ALGORITHMIC DECIDABILITY OF COMPUTER PROGRAM-FUNCTIONS LANGUAGE PROPERTIES

Nikolay Kosovskiy

Abstract*: A mathematical notion of a computer program-function for some programming languages (in particular, for some versions of Pascal and Turbo/Visual Prolog) is introduced as a restriction of an algorithm class **FP**. Such a notion is useful for more adequate formalization of finite memory of computer intended for a program run. Complexity of logic-mathematical properties proof of such program-functions is investigated. A predicate language is introduced to formulate such a partial program properties. **P-SPACE**-completeness of elementary theories under a finite signature based on such a kind of functions with the use of halting predicate and predicates of conditional equality and conditional inequalities. This allows to use the first order language for description of logic-mathematical properties of computer program-functions by extension of the used program-functions by means of a special additional value in the case of infinite looping of a program-function. In many cases it allows to replace traditional mathematical notions of an algorithm by the offered notion of a program-function in the framework of mathematical informatics.*

Keywords*: programming languages, computer program-function, complexity classes **FP**, **FP-SPACE**.*

ACM Classification Keywords*: F.2.m ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY Miscellaneous.*

Introduction

At the 30th year of XX century various mathematical notions of an algorithm were formulated. Such notions as Turing machine, Markov algorithm, RAM and RASP are widely known. Later many other notions were introduced, for example an offered by the author notion of Markov-Post algorithm [8]. From the 70th years of XX century the most of scientists believe that the class of efficient algorithms coincides with the class **FP**. Here and below **FP** is the class of functions computable by a Turing machine with number of steps not more than a polynomial under the length of values of arguments [1, 3, 5].

Such an intuitive notion as "efficient computation" was formalized by the **extended Church-Turing thesis** (see, for example, [3]): "A function computable in polynomial time in any *reasonable* computational model using a *reasonable* time complexity measure is computable by a deterministic Turing machine in polynomial time".

RAM+BOOL (i.e. random access machine with the additional bitwise logical operations) seams to be a suitable enough mathematical notion of an algorithm for a more adequate (according to the computer operations) simulation of the number of computational steps than the notion of Turing machine. For example, the definition of RAM+BOOL contains random access memory widely used in computer. The RAM+BOOL used memory size (with logarithmical weight) is maximum over all steps of a RAM+BOOL program run of the sums of the lengths of all values of all cells.

The introduced by me notion of the size of changes (which is a production of the number of steps and the used memory size) is offered to use in order to guarantee the belonging of RAM+BOOL computations to the class **FP**.

Some statements concerning complexity characteristics of RAM+BOOL computations are proved in the paper. The most important of them is the following proposition.

Proposition [7]. The class of all RAM+BOOL programs, the size of changes of which has a polynomial under the input data length upper bound, coincides with the class **FP**.

The interest to the class **P** was strengthened in 2000 after promising $1 000 000 to one who proves or disproves the equality **P** = **NP**. Here **NP** is the class of predicates computable by a nondeterministic Turing machine with number of steps not more than a polynomial under the length of values of arguments. But even the class **FP** does not take into account that every program is implemented by a computer with a fixed memory the size of which is explicitly bounded by a constant.

To take into account such a property of a real computer a mathematical notion of a computer program-function is introduced in this paper. The use of a computer implies elimination of infinite size data types.

The initial version of Pascal language [4] offered by N. Wirth has the only one infinite size data type, namely the type file. The later versions of Pascal use dynamical arrays which are also infinite size data types.

Programming languages Turbo/Visual Prolog [2] use such infinite size data types as files, lists of elements (of the same type) and terms with only non-interpreted names of functions. The notion of symbol string is also an example of infinite size data type.

It must be marked that discrete mathematics uses as a rule elements of some infinite set. For example, the set of all words in the fixed alphabet, the set of all integers, the set of all rational numbers, etc. It is often not taken into account the way of these elements notation, for example, what is the radix of a number notation.

FP-SPACE is the class of functions computable by a Turing machine with number of cells not more than a polynomial under the length of values of arguments. In the frameworks of mathematical informatics algorithms only from the class **FP-SPACE** are interesting ones. Theorems proved below demonstrate this.

Earlier received results

A polynomial number of steps for different notions of an algorithm does not necessarily provides its belonging to the class **FP** [6]. At the same time a Turing machine programming is not quite adequate the contemporary practice of programming. In connection with this a mathematical notion of integer-valued algorithm based on the well known programming language Pascal offered by N. Wirth [4], with the inclusion of dynamic arrays is the base of the introduced notion.

Let a variable of the type *integer* may take any integer (i.e. integer with any length of notation) for its value and the other types of variables are not used.

Definition. *Integer-valued algorithm (Pascal-like function) is a function of the proposed by N. Wirth language Pascal if its actual parameters are integers and it does not use files, records, sets and pointers as well as procedures and functions having names of procedures or functions for their parameters.*

Definitions of the computation step and the used space (memory length) are the basis of the complexity bounds of any computation. The definition of a computation step has, as a rule, no difficulties. But the function 2^{2^n} may be computed by different models of computer in rather different number of steps.

Definition. *Execution of every Pascal language operator from the definition of a function (including a loop and an assignment statement) is counted for one step. Every call of a previously defined function or procedure is counted as one step. A number of all additional steps necessary for execution of a main step (including all recursive calls) is adjoined to every step. (Such a call is possible while computing an arithmetical or Boolean expression.)*

A length of a result notation will be additionally computed while execution of every step.

Let *t* be an arithmetical or Boolean expression. The process of its execution may be represented by a root tree. The depth of this tree is called *the depth* of notation of *t*.

Definition. *The sum of absolute values of all sub expressions of the depth i of the expression t is called the notation length of computation of t of the depth i.*

Maximum for all i (from 1 up to the depth of t) of the notation lengths of computation of t of the depth i is called the notation length of computation of t.

Definition. *At the initial step the notation length of computation equals to the notation length of input data (values for arguments of a function). At the final step of function computation the notation length of computation equals to the length of absolute value of the computed function result.*

Maximum for all computation steps (including initial and final ones) of the notation lengths of computation is called the notation length of intermediate computations.

Here we suppose that all the numbers are written in decimal number system (although it is not in principle what radix distinguishing from 1 is used).

Definition. An integer-valued algorithm (Pascal-like function) is called double-polynomial if both the notation length of intermediate computations and the number of computation steps are not greater than a polynomial under the length of input notation.

As below we will speak about double-polynomial calculations (i.e. about class **FP**) it is needed to prove that both the notation length of intermediate calculations and the number of calculation steps are not greater than a polynomial under the length of input notation.

Note that a Pascal-like function may use integers of arbitrarily large notation length. Double-polynomial functions *c*, *l* and *r* as well as their iterations may be used instead of arrays. These functions have properties

$$c(x,y)=c(x',y') \Leftrightarrow x=x' \ \& \ y=y', \quad l(c(x,y))=x, \quad r(c(x,y))=y.$$

The function $(x+y)(x+y+1)/2+x$ numerates the set of all pairs of natural numbers (including ones containing zero) by natural numbers (including zero). So the function *c* based on it is a surjection from the set of all pairs of natural numbers on the set of natural numbers.

Value of all elements of the array *x[1..n]* may be represented by with the help of the function *c*.

Value of every element of the array *x[1..n]* may be expressed by means of superposition of functions *l* and *r*.

Note that the author of the Pascal language did not include the power operation. It can not be computed in a polynomial time. But he included the operator SQR (i.e. square function) which allows to calculate *2^{2^n}* in *n* steps.

An idea of the prove of the following theorem is done in the appendix of [KIO osnovy].

Theorem. *The class of double-polynomial Pascal-like functions coincides with the class* **FP**.

Initial definitions

As predicate calculus language is intended for signatures containing only completely defined functions and predicates then every data type (except the Boolean one) may be supplemented by a new value appeared in the case of infinite looping of a program. This value may be denoted by '?'. A theorem about existence of an algorithm solving a Halting problem for computer program-functions will be proved below. This is an essential argument for using classical first order predicate language for description of logic-mathematical properties of computer program-functions.

So, the set of all values of every individual variable contains '?'. If the value of at least one argument of a computer program-function is '?' then its value is '?'. If the value of at least one argument of a computer program-predicate is not computed within bounded space then its value is *false.*

Let ***x*** be a tuple of all arguments of the computer program-function *F*.

The expression *!F(**x**)* means that *F(**x**)* is computed within bounded space, i.e. it is false that *F(**x**) = '?'.* The predicate ! is called a halting predicate.

Let t_1 and t_2 be terms.

The expression t_1 ~= t_2 means conditional equality of the left and the right parts. That is the results of their computations coincide. As we have introduced the value '?', the conditional equality becomes a traditional one.

Fix positive integers *m* and *M*. Let the available memory of the used computer is *m* Kbyte. A first order language with the range of individual variables coinciding with the set of all elements of the computer type *integer* added by the value '?' would be used. This set is the set of all reminders modulo *2^(2^M)* in the semi-segment *[– 2^(2^M – 1), 2^(2^M – 1))*.

If *M = 4* then this semi-segment coincides with the set of all integers of computer type *integer* for IBM-compatible computers. If *M = 5* then it is numbers of the type *double integer* for the same computers.

Definition. Computer program-function is such a program-function that for every input data its value not equals to '?' iff it is computed within *m* Kbyte bounded space.

Describe the classical first order predicate language for description of logic-mathematical properties of computer program-functions.

The considered signature consists of

- range of individual variables containing the symbol '?' and all integers with the absolute value not greater then *m* Kbyte*;*
- constants which are the same*;*
- a finite number of computer program-functions arguments and results of which are '?' or integers from all integers from *[– 2^(2^M – 1), 2^(2^M – 1));*
- a finite number of computer program-predicates arguments of which are '?' or integers from all integers from *[– 2^(2^M – 1), 2^(2^M – 1))* and results are of the type *Boolean,* including halting predicate, predicates of conditional equality and conditional inequalities.

Under predicates of conditional inequalities of terms (denoted as ~<, ~≤, ~>, ~≥) we understand completely defined predicates every of which is valid iff the both arguments are infinitely looped (i.e. equals '?') or the both arguments are defined and their values are in the condition <, ≤, >, ≥ respectively.

Besides that under predicates of conditional inequalities of terms (denoted as ⊂~<, ⊂~≤, ⊃~>, ⊃~≥) we understand completely defined predicates every of which is valid if the both arguments are not '?' and their values are in the condition <, ≤, >, ≥ respectively.

More precisely, the relations ⊂~< and ⊂~≤ are valid iff, at the first, if the first argument is are not '?' then the second one is are not '?' too and, at the second, the values of arguments are in the condition < or ≤ respectively.

The relations ⊃~> and ⊃~≥ are valid iff, at the first, if the second argument is are not '?' then the first one is are not '?' too and, at the second, the values of arguments are in the condition < or ≤ respectively.

Logic-mathematical formulas (describing assertions upon computer program-functions) are constructed according to the rules of first order predicates language by means of logical connectives and quantifiers upon integers from the semi-segment *[– 2^(2^M – 1), 2^(2^M – 1))*.

Valid formulas of such a language will be called formulas of elementary theory in the described signature.

Main results

Theorem 1. *The problem of belonging of a first order formula in the considered signature to the elementary theory in the same signature is* **P-SPACE***-complete.*

Scheme of proof. The formulated in the theorem problem may be solved on a Turing machine with polynomially bounded number of cells, because the absolute value of the used (including auxiliary) numbers are not greater then *2^(2^M – 1)*. And the used memory is not greater than *m* Kbyte.

The problem Quantified Boolean Formula (QBF) [Aho, Gary, Du] is polynomially m-reducible to the problem under consideration.

The proved theorem permits some generalizations. The first one is connected with the introducing of several sorts independent signatures united into one common signature. For every function all its arguments and its result of belong to the same independent signatures.

The second generalization consists in the introducing of an argument and a result type of every base function and predicate from the common signature. Every argument and result type uses its own finite set of all variable values. Elements of this common finite set of all variable values have names written as a string of symbols of the used code. New data types may be '?' or numbers of the type *real* or *double real* or strings of symbols with the bounded in advance length. Remind that If the value of at least one argument of a computer program-predicate is not computed within bounded space then its value is *false.*

Formulations and schemes of proofs of the theorems for these generalizations are similar to such for theorem 1. They are valid for every Pascal version computed program-function and computed program-predicates.

Consider another problem, namely the Halting problem for a computer program-function.

Problem H_m.

INSTANSE: *f* – text of a computer program-function *F*;

x – list of values.

QUESTION: Whether *F* has a successful result while its run under the list ***x*** within *m* Kbyte space.

Theorem 2. *For every positive integer m the problem* $\mathbf{H_m}$ *is* **P-SPACE***-complete.*

Scheme of proof. Test of inequality to '?' for a computer program-function result with the input data ***x*** may be computed by means of number of steps calculation within additional memory. If such number of steps is greater than the number of different memory contents multiplied by the length of the program text, then the value of program-function result with the input data ***x*** equals to '?'. Such a test may be computed within the polynomially (under the length of a program text) bounded memory. Hence the problem belongs to **P-SPACE**.

As the problem QBF is polynomially m-reducible to the problem $\mathbf{H_m}$ then $\mathbf{H_m}$ is **P-SPACE**-complete.

Remark. *For every positive integer m the problem* $\mathbf{H_m}$ *remains* **P-SPACE***-complete if all variables have the type Boolean.*

Conclusion

Widely known algorithmically undecidable Halting problem for algorithms (with infinite type of data) becomes a decidable one and, moreover, **P-SPACE**-complete one, if we use computer program-functions. It must be marked that the decidability of the algorithmic halting problem can not be checked by a finite automata because the parameter *f* of the above formulated halting problem $\mathbf{H_m}$ has potentially infinite set of values. But it is clear that a computer program-function within bounded space has essentially different properties in comparison with a program-function using arbitrarily large space.

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. Babaev I.O., Gerasimov M.A., Kosovskiy N.K., Soloviev I.P. Intellectual Programming. Turbo Prolog and Refal-5 for personal computers. St.Petersburg State University Press. 1992. (In Russian)
3. Du D.Z.,Ko K.I. Theory of Computational Complexity. A Wiley-Interscience Publication. John Wiley & Sons, Inc. 2000.
4. Forsyth R.S. Pascal at Work and Play. An introduction to computer programming in Pascal. Chapman & Hall. London, New-York. 1982.
5. M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Freeman, New York, 1979).
6. Kosovskiy N., Kosovskaya T. Polynomial-Time Effectiveness of Pascal, Turbo Prolog, Visual Prolog and Refal-5 Programs // International Journal "Information Models and Analysis", vol.1. 2012. Pp. 94 – 99.
7. Kosovskiy N.K. Polynomial Upper Bounds of RAM+BOOL Program Size of Changes for the Proof of Belonging to FP. // Researches on constructive mathematics and mathematical logic. – Spb: St.Petersburg Department of Mathematical V.A. Steklov Institute of Russian Academy of Sciences. 2012. Pp. 105 – 110.
8. Yakhontov S.V., Kosovskiy N.K., Kosovskaya T.M. Effective upon Time and Memory Algorithmic Approximations of numbers and functions. Manuel. St.Petersburg State University Press. 2012. (In Russian)

Acknowledgments

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).

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 Complexity of Algorithms.