Flexible Arithmetic for Huge Numbers with Recursive Series of Operations

Vagan Y. Terziyan*, Alexey V. Tsymbal*, Seppo Puuronen**

Department of Artificial Intelligence and Information Systems, Technical University of Radioelectronics, Kharkov, Ukraine, e-mail: vagan@jytko.jyu.fi

*** Department of Computer Science and Information Systems, University of Jyvaskyla, Finland, e-mail: sepi@jytko.jyu.fi*

Abstract. In this paper a recursive expansion of the set of ordinary arithmetical operations is investigated. The addition is considered to be a base of recursion. Next we have multiplication, rising to a power, and so on, up to the infinity. Algebra is considered based on the set of recursive operations. The variable arithmetical operation $a \overset{n}{+} b$ is defined, where n is the level of recursion starting with ordinary + ($n=1$). The same arithmetical expressions can be interpreted in a various way accordingly to values of some variables. One can use such expressions to build more flexible mathematical models in which not only parameters are able to change but also relationship between parameters (model's structure). Basic properties of recursive operations are investigated, an algorithm of calculation of expressions of this algebra is considered. The recursive counters' apparatus is proposed to be used to represent huge integers, which are the results of recursive operations, in restricted memory. Method based on recursive calculation of a number of digits in the number being represented. The numbers' coding tool, offered in this paper, allows acquiring essential part of information included to a huge number without necessity of essential computer resources.

1 Introduction

A recursive continuation of the series of binary arithmetic operations already was considered in mathematics. In [1, 9, 11] the Akkerman's recursive function was discussed. This function is an infinite continuation of the series of arithmetic operations. Setting one parameter of this function, we receive function, realizing some operation from this series of arithmetic operations. For example, setting this parameter as 1, we receive usual addition, as 2 - multiplication and etc.. The Akkerman's function was considered from the point of view of the theory of recursive functions. This function is a vivid example of recursion of second degree (two variables are used to decrease the level of the recursion).

Because of some reasons the research in this area had fail. One reason is that the results of operations from the infinite series are huge numbers which can not be located not only on a paper, but even in a computer memory. One of founders of the theory of recursive functions, Hungarian mathematician R. Peter, said: "... the number 9^{9^9} is so huge, that for only its writing it would be necessary a piece of paper with length 1800 kilometers (if each figure would have half-centimeter width). There would be not enough a human life for the exact calculation of the value of 9^{9^9} " [11].

Another reason is that the algorithm, calculating values of operations of the infinite series is very labor consuming, since it realizes a recursion of second degree. There are not enough resources of the most modern computer for such calculations. The third reason is the difficulties to find any practical applications for such apparatus.

The problems of recursion in arithmetic are still under interest [5] as well as hierarchical structures in arithmetic [8]. Research in artificial intelligence needs reconfigurable arithmetic [12] to realize artificial neural computation. Results in area of flexible modeling using reconfigurable structure [4] make certain contribution to the discussion around Godel's Theorem and information [3]. Research in fuzzy arithmetic technique [2] and in interval arithmetic [6] is also accepted to various computer science applications.

In this paper a recursive expansion of the set of ordinary arithmetical operations is investigated. The addition is considered to be a base of recursion. Next we have multiplication, rising to a power, and so on, up to the infinity. An algebra is considered based on the set of recursive operations. The variable arithmetical operation $a\overset{n}{+}b$ is defined, where n is the level of recursion starting with ordinary + (*n=1*). The same arithmetical expressions can be interpreted in a various way accordingly to values of some variables. One can use such expressions to build more flexible mathematical models in which not only parameters are able to change but also relationship between parameters (model's structure). Basic properties of recursive operations are investigated, an algorithm of calculation of expressions of this algebra is considered. The recursive counters' apparatus is proposed to be used to represent huge integers, which are the results of recursive operations, in restricted memory. Method based on recursive calculation of a number of digits in the number being represented. The numbers' coding tool offered in this paper allows to acquire essential part of information included to a huge number without necessity of essential computer resources.

2 The Infinite Series of Arithmetical Operations

In this chapter, we consider the definition and main equalities of recursive arithmetical operations, which are used as a basis for Algebra of recursive arithmetical operations over natural numbers. Also we present the definition of the reverse and variable recursive arithmetical operations.

2.1 The Definition of Recursive Arithmetical Operations

We shall consider the traditional definitions of arithmetical operations over natural numbers:

1. Addition: $a+b$ (we use as basic operation);
2. Multiplication: $a*b=\underbrace{a+a+...+a}_{b}$;
3. Rising to a power: $a^b=\underbrace{a*a*...*a}_{b}$.

Notice, that the functions, which realize considered operations are recursive. We shall designate these functions φ_1, φ_2 and φ_3 accordingly:

$$\varphi_2(a,b) = \varphi_1(\varphi_2(a,b-1),a); \varphi_2(a,1) = a;$$
$$\varphi_3(a,b) = \varphi_2(\varphi_3(a,b-1),a); \varphi_3(a,1) = a. \qquad (1)$$

According to the theory of recursive functions [1], the functions, realizing the arithmetic operations are primitively recursive. Let us continue the recursive series of arithmetical operations up to the infinity:

$$\varphi_n(a,b) = \varphi_{n-1}(\varphi_n(a,b-1),a); \varphi_n(a,1) = a. \qquad (2)$$

These functions grow extremely fast. The number $\varphi_4(4,4) = ((4^4)^4)^4$ consists of 39 figures; $\varphi_4(5,5)$ - already of 437.

It is assumed that each operation from the given series has appropriate inverse operation ($\overline{\varphi}_1$ - subtraction, $\overline{\varphi}_2$ - division, $\overline{\varphi}_3$ - taking a root and etc...).

According to the algebra theory notation [7], we shall consider algebra

A'= {N, S},

where N - is the set of natural numbers, S - is the set of operations, defined upon N.

$$S = \{\varphi_1, \varphi_2, ..., \overline{\varphi}_1, \overline{\varphi}_2, ...\}.$$

We shall name the algebra A' as the algebra of recursive arithmetical operations. Each operation of this algebra is defined on natural numbers and can be obtained recursively from the previous operation.

We shall use the following symbols to denote the operations of the algebra A':

$$\varphi_1 - \overset{1}{+}, \varphi_2 - \overset{2}{+}, ... ; \overline{\varphi}_1 - \overset{1}{-}, \overline{\varphi}_2 - \overset{2}{-}, ... \qquad (3)$$

That is $a \overset{1}{+} b = a + b$; $a \overset{1}{-} b = a - b$; $a \overset{2}{+} b = a * b$, and so forth.

Using the expression (2) and notation (3) one can write the equality, allowing to calculate recursively the values of expressions of the algebra A':

$$a \overset{n}{+} b = (a \overset{n}{+} (b-1)) \overset{n-1}{+} a; \quad a \overset{n}{+} 1 = a; \quad a \overset{1}{+} b = a + b. \qquad (4)$$

From (4), recursively determining expression in brackets, we receive the basic equality of the algebra A' (5):

$$a \overset{n}{+} b = \underbrace{a \overset{n-1}{+} a \overset{n-1}{+} ... \overset{n-1}{+} a}_{b\ operands}; \quad a \overset{1}{+} b = a + b. \qquad (5)$$

The function (5) contains the recursion of the second degree (recursion is based on two variables *n* and *b*). The classic theory of recursive functions is not offering any suitable tool for non-recursive calculation of such expressions. We shall consider an iterative non-recursive algorithm in this paper to calculate expressions like (5).

2.2 The Inverse Recursive Arithmetical Operations

The inverse operations ($\overline{\varphi}_1, \overline{\varphi}_2, ...$) from the set S of the algebra A' can be defined by using the μ - operator [9]. This operator is named also as the limited operator of minimization or the operator of inversion. The definition of the operation was given in [9] as follows:

$$[z/x] = \mu y_{y\le z}(x(y+1) > z) \qquad (6)$$

The expression (6) can be interpreted in a following way: " The integer part from the division z by x is equal to the least y, $y \le z$ such, that $x(y+1) > z$ ". We shall generalize this formula for the recursive arithmetical operations:

$$\left[z \overset{n}{-} x\right] = \mu y_{y\le z}(y \overset{n}{+} x \ge z) \qquad (7)$$

The expression (7) defines the inverse operations of the algebra A'. It may be interpreted in a following way " The integer part from $z \overset{n}{-} x$ is equal to the least y, $y \le z$ such, that $y \overset{n}{+} x \ge z$".

2.3 The Variable Recursive Operations

The definition of the infinite series of recursive arithmetical operations allows to consider the operation as variable value, which is equal to an integer result of some other operation. Thus, one can build the multilevel structures of arithmetical equations, where the results of operations at the certain level determine which operations are valid at the previous level. The example of a multilevel arithmetical equation with variable recursive arithmetical operations is presented as follows:

$$f = a \overset{k \overset{t\overset{4}{+}1}{+} l}{+} b \overset{l \overset{t\overset{3}{+}2}{+} r}{+} c.$$

The multilevel structure of equations with recursive arithmetical operations allows to build more flexible mathematical models in which not only parameters are able to change but also relationship between parameters (model's structure).

3 Properties of Recursive Arithmetical Operations

We shall consider known to the present moment equalities of the algebra of arithmetic operations. We used definitions of recursive operations both in (4) and (5) form.

Property 1: $$a \overset{n}{+} (b+c) = (a \overset{n}{+} b) \overset{n-1}{+} \underbrace{a \overset{n-1}{+} \dots \overset{n-1}{+} a}_{c\ operands}.$$

Proof: $$a \overset{n}{+} (b+c) = \underbrace{a \overset{n-1}{+} a \overset{n-1}{+} \dots \overset{n-1}{+} a}_{b+c\ operands} = (\underbrace{a \overset{n-1}{+} \dots \overset{n-1}{+} a}_{b\ operands}) \overset{n-1}{+} \underbrace{a \overset{n-1}{+} \dots \overset{n-1}{+} a}_{c\ operands} =$$

$$= (a \overset{n}{+} b) \overset{n-1}{+} \underbrace{a \overset{n-1}{+} \dots \overset{n-1}{+} a}_{c\ operands}. \ \blacksquare$$

Property 2: $$a \overset{4}{+} b = a \overset{3}{+} (a \overset{3}{+} (b-1)).$$

Proof:

$$\overset{4}{a+b} = \underbrace{a\overset{3}{+}a\overset{3}{+}...\overset{3}{+}a}_{b\ operands} = a\overset{3}{+}(\underbrace{a*a*...*a}_{b-1\ operands}) = a\overset{3}{+}(a\overset{3}{+}(b-1)).\ \blacksquare$$

The proof was based on following property of the rising to a power: $a\overset{3}{+}b\overset{3}{+}c = a\overset{3}{+}(b*c)$. The property (2) allows speeding up a procedure of calculation of the recursive arithmetical operations.

Equalities with constants:

Property 3: $a\overset{n}{+}1 = a$;

Property 4: $a\overset{n}{+}2 = a\overset{n-1}{+}a$;

Property 5: $2\overset{n}{+}2 = 4$;

Proof: $2\overset{n}{+}2 = 2\overset{n-1}{+}2 = ... = 2*2 = 2+2 = 4$. ■

Theorem 1: The result of any recursive operation $a\overset{n}{+}b$, where $n > 3$ can be represented as rising of number *a* to some power *c*:

$$\forall a,b,n \in N, n > 3, \exists c \in N:\ a\overset{n}{+}b = a^c.$$

Proof: We shall use the method of mathematical induction.

1. When $n=4$: $a\overset{4}{+}b = a\overset{3}{+}(a\overset{3}{+}(b-1))$, (*Property 2*). I.e. $c = a\overset{3}{+}(b-1)$.
2. Let the theorem is valid when $n=k$:

$a\overset{k}{+}b = a^c$, where c - some natural number. (*)

3. We shall prove for $a\overset{k+1}{+}b$:

$$a\overset{k+1}{+}b = \underbrace{a\overset{k}{+}a\overset{k}{+}...\overset{k}{+}a}_{b\ operands}$$ (the basic equality of algebra A').

Applying serially to each operation $\overset{k}{+}$ the assumption (*), we receive:

$$a\overset{k+1}{+}b = a^{c_1}\overset{k}{+}\underbrace{a\overset{k}{+}a\overset{k}{+}...\overset{k}{+}a}_{b-2\ operands} = (a^{c_1})^{c_2}\overset{k}{+}\underbrace{a\overset{k}{+}a\overset{k}{+}...\overset{k}{+}a}_{b-3\ operands} =$$

$$= (((a^{c_1})^{c_2})^{\cdots})^{c_{b-1}} = a^{c_1*c_2*...*c_{b-1}}.$$

That is $a\overset{k+1}{+}b = a^c$, where $c = c_1*c_2*...*c_{b-1}$. ■

It is necessary to note, that *the results of recursive arithmetical operations can not be presented as sequence of operations higher than rising to a power:*

That is expression $\forall a,b,n,k \in N, k > 3, n > k, \exists c \in N:\ a\overset{n}{+}b = a\overset{k}{+}c$ is not true. This can be confirmed by counter examples. The last expression is valid only when *k*=1÷3 (for *k*=3 it was shown in the *Theorem 1*, for *k*=2 - further in the *Corollary 1*, for *k*=1 it is evident).

Corollary 1: The result of any operation $a\overset{n}{+}b$, where $n>2$, can be represented as product of number *a* with some natural number *d*:

$$\forall a,b,n\in N, n>2, \exists d\in N:\ a\overset{n}{+}b=a*d\,.$$

Proof: From the *Theorem 1*: $a\overset{n}{+}b=a^c$.

$$a^c=a*a^{c-1},\ \text{i.e.}\ a\overset{n}{+}b=a*d,\ \text{where}\ d=a^{c-1}.\blacksquare$$

Corollary 2: If a - some degree of number *m* $(a=m^{d_1})$, then $a\overset{n}{+}b$, where $n>2$ - some degree of number *m* also: $a\overset{n}{+}b=m^{d_2}$, where $d_2\geq d_1$.

Proof: From the *Theorem 1:* $a\overset{n}{+}b=a\overset{3}{+}c=(m^{d_1})^c=m^{d_1*c}$.

Thus $a\overset{n}{+}b=m^{d_1*c}$.■

Corollary 3: If *a* is divisible by some number *c* $(a=k_1c)$, then $a\overset{n}{+}b$, where $n>1$ is also divisible by the number *c:* $a\overset{n}{+}b=k_2c$.

Proof: From the corollary 1: $a\overset{n}{+}b=a*d=k_1*c*d=(k_1*d)*c$.■

Corollary 4: The result of operation $a\overset{n}{+}b$, where $n>2$ in *a*-th numeration looks as follows: $100\,...0_a$ (one and some zero).

Proof: When *n=3* (rising to a power) it is evident: $a^b=100...0_a$ (*b* zero).

When *n> 3* from the *Theorem 1*: $\exists c\in N:\ a\overset{n}{+}b=a^c$. $a\overset{n}{+}b=100...0_a$ (c zero). ■

Corollary 5: The number $a\overset{n}{+}b, n>2$ in a simple multiplier factorization contains the same elements, as the number *a*. It directly follows from the *Theorem 1.* ■

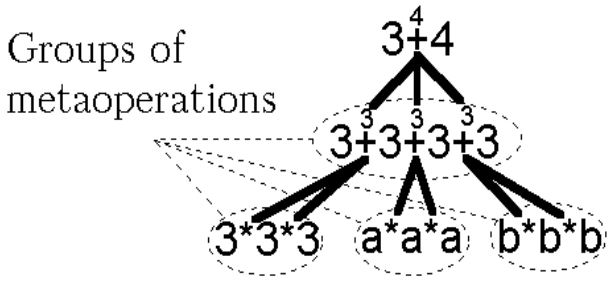
4 Calculation of Recursive Arithmetical Operations

The elementary way of realization of introduced operations will be programming of a recursive procedure, evaluating the expression (5) (the basic equality of the considered algebra). Despite the simplicity of program realization, such procedure appears extremely inefficient and occupies many resources of a computer (memory, time). For example, for the calculation of the expression $5\overset{5}{+}5$ on the computer Intel Pentium, 66 MHz, it was necessary to wait the result more than an hour. Even using the formula of the fast rising to a power (8), fast multiplication and some properties of the introduced arithmetic operations (*Properties 2,4*), such algorithm allows reducing only few resources of a computer.

$$a\overset{3}{+}(2*b)=(a\overset{3}{+}b)\overset{3}{+}2;\quad a\overset{3}{+}(2*b+1)=((a\overset{3}{+}b)\overset{3}{+}2)*a\,. \qquad (8)$$

That is why we offer non-recursive algorithm of calculation and the apparatus of recursive counters for evaluating of results with some precision.

Iterated algorithm allows to avoid unnecessary recursive calls of functions and, accordingly, to avoid overflow of the stack. The algorithm makes decomposition of operation $a\overset{n}{+}b$ to a sequence of operations of a lower level, $\overset{m}{+}$, $m < n$. For example, with the help of this algorithm it is possible to calculate the result of $a\overset{n}{+}b$ by a sequence of operations of rising to a power only, or multiplication only, or any other known arithmetical operation. The algorithm is based on regularity, which is inherent to any recursive process of calculation of operations considered. Such regularity it is possible to see in the example presented in Figure 1.



$$a = 3\overset{3}{+}3,\ b = 3\overset{3}{+}3\overset{3}{+}3.$$

Fig. 1. Example of calculation of the value $3\overset{4}{+}4$

At each level, the calculation represents a sequence of groups of operations with identical operands. As such group, in the example, we use expression $a\overset{m}{+}a\overset{m}{+}\ldots\overset{m}{+}a,\ a,m \in N$. We shall name an element of a group *a* as a multiplier, and the number of operations in a group - size of a group.

The algorithm is based on the following two rules:

1. At the end of calculation of the result for some group, the multiplier for the following group at this level becomes equal to the result of previous group.

2. If the calculation of the group *g* from the level *h* has been completed and some group from the level *h+1* simultaneously comes to the end, then the size of the group *g+1* at the level *h* becomes equal to the result of the group's *g* calculation minus one.

We shall consider the text of algorithm in the language, being simplified Pascal:

```
1. Input (a, b, n, m) (a, b, n, m $\in$ N; a, b, n>1; m<n-1).
2. $C1_1$:=b-1; $C2_1$:=b-1; $C1_2$:=a-1; $C2_2$:=a-1;... ;$C1_{n-m-1}$:=a-1; $C2_{n-m-1}$:=a-1;
3. Result:=a; GroupSize:=a-1;
4. Factor:=Result; Operations:=GroupSize;
5. Result:=Result$\overset{m}{+}$Factor; Operations:=Operations-1; if Operations>0 then goto 5;
6. i:=n-m-1;
7. if i=0 then Output (Result), End; $C1_i$:=$C1_{i-1}$; if $C1_i$=0 then i:=i-1, goto 7;
8. if i=n-m-1 then goto 4; GroupSize:=Result-1; i:=i+1; $C1_i$:=$C2_i$;
9. i:=i+1; if i=n-m goto 4; $C1_i$:=Result-1; $C2_i$:=Result-1; goto 9.
```

Variables, used in the text of the algorithm:

- a, b - operands of the evaluated operation;
- n - the operation calculated $\overset{n}{+}$;
- m - the operation $\overset{m}{+}, m < n$, which is used to carry out the calculation;
- C1 [1.. n-m-1] - array, that contains the current sizes of groups at levels above m;
- C2 [1.. n-m-1] - array, containing a copy of the array C1 for the subsequent restoration;
- Result - variable, in which the result of the calculation is formed;
- GroupSize - size of a current group at the level m;
- Factor - operand of current group;
- Operations - number of non-calculated operations in current group;
- i - auxiliary index variable.

We shall describe actions, made in the algorithm:

1 - input of the initial data (operands, operations);
2 - initialization of arrays - counters of the groups' sizes at each level;
3,4 - initialization of variables;
5 - calculation of the next group;
6,7 - decrement of the sizes of groups after making an operation;
8 - change of the size of current group (rule 2);
8,9 - restoration and change of the sizes of current groups at all levels.

The non-recursive algorithm for evaluating the results of the recursive arithmetical operations of the algebra A' was realized on a computer and checked experimentally. The algorithm allows refusing labor-consuming recursive algorithm. One of problems remains: results of most recursive operations grow quickly and are not located in memory. The further research is necessary to find out completely new properties of the infinite series of operations, that would essentially effect to the resources of calculation.

5 The Recursive Counters as a Tool for Evaluation of Huge Natural Numbers

In this chapter, the recursive counter apparatus is proposed to be used to represent huge integers in restricted memory. This representation is recursive with partial loss of information about number (losing of accuracy, and getting an advantage with an amount of a computer memory required). The method is based on recursive calculation of a number of digits in the number being represented. Representation consists of the number of recursion's steps, which are necessary to reduce initial huge integer so that it becomes less than certain value. Representation also contains the result obtained at the last step of recursive algorithm. The more long a number is the more steps of recursion are necessary to represent it. The certain loss of the information is expected for each level of recursion. However the rate of saved memory grows quite fast from level to level. Goals of such representation include possibility to proceed the recursive arithmetical operations.

5.1 The Basic Definitions and Equalities of the Recursive Counters' Apparatus

We shall refer to the recursive counter (here and after - *recounter*) of *n*-th level of some number as the number of figures in the recounter of *n-1*-th level of the number. Thus a recounter of *0*-th level - an initial number. Naturally, since a certain level *l*, all further recounters of number are equal to *1*. We shall name the number *l* as a *level of a number*. With the help of the level of a number it is possible to restore easily the interval, in limits of which this number can lay (to give an estimation of a number). In addition, if the initial number is rather great, the relative error of the number representation by the offered apparatus will be small enough. Thus we will distinguish the two concepts: the *level of a recounter* and the *level of a number*. The level of a recounter means the number of steps of recursive count of a quantity of the figures in a number. The level of a number means the value of the level of recounter in which recounter begins to be equal to one.

We will mark the operation of *taking the recounter of certain level* as an initial number with level of recounter in corner brackets:

n' *or* $n^{<1>}$ - recounter of the first level taken from the number *n*;

$n^{<m>}$ - recounter of the *m*-th level taken from the number *m*.

Then it is possible to write (square brackets denote the operation of taking the integer part from a number):

$$n' = \lfloor log_k\, n + 1 \rfloor = \lfloor log_k (n+1) \rfloor \qquad (9)$$

$$n^{<m>} = (n^{<m-1>})^{<1>} = \left\lfloor log_k (n^{<m-1>}) + 1 \right\rfloor = \left\lfloor log_k (n^{<m-1>} + 1) \right\rfloor, \qquad (10)$$

where *k* is the base of the numeration system.

The expressions (9) and (10) allow to calculate recursively the value of recounter of any level in a numeration system with the base *k*. The level of a number *n* we shall designate as *ord(n)*. Then it is possible to write:

$$ord(n) = \mu m (n^{<m>} = 1) \qquad (11)$$

The expression (11) defines the level of number with the help of the μ - operator (the limited operator of minimization or the operator of inversion). So, *ord(n)* is equal to the least natural *m* such, that $n^{<m>} = 1$. From (11) it follows, that the figures have the first level, numbers $10 \div 999999999$ - the second, and etc.

It is offered to code numbers with the following pair:

$$\{m, n^{<m-1>}\}, \qquad (12)$$

where *m = ord(n),* or *m* - is the level of a number *n,* $n^{<m-1>}$ is the last recounter which is not equal to 1.

For example, one of the huge results of forth-level recursive operation can be coded as follows: $8 \overset{4}{+} 9 = \{3,8\}$. It means that number of figures in the number of figures in the result is equal to 8.

In the pair (12) $n^{<m-1>} = 2 \div 9$, since by the definition of a level of a number $n^{<m>} = 1$. We shall show numerical intervals, which correspond to one coding pair:

$$\{1, n\} = n(digits\ 2 \div 9);\ \{2,2\} = 10 \div 99;\ \{2,3\} = 100 \div 999; ...$$

$$\ldots;\{2,9\}=100000000\ \div 999999999\ ;\ \{3,2\}=1\underbrace{0\ldots0}_{9}\div\underbrace{9\ldots9}_{99}; \qquad (13)$$

$$\{3,3\}=1\underbrace{0\ldots0}_{99}\div\underbrace{9\ldots9}_{999};\ \ldots;\ \{3,9\}=1\underbrace{0\ldots0}_{10^8-1}\div\underbrace{9\ldots9}_{10^9-1};\ldots$$

The expression (13) contains the intervals of numbers, coded by one pair with the level of numbers from 1 up to 3. Let $\min\{m,n\}$ and $\max\{m,n\}$ - accordingly the minimum and the maximum numbers of an interval, coded by such pair. They can be determined recursively from the expression (14):

$$\begin{aligned}&\min\{2,n\}=10^{n-1};\max\{2,n\}=10^{n}-1;\\&\min\{m,n\}=10^{\min\{m-1,n\}-1};\max\{m,n\}=10^{\max\{m-1,n\}}-1.\end{aligned} \qquad (14)$$

The formula (14) can be used for coding of natural numbers only in the decimal numeration system. We can generalize its to use for coding in any numeration system with the base *k* (15) (*n* can vary from 2 to *k-1*):

$$\begin{aligned}&\min\{2,n\}=k^{n-1};\max\{2,n\}=k^{n}-1;\\&\min\{m,n\}=k^{\min\{m-1,n\}-1};\max\{m,n\}=k^{\max\{m-1,n\}}-1.\end{aligned} \qquad (15)$$

When we realize the coding system (12) in a computer we have to restrict the maximal value of a level of a number in order to have an ability to place it in restricted volume of a computer memory. Let m* denote the maximal value of a level of a number. Then we shall need $\lceil \log_2 m^* \rceil + \lceil \log_2 (k-2) \rceil$ bits to store one pair (12) that codes some number *n* in a computer memory, where *k* denotes the base of the used numeration system. The number *n* can vary in such case from 2 to $\max\{m^*, k-1\}$. Consequently with the pair (12) when level of number is restricted by m^* $(m \le m^*)$, we can code any natural number:

$$n = 2 \div \max\{m^*, k-1\} \text{in} \lceil \log_2 m^* \rceil + \lceil \log_2 (k-2) \rceil \text{bits.}$$

In order to evaluate in some way the quality of the coding by the pair (12) we will introduce the operator Q[m,n], which will help to determine the quality of the coding by the pair $\{m,n\}$. The quality Q[m,n] in our case is equal to the coefficient of information advantage *i* in comparison with usual representation of numbers divided by the coefficient of relative error of the coding δ(16).

$$Q[m,n]=\frac{i[m,n]}{\delta[m,n]} \qquad (16)$$

Here $$i[m,n]=\frac{\lfloor \log_2(\max\{m,n\})+1 \rfloor}{\lceil \log_2 m^* \rceil + \lceil \log_2 (k-2) \rceil} \qquad (17)$$

(17) - is the ratio of the number of bits needed for ordinary coding to the number of bits is needed for proposed coding.

$$\delta[m,n] = \frac{\max\{m,n\} - \mu\{m,n\}}{\max\{m,n\}} = \frac{\max\{m,n\} - \frac{\max\{m,n\} + \min\{m,n\}}{2}}{\max\{m,n\}} = \qquad (18)$$

$$= \frac{\max\{m,n\} - \min\{m,n\}}{2 * \max\{m,n\}}, \mu\{m,n\}\text{-the average of the coded interval.}$$

Value δ[*m,n*] is a relative error that is equal to the maximal absolute error of the coding by the pair $\{m, n\}$ divided by maximal number in this interval.

Let us determine Q[*2,n*] and Q[*3,n*] in order to see how the quality will change from the second level of numbers to the third.

$$Q[2,\mathrm{n}] = \frac{\lfloor n \log_2 10 \rfloor}{\lceil \log_2 m^* \rceil + \lceil \log_2 (k-2) \rceil} \cdot \frac{2 \cdot (10 - \frac{1}{10^{n-1}})}{9 - \frac{1}{10^{n-1}}} \approx q_2 \cdot n, (q_2 = Const). \qquad (19)$$

$$Q[3,n] \approx \frac{\lfloor (10^n - 1) \log_2 10 \rfloor}{\lceil \log_2 m^* \rceil + \lceil \log_2 (k-2) \rceil} * 2 \approx q_3 \cdot 10^n, (q_3 = Const). \qquad (20)$$

One can see that we have a recursive power growth of function Q[m,n]. It is easy to show that $Q[4,n] \approx q_4 * 10^{(10^n)}; Q[5,n] \approx q_5 * 10^{(10^{(10^n)})}$ and so on, where q_4 and q_5 - some constants. So, we always have almost linear graphic of function Q[m,n] with m-2 - logarithmic scale of vertical axis. For example, graphic Q[6,n] will appear linear with the twice-logarithmic scale of vertical axis. With the growth of a level of a number in a pair (12) the function Q[*m,n*] also grows. It means that the size of a number grows faster than the size of coding interval grows. It means also that the relative error grows much slower than the size of a number grows.

Naturally, the coding with the help of the pair (12) is not a universal tool of representation of any numbers right up to infinity in a limited volume of computer memory. Since certain number *n*, its level in the coding pair will exceed an allowable limit of an occupied volume of memory. It is difficult even to imagine such number. However if such number will appear in some problem, it is possible to code the level of a number by the same tool, which we use to code number: to consider the level of the second degree of a number: $ord_2(n) = ord(ord(n))$. If it is not enough, then one can use the level of the third, fourth degree,..., level of degree of number level, and etc.

5.2 Operations Over Recursive Counters

The realization of operations over numbers, represented by recounters, enables to estimate the results of arithmetic expressions containing operations over recounters. The realization of operations over recounters is complicated by ambiguity (loss of the information) of recounters. We shall consider it by giving an example. Let *a'*=2, and *b'*=1. We can not unequivocally specify result of expression (*a***b*)*'*: (*a***b*)*'*=*a'*+*b'* or *a'*+*b'*-*1* (3 or 2 in our case). We can estimate the lower or the upper boundary of calculated expression. We shall consider possible outcomes of taking of recounter from a result of operation of multiplication:

$$(n_1 * n_2)' = \begin{cases} n_1' + n_2' \\ n_1' + n_2' - 1 \end{cases}; (n_1 * n_2)^{<k>} = \begin{cases} \max(n_1^{<k>}, n_2^{<k>}) \\ \max(n_1^{<k>}, n_2^{<k>}) + 1 \end{cases}, \text{where } k > 1. \quad (21)$$

If $n_1 \geq n_2$, then we obtain: $(n_1 * n_2)^{<k>} = \begin{cases} n_1^{<k>} \\ n_1^{<k>} + 1 \end{cases}, \text{where } k > 1.$ (22)

The expressions (21) and (22) contain possible outcomes of the multiplication of numbers, presented by its recounters of various levels. The similar equality can be written down for other operations, for example, for addition (23), (24):

$$(n_1 + n_2)^{<k>} = \begin{cases} \max(n_1^{<k>}, n_2^{<k>}) \\ \max(n_1^{<k>}, n_2^{<k>}) + 1 \end{cases}, \text{where } k > 0. \quad (23)$$

If $n_1 \geq n_2$, then we obtain: $(n_1 + n_2)^{<k>} = \begin{cases} n_1^{<k>} \\ n_1^{<k>} + 1 \end{cases}, \text{where } k > 0.$ (24)

Knowing possible outcomes of operations over recounters (21) - (24) we can estimate the lower and the upper boundaries of a result of any arithmetical expression with such operations over huge numbers. Further it is necessary to simulate distribution with the specified probabilities for more exact evaluation. The more the volume of calculations there will be, the more precise result will be received. It can be explained so: under identical probability of appearance of each of figures in the categories of number, the probability of each of possible outcomes remains constant. The probability of each of outcomes can be calculated by two ways: *statistical* or *analytical*. Statistical - on great volume of calculations, when identical probability of occurrence of each figure of a number is reached, the probability is equal to ratio of number of occurrence of researched outcome to common number of tests. Analytical - with use of mathematical calculations, based on fact that probabilities of occurrence of each figure of number are identical, on theorems of the theory of probabilities. For example, probabilities to obtain exact recounter valuation of the result of multiplication are presented as follows:

$$(n_1 * n_2)' = \begin{cases} n_1' + n_2', \ p \approx 0.8268398, \\ n_1' + n_2' - 1, p \approx 0.1731602, \end{cases} \tag{25}$$

where p - probability of corresponding outcome. The amount of digits in a multiplication of two numbers depends only of the first digits of the numbers. For example, if the first digits are 4 and 3 (or if at least one of the digits is more) then we may say that we shall have the second outcome. If the first digits are 2 and 2 (or if at least one of the digits is less) - the first. If the first digits are 3,3 or 2,3- we have to check the second digits because in this case we may have both of the outcomes.

One important property of the apparatus of recounters is the opportunity to operate not with numbers, but with their recounters. Recounters allow simplifying considerably the processing of huge numbers, to lower considerably required resources. The work with huge numbers occurs on the following algorithm: if in result of any operation the recounter of the current *n*-th level exceeds a certain threshold, the initial number is coded by recounter of *n+1*-th level, and all operations will be carried out over this recounter. The apparatus of recounters is expected to be applied to such areas of computer science where it would be necessary to handle huge numbers.

5.3 A Relationship Between Recounters and Recursive Arithmetical Operations

We will prove a theorem which defines a relation between the concept of a recursive arithmetical operation and the concept of a recursive counter. The considered theorem allows to define a recounter of a certain level as a result of a recursive operation. The proof of the theorem is based on two facts (10) and (26).

The expression (10) allows to calculate a recounter of the level *l* of some number *n* in a numeration system with the base *k*. This expression is based on properties of positional numeration system [13].

$$\log(x \overset{5}{+} l) = \log((x \overset{5}{+} (l-1)) \overset{4}{+} x) = \log((x \overset{5}{+} (l-1))^{((x \overset{5}{+} (l-1))^{(x-1)})}) =$$
$$= (x \overset{5}{+} (l-1))^{(x-1)} \times \log(x \overset{5}{+} (l-1)) = (x \overset{5}{+} (l-1))^{(x-1)} \times (x \overset{5}{+} (l-2))^{(x-1)} \times \ldots \tag{26}$$
$$\ldots \times (x \overset{5}{+} 1)^{(x-1)} \times \log x, \ where \ l > 1.$$

The expression (26) determines the logarithm of the fifth recursive operation.

Now we can proof of the theorem. Words “(or + 1)”, written down after some equality mean here and further, that the expression can also be more on a unit.

Theorem:

$$(x \overset{5}{+} l)^{<z>} = \left[(x-1) \times \log(x \overset{5}{+} (l-z+2)) \right]' (or + 1), where \ z > 2, l \geq z.$$

Proof:

$$(x\overset{5}{+}l)^{<3>}=\left[\log(x\overset{5}{+}l)+1\right]^{<2>}=\left[(x\overset{5}{+}(l-1))^{x-1}\times\log(x\overset{5}{+}(l-1))+1\right]^{<2>}=$$

$$=\left[(x-1)\times\log(x\overset{5}{+}(l-1))+\log\log(x\overset{5}{+}(l-1))\right]'(or+1)=$$

$$=\left[(x-1)\times\log(x\overset{5}{+}(l-1))\right]'(or+1).$$

The last transformation is explained by the fact that the number of figures in the expression in brackets is equal to the number of figures in the first summand (or more on a unit of it).

Now we use the mathematical induction in a following way:

1. $$(x\overset{5}{+}l)^{<3>}=\left[(x-1)\times\log(x\overset{5}{+}(l-1))\right]'(or+1).$$

2. Let: $$(x\overset{5}{+}l)^{<n>}=\left[(x-1)\times\log(x\overset{5}{+}(l-n+2))\right]'(or+1).$$

3. Then: $$(x\overset{5}{+}l)^{<n+1>}=\left[(x-1)\times\log(x\overset{5}{+}(l-n+2))\right]^{<2>}(or+1)=$$

$$=\left[(x-1)\times(x\overset{5}{+}(l-n+1))^{x-1}\times\log(x\overset{5}{+}(l-n+1))\right]^{<2>}(or+1)=$$

$$=\left[\log(x-1)+(x-1)\times\log(x\overset{5}{+}(l-n+1))+\log\log(x\overset{5}{+}(l-n+1))\right]'(or+1)=$$

$$=\left[(x-1)\times\log(x\overset{5}{+}(l-n+1))\right]'(or+1).$$

The last transformation is explained by fact that the number of figures in the expression in brackets is equal to the number of figures in the second summand (or more on a unit of it). ■

Corollary: $(x\overset{5}{+}l)^{<l>}=\left[(x-1)\times x^{x-1}\times\log x\right]'\ (or+1),\ where\ \ l>2$.

Proof: Let in the theorem $z=l$. We receive:

$$(x\overset{5}{+}l)^{<l>}=\left[(x-1)\times\log(x\overset{5}{+}2)\right]'(or+1)=\left[(x-1)\times\log(x\overset{4}{+}x)\right]'(or+1)=$$

$$= \left[(x-1)\times\log(x^{(x^{x-1})})\right]^{,}(or+1) = \left[(x-1)\times x^{x-1}\times\log x\right]^{,}(or+1). \ \blacksquare$$

The corollary shows a close relation between the fifth recursive operation and the operation of taking a recounter. One can see that the simultaneous increasing of the second operand and the level of a recounter is not effecting the result. It shows that the operation of taking a recounter allows evaluating the results of a recursive arithmetical operation. In the Figure 2, the results of fifth recursive operation for *x = 3÷20*, calculated using the corollary, are presented.

x	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
y	1	3	4	5	6	8	9	10	12	13	15	17	18	20	21	23	25	27

Fig. 2. The values of function $y(x) = (x \overset{5}{+} l)^{<l>}$ with discrete *x = 3÷20*

6 Conclusion

In this paper, the mathematical apparatus, being integration of arithmetic, was offered. Expansion of the set of ordinary arithmetical operations is investigated with a goal to make a tool for representation of quite flexible mathematical expressions. Unfortunately, to create of the perfect mathematical apparatus on the basis of offered ideas and realize it on practice are difficult enough problems and need a lot of further research. Search of practical applications of the results, considered in this paper, is now going on. Philosophical aspects of the infinite series of recursive operations are under consideration, physical analogues are being studied.

The given work can be considered as an attempt to guess the future flexible programming in which the same arithmetical expressions can be interpreted in a various way accordingly to values of some variables. The multilevel structure of equations with recursive arithmetical operations one can use to build more flexible mathematical models in which not only parameters are able to change but also relationship between parameters (model's structure).

The large numbers that occur as the result of recursive operations can be handled using the apparatus of the recounters. The numbers' coding tool, offered in this paper, allows acquiring essential part of information included to a huge number without necessity of essential computer resources.

References

1. D.W. Barron: Recursive Techniques in Programming. London: MacDonald 1969.
2. R.A. Bielefeld: Estimation of HIV Infection and AIDS Onset Times Using Fuzzy Arithmetic Techniques. In: K.C. Lun, et al. (eds.): MEDINFO 92 (Part 2), Amsterdam: North-Holland 1992, pp.921-929.
3. G.J. Chaitin: Godels Theorem and Information. International Journal of Theoretical Physics 22, 941-954 (1982).

4. D. Dengler: Customized Plans Transmitted by Flexible Refinement. In: W. Wahlster (ed.): Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96), Chichester: Wiley 1996, pp.609-613.
5. L. Fribourg: Mixing List Recursion and Arithmetic. In: Proceedings of Seventh Annual IEEE Symposium on Logic in Computer Science. Santa Cruz, California: IEEE Computer Society Press 1992, pp.419-429.
6. P. Girodias, E. Cerny, W.J. Older: Solving Linear, Min and Max Constraint Systems Using CLP Based on Relational Interval Arithmetic. In: U. Montanari and F. Rossi (eds.): CP95 - Proceedings of the First International Conference on Principles and Practice of Constraint Programming. Lecture Notes in Computer Science 976. Berlin: Springer 1995, pp.186-203.
7. V.M. Glushkov, G.E. Tseytlin, E.A. Jyschenko: Algebra. Languages. Programming. Kiev: Scientific Thought 1974.
8. P. Hajek: Epistemic Entrenchment and Arithmetical Hierarchy. Artificial Intelligence 62 (1), 1993.
9. O.P. Kuznetsov, V.M. Adelson-Velsky: The Discrete Mathematics for an Engineer. Moscow: Energy 1980.
10. S. Majumdar: Application of Relational Interval Arithmetic in Performance Analysis of Computing Systems. In: Proceedings of the ILPS95 Post-Conference Workshop on Interval Constraints, 1995.
11. R. Peter: Game With Infinity. Moscow: Science 1967.
12. A. Strey, N. Avellana, R. Holgado, J.A. Fernandez, R. Capillas, E. Valderrama: A Massively Parallel Neurocomputer with a Reconfigurable Arithmetical Unit. In: J. Mira and F. Sandoval (eds.): From Natural to Artificial Neural Computation. Lecture Notes in Computer Science 930. Berlin: Springer 1995, pp.800-806.
13. J. Zhang, D.A. Norman: A Representational Analysis of Numeration Systems. Cognition 57, 271-295 (1995).