Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 11, Number 6 (2015), pp. 4911-4917

http://www.ripublication.com

# A Mathematical Notion Of A Package Of Level Markov-Post Algorithms For Word Processing

**Nikolay K. KOSOVSKII, Tatiana M. KOSOVSKAYA**

*Mathematics and Mechancs Faculty, St. Petersburg State University*
*University av., 28, Stary Petergof, St. Petersburg, Russia*
*kosov@nk1022.spb.edu, kosovtm@gmail.com*

**Abstract**

A mathematical notion of a package of level Markov-Post algorithms which is more adequate then that of the Markov algorithm to the contemporary programming languages using calls of sub-programs is introduced. A possibility of an algorithmic total extension of an algorithm from such a package (using a bounded space) is proved.

**Keywords:** Markov-Post algorithm, halting problem for a Markov-Post algorithm, classes **P**, **FP**, **P-SPACE**, **FP-SPACE, LIN-SPACE, FLIN-SPACE.**

## Introduction

Programming languages may be regarded as machine-oriented ones, assemblers, that of high-level and super-high-level. At the same time mathematical notions of an algorithm are developed as a rule in the frameworks of a machine-oriented level because one of the main properties of a high-level programming language is the use of a procedure. But sometimes formal and informal Algol-like or Pascal-like variants of classical mathematical notions of an algorithm are considered (see, for example, [1], [2]).

A mathematical notion of a high-level algorithm is proposed below. Such a notion is intended for processing of strings (words in a done alphabet) and is defined on the base of Markov algorithm notion, which is a base one in the education in the field of algorithm theory and its applications to programming [8]. It is allowed to use a premise of a Post rule (with deterministic semantics of their matching) as a pattern. It is also allowed to use the computation of a functional term value. The new notion is defined as a package of algorithms and almost every algorithm from this package is shorter and more powerful than a Turing machine or a Markov algorithm. Also this

notion preserves the time and space complexity (up to a polynomial for the time complexity and up to a multiplicative constant for the space complexity) of a Turing machine.

The introduced notion is called a package of level Markov-Post algorithms. Package of *k*-level Markov-Post algorithms is a collection of Markov-Post algorithms divided by *k* levels. A first-level algorithm does not contain a call of any algorithm. Markov-Post algorithm of a higher level does not contain a call of an algorithm (from the package) of the same level algorithm or an algorithm of a greater level. But it contains a call of at least one algorithm of the previous level.

According to this description a level Markov-Post algorithm does not contain an algorithm with a recursive call.

The proposed mathematical notion of a level Markov-Post package is a subset of the majority of Refal language dialects from [7], [2] and, hence, unites mathematical notions of an algorithm package and practical programming languages.

As the notion of a level Markov-Post algorithm is a mathematical notion of an algorithm then the theorem about algorithmic undecidability of the halting problem is valid for it. But, as it would be proved bellow in theorem 1, for every main algorithm from a level Markov-Post package which runs in a bounded space there exists its total extension running within the space bounded by a linear (with the coefficient 2) function of the initial algorithm space length. This coefficient is less than corresponding coefficient for a Turing machine.

So, we can require for many algorithms to be total ones. Note that the totality of a program is an important property of a friendly software.

Besides, total functions and predicates may be used in a signature for traditional classical first order formulas [5].

## 1. The used definitions

Let us give some definitions used below.

**Defininion 1.** *Markov algorithm f is a system of Markov algorithm rules in the form*
$f: \{A_i \to [\cdot] B_i, i=1,...,k,$
*where* $A_i$, $B_i$ *are the words for every* $i=1,..., k$. *The rule is called a final one if it contains the sign* $\cdot$.

The notation $[\cdot]$ means that the sign $\cdot$ may be present or absent.

**Defininion 2.** *The notion of a word term is defined by induction: a word is a word term; a variable for letters (s-variable in the form* $s_i$*) is a word term; a variable for strings of letters (e-variable in the form* $e_i$*) is a word term; if* $t_1$, $t_2$ *are word terms then* $t_1t_2$ *is a word term.*

**Defininion 3.** *A Markov-Post algorithm rule has the form*

$$A_0\, e_{j1}\, A_1\, e_{j2}\, A_2 \ldots A_m\, e_{jm}\, A_{m+1} \to [\cdot]t, \qquad (1)$$

*where* $A_0$, $A_1$, $A_2$, …, $A_{m+1}$ *are word terms without word variables, t is a word term without any word variables except* $e_{j1}$, $e_{j2}$, …, $e_{jm}$ *and without any letter variables*

*except having an occurrence in at least one of the words $A_0, A_1, A_2, \ldots, A_{m+1}$. Variables $e_{j1}, e_{j2}, \ldots, e_{jm}$ are local variables of this rule. The rule is called a final one if it contains the sign □.*

*Markov-Post algorithm is a sequence of Markov-Post algorithm rules divided by a sign ; and included into curly brackets preceded by the name of this algorithm.*

The manner of a Markov-Post algorithm rule application is the same as that for a Markov algorithm. More exactly, every step of computation uses the first (from the beginning) applicable rule.

If a final rule is applied to the word *W* or no rule is applicable to the word *W* then the algorithm stops (with the word *W* as a result in the second case). Deterministic application of the above formulated Markov-Post algorithm rule (1) to the word *Q* is the following.

– *m* times embedded in each other loops upon the lengths of variables $e_{jk}$ ($k= 1,..., m$) are created. For every loop the loop counter increases from 0 up to the length of the word *Q*.
– Matching of the left part of the rule with the word *Q* is checked for every list of these variables values.
– After the first successful matching of the values of $e_{j1}, e_{j2}, \ldots, e_{jm}$ they are substituted into the term *t*. The word received from *t* is the result of the rule application.

**Defininion 4.** *The notion of a functional word term is defined by induction: a word term is a functional word term; a word in the form*

< ⟨ Function Name ⟩ ⟨ Blanc ⟩ ⟨ Functional Word Term ⟩ >

*is a functional word term; if $t_1, t_2$ are functional word terms then $t_1t_2$ is a functional word term.*

Note that a prefix notation is used for every function.

**Defininion 5.** *A Markov-Post algorithm without any call of an algorithm is called a 1-st level Markov-Post algorithm. A finite set of 1-st level Markov-Post algorithms is called a package of the 1-st level Markov-Post algorithms.*

*An (l+1)-th level Markov-Post algorithm is a Markov-Post algorithm which does not contain a call of an algorithm of the same level algorithm or an algorithm of a greater level but contains a call of at least one algorithm of the lth level.*

*An (l+1)-th level Markov-Post package consists of the main (l+1)-th level algorithm and a finite set of auxiliary algorithms of the lesser levels. The main algorithm may contain rules with a functional word term in the right part which uses a call of an algorithm from a k-th level Markov-Post package for $k \leq l$ and at least one call of an algorithm from a lth level Markov-Post package. An auxiliary algorithm is that of kth level Markov-Post package for $k \leq l$ which is called from the higher level one.*

*A package of algorithms is a level Markov-Post package if it is a k-th level Markov-Post package for some k.*

*An algorithm is a level Markov-Post algorithm if it is the main algorithm of some level Markov-Post package.*

For example, the 1-st level Markov-Post algorithm checking whether a word is palindrome has the form

*Palindrome{* $\to$ □*true;* $s_1\ e_2\ s_1 \to$ ⟨*Palindrome*~$e_2$⟩*;* $e_1 \to$ □*false }.*

The definition of the main algorithm from a level Markov-Post package corresponds to the main non-recursive program with calls of auxiliary non-recursive subprograms which are included into the package.

**Defininion 6.** *An algorithm F from a level Markov-Post package uses the space s(n) (where n is the length of an input word) if during the computation of F for every called algorithm every intermediate word and the result one (word term and functional word term) have the length not greater than s(n).*

**Defininion 7.** *A function s(n) is called a space constructible one if there exists a level Markov-Post algorithm which gives a word consisting of s(n) letters 0 according to an input data with the length n.*

It is evident that every space constructible function is a total one. Such usual functions as $2^n$, $n!$, $\lceil n \log(n+1) \rceil$, $n^k$ are space constructible ones [1].

**Defininion 8.** [1], [3], [4] *Class* **FP** *(class* **P***) is the class of all total functions (predicates respectively) which are computable by a Turing machine in a polynomial under the input data length number of steps.*

*Class* **FLIN-SPACE** *(class* **LIN-SPACE***) is the class of all total functions (predicates respectively) which are computable by a Turing machine, the number of cells visited by the tape head of M at least once during the computation, is not greater than a linear function of the input data length.*

**Defininion 9.** *For every space complexity class* **C** *the notation* **pC** *denotes the class of all partial algorithms such that the complexity characteristic of its run coincides with that of the class* **C**.

**Defininion 10.** *For every complexity classes* **C1** *and* **C2** *the class* **C1** || **C2** *is the class of all algorithms belonging both to the class* **C1** *and to the class* **C2**.

The classes **C1** || **C2** and **C1** $\cap$ **C2** are not equal. Definitions of the complexity classes are done in the terms of a Turing machine that is why the class **C1** $\cap$ **C2** may contain such a level Markov-Post algorithm that there exist two Turing machines computing the same functions one of which belongs to the class **C1** \ **C2** and the other belongs to the class **C2** \ **C1**.

## 2. Main results

**Theorem 1.** *Let a level Markov-Post package with the main algorithm M uses an alphabet A having at least two letters and uses the space measured by a done space constructible function s of the input word length n. Let alphabet* $A_1$ *be an extension of A by means of one letter.*

*Then for the main algorithm M from the level Markov-Post package there exists a level Markov-Post package with the main algorithm $M_1$ which is a total extension of M by means of a fixed word z and $M_1$ uses the alphabet $A_1$. The space used by $M_1$ and all auxiliary algorithms is not greater than the linear function 2s(n)+C for some constant C.*

**Proof.** Let every algorithm from the level Markov-Post package with the main algorithm *M* use an alphabet with digits $A= \{0,1,...,d-1\}$. The word in such an alphabet may be regarded as a number written in the number system with the radix *d*. The number of all distinct words with the length not greater than *s(n)* is $1 + d + d^2 +... + d^{s(n)} = (d^{s(n)+1} - 1) / (d - 1)$ and is less than $d^{s(n)+1}$.
The contents of the used space may be repeated for every auxiliary less level algorithm. Let *N* be the number of the auxiliary less level algorithms in the definition of *M*. If the algorithm *M* stops and uses the space *s(n)* then the number of its steps (i.e. the number of its rule applications and the number of its less level sub-algorithm rule applications) does not exceed $d^{s(n)+1} \mathrm{N} = d^{s(n)+1+\log N}$. Otherwise the algorithm *M* does not stop.
Let the sequence of $s(n) + 1 + \log N$ digits *d* be built by a *k*-th level Markov-Post algorithm *B*. Let also the list of rules for the algorithm *M* consists of the rules in the form $u \to v$ or $u \to\!\square\, v$. And let $A_1 = A \cup \{*\}$.
The algorithm $\mathrm{M}_1$ is defined in the following way
$M_1 : \{e_1 \to \langle M_2\, e_1 * 1 \langle B\, e_1 \rangle \rangle$,
where $e_1$ is a variable for words.
The first rule of $M_2$ is $e_1 * 0 \to\!\square\, z$. The other rules of $M_2$ are all modified rules (taken in the same order) for *M*:
a rule in the form $u \to v$ is changed by a rule $u * e_1 \to v * \langle dec\ e_1 \rangle$,
a rule in the form $u \to\!\square\, v$ is changed by a rule $u * e_1 \to\!\square\, v$ ,
where *dec* is the name of the function decreasing its non-zero argument value by 1 and not changing its zero argument value.
The algorithm $M_2$ is a total one as well as all auxiliary ones of the less level. At the same time the space used by $M_1$ does not exceed $2s(n)$ added with a constant which does not depend on the input word length *n*. ■

**Theorem 2.** *For every alphabet used by a level Markov-Post algorithm M from theorem 1 there exists an algorithm from the class* **FP || FLIN-SPACE** *transforming the notation of a program M and the notation of a program for s into the notation of a program $M_1$ which is a total extension of M by means of a fixed word.*
The proof immediately follows from complexity bounds of constructions used in the proof of theorem 1 and simulation of a level Markov-Post algorithm by a Turing machine.

**Corollary 1.** *If an algorithm M belongs to the class* **pFLIN-SPACE** *then an algorithm $M_1$ from theorem 1 belongs to the class* **FLIN-SPACE**.
Nevertheless the following proposition is valid.

**Proposition 1.** *The halting problem for an arbitrary algorithm from the class* **pFLIN-SPACE** *and an arbitrary input data does not belong to the class* **FLIN-SPACE***.*

**Proof.** Let the formulated problem belongs to **FLIN-SPACE**. Then the predicate $H$ solving this problem belongs to **LIN-SPACE**. That is

$H \in$ **LIN-SPACE** $\& \forall M\,(M \backslash in \in$ **pLIN-SPACE** $\Rightarrow (H(\#\,M) = 1 \Leftrightarrow \neg !M(\#\,M)))$,

where the notation $\#\,M$ is used for the code of the algorithm $M$.

As **LIN-SPACE** \subset **pLIN-SPACE** then

$H \in$ **pLIN-SPACE** $\& \forall M\,(M \backslash in \in$ ***p*LIN-SPACE** $\Rightarrow (H(\#\,M) = 1 \Leftrightarrow \neg M(\#\,M)))$.

For $M = H$ we have

$H \in$ **pLIN-SPACE** $\& (H(\#\,H) = 1 \Leftrightarrow \neg\, !H(\#\,H))$.

Taking into account that $H(\#\,H) = 1 \Rightarrow !H(\#\,H)$ we have a contradiction. ■

So the halting problem for an algorithm from **pLIN-SPACE** (an algorithm is fixed) belongs to **LIN-SPACE**, but the halting problem for all algorithms from **pLIN-SPACE** (an algorithm is an argument of the problem) does not belong to **LIN-SPACE**.

**Remark.** The analysis of theorem 2 proof allows to state that the number of steps of the level Markov-Post algorithm $M_1$ developed in a such a way exponentially depends on the length of the value $s(n)$, where $n$ is the length of the input word for the algorithm $M$.

**Conclusion**

A traditional mathematical notation for a value of a function of its arguments computation (except, for example, an explicit division by zero) means its successful computation.

At the same time in mathematical theory of algorithms the notation for application of an algorithm to its arguments assumes that the algorithm can never stop. That is why the notation $A(X)$ for an algorithm $A$ can not be used in a logic-mathematical first order formula describing some property of an algorithm or relation between several algorithms (see, for example, [5]).

Note that the use of the halting sign and the sign of conditional equality leads out the frameworks of the traditional classical first order logic.

Nevertheless, as it was shown in the present paper, an important practical algorithm running upon (polynomially or even more) bounded space may be extended by means of a fixed word up to a total algorithm using a linear (under the initial) space.

So, such a total extension may be used in a formula of a signature of the traditional classical first order logic.

**References**


1. Aho, A.V., J.E. Hopcroft and J.D. Ullman (1976). *The design and analysis of computer algorithms.* Addison-Wesley Publishing Company Reading, Massfchusetts.
2. Babaev, I.O., M.A. Gerasimov, N.K. Kosovskii and I.P. Solovjov (1993). *Intellectual programming. Turbo-Prolog and Refal-5 for personal computers.* St.Petersburg State University Publishing. (in Russian).
3. Du, D.Z. and K.I. Ko (2000). *Theory of Computational Complexity*. A Wiley-Interscience Publication. John Wiley & Sons, Inc.
4. Garey. M.R. and D.S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York.
5. Kosovskii,N.K. *Elements of mathematical logic and its applications to the theory of sub-recursive algorithms*. Leningrad University Press, Leningrad (1981). (in Russian).
6. Kosovskaya, T.M. and N.K. Kosovskiy (2010). Belonging to **FP** of double-polynomial Pascal-like function over subprograms from **FP**. *Computer Tools in Education.* **3**, pp. 3-7. (in Russian).
7. Turchin, V.F. Refal 5. A guidebook on programming and handbook // www. refal.org/rf5_frm.htm (in Russian).
8. J\H{u}rg Nievergelt. Equivalence of TMs, PMs and Markov algorithm. // http:// www.jn.inf.ethz.ch/education/script/chapter8.pdf, accessed December 15, 2007.