementation of the operator $\Omega_t$. In such cases the process of working is considered as an infinite sequence of states.

The transformation of memory $\Phi$ given by the PRAU-scheme $\Sigma$ is defined for a state $K$ only in the case when the process of working originating from $K$ is finite; otherwise the state $\Phi(K)$ is admitted to be undefined (as well as its components). Other definitions (in particular, the definition of PRAU-computable function) are given in the same form as the corresponding definitions for the PRA-scheme. Clearly, PRAU-computable functions are in general not total. For example, everywhere undefined function $U(x)$ is computable by the following PRAU-scheme on the memory $(x_1)$:

\[
\text{begin } \text{if } x_1 = x_1 \text{ then go to 1 else go to 1 end.}
\]

**Theorem 3.1.** Any arithmetical function is PRA-computable (correspondingly, PRAU-computable) if and only if it is a PRF (correspondingly, GPRF).

The statement concerning PRA-schemes is proved as in [2]. The statement concerning PRAU-schemes is proved in an analogous way. The implementation of logical operators is described using the operation of branching of functions and Lemma 2.2. If in the PRAU-scheme (3.2) some logical operator $\Omega_t$ has the form (3.1), where $t = r$ or $s$, then everywhere undefined function $U(x)$ is taken as an element of the mentioned branching. The remaining details of the proof are similar to the considerations in [2].

**Acknowledgements**

The author is grateful to the referees of this paper for valuable notes and suggestions, and to P. Cegielski, G.E. Mints and A.P. Beltiuokov for the help in the formulation of the final version of this article.


References


