# A LANGUAGE USING QUANTIFIERS FOR DESCRIPTION OF ASSERTIONS ABOUT SOME NUMBER TOTAL PASCAL FUNCTIONS

**Nikolay Kosovskii**

***Abstract**: Mathematical models of N-finite ($N \geq 2$) (especially for $N= 2^{16}$ typical for IBM-compatible personal computers) real, integer or Boolean valued computations are proposed in the frameworks of a finite-discrete mathematics. Such a mathematical model more precisely characterizes the contemporary practice of the computer use. Notions of an N-finite RAM (Random Access Machine), an N-finite RASP (Random Access Machine with Stored Program) and an N-finite Pascal program using integer operations modulo N with the reminders from the segment $[-\lfloor N/2 \rfloor, \lfloor N/2 \rfloor - 1 + (N \bmod 2)]$ are introduced. A formal N-finite quantifier logic language of modulo N arithmetic operations is defined. It allows formulating mathematical properties of such a non-recursive total Pascal function or predicate. It is proved that the problem of a formula identical truth in such a language containing arithmetic operations modulo N is a P-SPACE-complete one. So, the proof of a correctness of a non-recursive total Pascal program may be simplified with the use of the proposed language.*

***Keywords:** P-SPACE-completeness, RAM, RASP, Pascal, correctness of a Pascal program.*

***ACM Classification Keywords:** F.2.m Analysis of Algorithms and Problem Complexity Miscellaneous.*

## Introduction

Every computer program is a mathematically precise one. But some assertions about a program run result may be rather ambiguous. A language for the mathematically precise notation of properties, conditions and assertions about some Pascal programs is proposed below.

A Pascal function is a total one if it halts and has a value for any input data. For example, a Pascal function is a total one if the following conditions are fulfilled:

- Every division of two numbers is a total one (i.e. before performing the division there is an auxiliary checking whether the divisor differs from zero and if the divisor is zero then the program gives some value for the result);
- The operator *goto* backward is forbidden;
- The loop header begins only with *for*;
- Every non-empty set of Pascal functions (even a set of one function) must not be mutually recursive one;
- Input statements and files are not used.

Let's consider a Pascal function as a total one. Even with such a restriction the first order logic language signature is not sufficient for formal description of properties of an array with integer indexes.

Let a Pascal program uses variables only of the types *real*, *integer, Boolean* and arrays of these types. The set of all numbers of these types usually contains only a finite amount of distinct numbers. For example, a maximal

amount of distinct numbers of the types *real* and *integer* for a personal computer is $2^{16}$ or $2^{32}$ or sometimes $2^{64}$. This illustrates that the use of the set of all integers as a possible set of input data is evidently excessive. Besides, it allows (as it is shown below) essentially to simplify the checking of identical truth for assertions about a Pascal program formulated by means of the proposed extension of the first order logic language.

Besides, a practical number computation mostly uses not integers with unbounded length of their notations but only integers from $[-2^{15}, 2^{15} - 1]$ (the computer type *integer*) or integers from $[-2^{31}, 2^{31} - 1]$ (the computer type *long integer*). And all IBM-compatible computer computations are fulfilled modulo $2^{16}$ or $2^{32}$ respectively. That is why traditional mathematical models of computation are not useful for detailed mathematical simulation of a practical personal computer number program.

To formulate mathematical properties of a program it is convenient to use the formal arithmetic language with additional total functions and total predicates of some finite signature. It is possible because every function and every predicate is a total one. At the same time the totality of a Pascal function is an important aspect of friendly software.

That is why the condition of a program totality is natural for the contemporary programmers. In particular, assertions about the total correctness of a program based on the Hoare triads [5, 7] for a Pascal total program may be formulated in such a language.

Below a notion of *N*-finite data of the types *real* and *integer* as well as a notion of the first order *N*-finite index formula are introduced to prove a P-SPACE-completeness of the problem of an *N*-finite index first order formula identical truth. This is impossible if an algorithm (a program) is used to process data with arbitrarily large integers, in particular, because of the use in the formulas quantifiers upon the set of all integers.

Hence, the proposed below mathematical models of computation and formal language for a condition or an assertion setting about the result of such a computation is more adequate to the contemporary practical computation by a program (for example, by a Pascal program) than the use of arbitrarily large numbers as input data.

## Initial definitions

A number of the type *integer* from the segment $[-\lfloor N/2 \rfloor, \lfloor N/2 \rfloor - 1 + (N \bmod 2)]$ is called here an *N-integer*. Here $\lfloor N/2 \rfloor$ denotes the maximal integer not greater than N/2 and (N mod 2) denotes the remainder of *N* divided by 2.

A floating-point representation of a number of the type *real* with the notation length not greater then $\lfloor \log_2 N \rfloor$ bits is called here an *N-real* one. The amount of such *N-real* numbers is not greater than *N*.

Pascal functions and Pascal predicates dealing only with numbers and variables of the types *N-real* and *N-integer* are called here *N*-finite ones.

In such a case the use of a traditional for the Pascal language [3] notation for the call of a Pascal function or a Pascal predicate inside an expression (below the term "*N*-finite index term in a finite signature" is used) does not require a more detailed definition than that of a Pascal expression computation. In particular, if an expression contains an *N-real* number then the value of this expression has the type *N-real*.

The notions of RAM (Random Access Machine) and RASP (Random Access Machine with Stored Program) are defined in [1] as models for a non-negative integer valued computation using the only one (actually dynamical) linear array for integers with unbounded notation lengths.

Let's define the notions of an ***N*-finite RAM** and an ***N*-finite RASP** using arithmetic operations modulo *N* with the reminders from the segment $[-\lfloor N/2 \rfloor, \lfloor N/2 \rfloor - 1 + (N \bmod 2)]$.

It is suggested to use the only one array of integers from this segment with arbitrary number of dimensions (one-dimensional, two-dimensional, ...) inside a program.

The number of elements in every dimension does not exceed *N*. The input data for an *N*-finite RAM and an *N*-finite RASP are situated in the several first dimensions but the *N*-finite RASP program (with the counter of commands) is situated in the last dimensions. Besides, the counter of commands is situated in the very last dimensions. The bound of an additional dimension uses the only one constant (for example, 1). This additional dimension may be used for the indication of the end of the input data (and the beginning of an *N*-finite RASP program as well as the beginning of the command counters).

So, the array for every *N*-finite RAM or *N*-finite RASP program contains the only finite number of elements. More exactly, this number must be bounded by $N$, $N^2$, ..., because of potentially increasing number of array dimensions both for *N*-finite RAM and *N*-finite RASP programs. The last models are the base for interpretation of an arbitrary *N*-finite Pascal function.

The notion of a formula of the first order many sorted logic language is not sufficient if we use a traditional notation for an array element in a Pascal language expression. That is why the notion of such a logic formula with the calls of *N*-finite number Pascal total functions and *N*-finite number Pascal total predicates of a finite signature is extended up to the notion of an *N*-finite index formula of a finite signature.

First of all, the notion of an ***N*-finite index term of a finite signature** is introduced. Its definition may be received from the definition of the *term of a finite signature* (see, for example, [6, 7]) by means of replacing the word "term" by the words "*N*-finite index term of a finite signature" and adding to the definition the following construction: a word of the form "array variable followed by an included into the square brackets sequence of *N*-finite index terms of a finite signature" is an *N*-finite index term of a finite signature. Every *N*-finite index term of this sequence must have the same enumerated type as it was defined in the description of the used array. Only *N*-finite Pascal total functions and arithmetic total operations with *N-reals* and *N-integers* as well as logical constants and operations are used in the new definition. As it is usual for the Pascal language an infix notation form for functions (for example +, *) and predicates (for example =, >= ) may be used.

A **constant *N*-finite index term of a finite signature** is an *N*-finite index term of a finite signature which does not contain occurrences of variables.

The definition of the notion of an ***N*-finite index atomic formula of a finite signature** may be received from the definition of the notion of a *many sorted (many typed) atomic formula of a finite signature* by means of replacing the word "term of a finite signature" by the words "*N*-finite index term of a finite signature". Only *N*-finite Pascal total functions and *N*-finite Pascal total predicates of the signature, *N-reals* and *N-integers* are used in the new definition.

The definition of the notion of an ***N*-finite index first order formula of a finite signature** may be received from the definition of the notion of a *many sorted (many typed) first order formula of a finite signature* by means of replacing the word "formula" by the words "*N*-finite index formula" and the addition to the definition the possibility of the using quantifiers upon array elements.

As it is usual for the Pascal language it is allowed to insert additional pairs of brackets into N-finite index term and into N-finite index formula in a finite signature.

It is allowed type coercion: *Boolean* into *N-integer* (*true* into 0 and *false* into –1) and *N-integer* into *N-real*.

Essentially, the notion of an objective variable used in the first order logic is extended up to an array element with constant indexes. The last ones may be situated immediately after a quantifier, have free or bound occurrences in an *N*-finite index first order formula of a finite signature. That is why it is needed to complete in a natural way the definition of a **quantifier scope** as such a minimal upon the length N-finite index sub-formula of a finite signature which begins with this occurrence of a quantifier.

An occurrence of an array element *t* with constant indexes is a **bounded occurrence** if it is in a scope of a quantifier immediately after which a name of array or sub-array (with indicating constant bounds of elements in every dimension) containing the element *t* is situated. An occurrence of an array element which is not a bounded one is called a **free** one.

Let's an *N*-finite Pascal total function or an *N*-finite Pascal total predicate uses only such types as *integer* or *real* or *Boolean* and arrays of these types.

The notion of **identically true *N*-finite index formula of a finite signature** (which consists of *N*-finite Pascal total functions and *N*-finite Pascal total predicates of *N-real* or *N-integer* or *Boolean* types) may be defined as usually.

## Correctness of an N-finite Pascal total function

Correctness of an *N*-finite Pascal total function body may be described by Hoare triads in the form *{A}S{B}*, where *A* and *B* are conditions, and *S* is a Pascal statement sequence [5, 7]. It may be written in the form $A \Rightarrow [B]^{x}_{S1(x)}$. Here the notation $[B]^{x}_{S1(x)}$ is used for the result of substitution into *B* the index terms from the list **S1**(**x**) instead of the correspondent variables from the list **x**. Both lists must have the same number of members. A name of a global variable of a statement sequence *S* is regarded as a name of an *N*-finite Pascal total function computing its value as a result of this *N*-finite Pascal total function (with the body *S*) run with the input data **x**. If an array is used as a result of a function then the name of this function coincides with the name of the array.

If all these *N*-finite Pascal total functions are included into the signature of the *N*-finite index first order logic then an assertion about full correctness of an *N*-finite Pascal statement sequence *S* may be formalized with the use of such an *N*-finite index first order logic language.

This section illustrates a new possibility of the introduced *N*-finite index first order logic of a finite signature using *N*-finite Pascal total functions and total *N*-finite Pascal predicates.

## Main results

The first theorem is a practically useful generalization of an application variant of the theorem from [8]. Definitions of the **QBF** problem and P-SPACE-completeness are in [1, 2, 4]. The second theorem may be regarded as a generalization of the first one.

**Theorem 1.** *For every integer N ($N \geq 2$) and for every finite signature of N-finite RAM or N-finite RASP total functions and total predicates containing {+, *, <} with constants from the set of all N-integers the problem of identical truth of an N-finite index first order formula (using arrays with the number of dimensions bounded by N) of such a signature is a P-SPACE-complete problem.*

**Proof.** An identical truth of a quantifier Boolean formula (**QBF**) which has a prenex normal form may be simulated by an *N*-finite RAM total program and sequentially by an *N*-finite RASP total program (with $N \geq 2$).

Let the logical value *true* corresponds to the number 0 and the value *false* corresponds to the number -1. In such a case logical connectives may be changed by a sequence of RAM commands:

| | | |
|---|---|---|
| $\neg x$ | corresponds to | $-x - 1$, |
| $x \vee y$ | corresponds to | $-x * y$, |
| $x \& y$ | corresponds to | $(-x-1)*(-y-1)-1 = (x+1)*(y+1)-1$. |

Quantifiers may be changed by loops based on *goto* statements. The length of the received *N*-finite RAM program increases relatively the initial one not more than in a polynomial under the initial QBF length times.

Hence, the formulated in the theorem problem is a P-SPACE-hard one as the **QBF** problem is a P-SPACE-complete one [1, 2, 4].

The formulated in the theorem problem belongs to the class **P-SPACE** because all *N-finite* RAM or *N-finite* RASP total program commands belong to **FP**. Besides, during such a program run the memory may be bounded by a polynomial under the input data length because the looping does not need the use of an additional array element.

**Theorem 2.** *For every integer N ($N \geq 2$) and for every finite signature of N-finite non-recursive Pascal total functions and N-finite non-recursive Pascal total predicates containing {+, *, <} with constants from the set of all N-reals and all N-integers the identically true problem of an N-finite index first order formula (using arrays with the number of dimensions bounded by N) of such a signature is a P-SPACE-complete problem.*

**Proof.** The number of *non-recursive N*-finite Pascal total functions and *non-recursive N*-finite Pascal total predicates are finite ones. Every *N*-finite Pascal total function and every *N*-finite Pascal total predicate belongs to **FP** because it has a finite domain and its values may be computed as a preliminary data base.

An *N*-finite Pascal total function is a generalization of an *N*-finite RAM total program. That's why the formulated in the theorem problem is a P-SPACE-hard one. This problem belongs to the class **P-SPACE** because an *N*-finite Pascal total program may be interpreted by an *N*-finite RAM program. Of course, the description of such a translator is very large and includes the full description of the Pascal language as a part.

**Note.** The predicate *"<"* allows easily to define the relation *"="*. But the author does not know a short definition of the predicate "<" in the terms of "=". That is why the relation "<" is used in the conditions of the above theorems.

The set of all *N*-finite index first order formulas of the signature from the theorem 2 may be named an *N*-finite arithmetic of such a signature.

## Conclusion

Models of a mathematical notion of a program processing only finite short data types are introduced for adequate description of a computer program. These models process only data of the computer types *real, integer and Boolean* with the bounded notation length.

Mathematical notions of an *N-finite* RAM program and an *N-finite* RASP program are introduced as generalization of a RAM program and a RASP program for computation of functions and predicates of a finite arithmetic signature. The notion of an *N-finite* Pascal program is introduced as an extension of an *N-finite* RAM program up.

An extension of a first order predicate formula is done on the base of *N-finite* data. Such an extension allows describing mathematical properties of total Pascal function or predicate. P-SPACE-completeness of the truth checking for these formulas with non-recursive functions and predicates is proved.

The introduced formulas are useful for formulation of a program correctness condition by means of Hoare triades.

## Acknowledgement

The paper is published with financial support of the project ITHEA XXI of the Institute of Information Theories and Applications FOI ITHEA (www.ithea.org) and the Association of Developers and Users of Intelligent Systems ADUIS Ukraine (www.aduis.com.ua ).

## Bibliography

1. Aho A.V., Hopcroft J.E., Ullman J.D. The design and analysis of computer algorithms. (Addison-Wesley Publishing Company Reading, Massfchusetts, 1976.
2. Du D.Z., Ko K.I. Theory of Computational Complexity. A Wiley-Interscience Publication. John Wiley & Sons, Inc. 2000.
3. Forsyth R.S. Pascal at Work and Play. An introduction to computer programming in Pascal. Chapman & Hall. London, New-York. 1982.
4. M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York, 1979.
5. Hoare C.A.R. A note on the for statement. BIT (Sver), 1972, vol.12, No 3.
6. Kleene S.C. Mathematical Logic. A Wiley-Interscience Publication. John Wiley & Sons, Inc., 1967.
7. Kosovskii N.K. Elements of mathematical logic and its applications to the theory of sub-recursive algorithms. Leningrad University Press, Leningrad, 1981. (In Russian)
8. Kosovskiy N.K. Pspace-completeness of finite order predicate logic over finite domain. 3 International Conference "Smirnov's lectures". Moscow, 2001, p. 44. (In Russian)

## Authors' Information

***Nikolay Kosovskiy*** *– Dr., Professor, Head of Computer Science Chair of St.Petersburg State University, University av., 28, Stary Petergof, St.Petersburg, 198504, Russia, e-mail: kosov@NK1022.spb.edu*

*Major Fields of Scientific Research: Mathematical Logic, Theory of Computational Complexity of Algorithms.*