AMERICAN SCIENTIFIC JOURNAL

American Scientific Journal
№ (17) / 2017
Vol.1

Layout man: Mark O'Donovan
Layout: Catherine Johnson

Address: 90 st. – Elmhurst AV, Queens, NY, United States
Web-site: http://american-science.com
E-mail: info@american-science.com

Copies: 1000 copies.
Printed in 90 st. – Elmhurst AV, Queens, NY, United States

CONTENTS

КОМПЬЮТЕРНЫЕ НАУКИ

THE PARADIGM OF LINGUISTIC SUPPLY SUBMISSION BY GENERATIVE GRAMMAR ASSISTANCE

Stanislav Velykodniy,
Ph.D. in Computer Science, Dr. hab. student, Associate Professor, Odessa State Environmental University, Odessa, Ukraine,
Olena Tymofieieva,
M.Sc. in Computer Science, Ph. D. student, Odessa State Environmental University, Odessa, Ukraine

Abstract
The article deals with the creation of a system of concepts that form the paradigm of the reengineering of information technologies, which is necessary in the case of their evolutionary development. The scientific basis of any programming languages is linguistics, which studies laws, models and language rules. The linguistic provision of information technology addresses the construction of a software system by means of one or more (mutually agreed) programming languages, each of which is based on the rules of a particular grammar. The mathematical apparatus of generating grammars allows to describe the process of translating a program system written in one programming language into another specific language. The created paradigm allows to work with multi-level information technologies, with components that are written in different programming languages. The paradigm formulated in the article, from the scientific point of view, will be the basis of the methodology of reengineering software systems, and from the practical point of view it will be useful for system programmers who work with multi-language superstructures of software systems that acquire evolutionary development over time and improve in the process of use.

Keywords: alphabet, axiom, generative linguistics, grammar, products string, program system, programming language, symbol

Introduction

Information is the base for any modern society. Without the exchange and analysis of information it is impossible not only the development of a living organism, but also its existence. There are many means of transmitting information between creatures, mostly through sensory organs. As for the humans, one of the main means of information transmission is the language, and not only its sound component (in fact information can be transmitted in written language, sign language, Braille font, etc.).

Information technology is the modern foundation for acceleration of the exchange of information among humans, the technical side of which consists of many different devices, which, in essence, are modifications of the computer. These devices also exchange information in the form of data (not only among themselves, but also with the person – the user or the operator), and, depending on the levels of data representation, the information can be presented as: binary, octal, decimal, hex codes; machine code; low and high level programming languages, etc.

Overview on Previous Research Activities

The scientific basis of any language (including programming languages) is *linguistics*, which studies laws, models and language rules. A *generic linguistics* [1], the founder of which was Avram Noam Chomsky, in the Soviet era, the interpretation of "A.N. Khomsky", was a special branch of linguistics that should be applied to the structure of programming languages, which created a revolution in linguistics [2, 3].

By the way of the assigning of true chains, formal grammars are divided into generative and recognizable. The generators include the grammars in which one can construct any valid chain with an indication of its structure and it is impossible to construct any wrong chains. For the first time, the notion of generative (generative) grammar was proposed by A. N. Chomsky [4]. Recognition grammar is a grammar that allows you to establish the fidelity of an arbitrarily selected chain, and, if it is correct, to find out its structure.

The *linguistic provision* of information technology addresses the construction of a software system by means of one or more (mutually agreed) programming languages, each of which is based on the *rules* of a particular grammar [5].

Formal languages include, in particular, artificial languages for communication between the operator and the computer (programming languages) [6].

The purpose of the study is to create a system of concepts that forms the paradigm of information technology reengineering, which will allow working with multi-level software systems whose component parts are written in different programming languages.

Statement of the problem

To designate the description of a formal language, it is necessary, first of all, to indicate the *alphabet*, that is, a collection of objects called symbols (or letters), each of which can be reproduced in an unlimited number of instances, and second to set the formal grammar of language, that is, list the rules by which the characters are compiledinto the sequences belonging to a specific language.

Any programming language is a set of chains in some final alphabet. In linguistics, instead of the term "alphabet", the term "dictionary" is used because the elements from which it is composed are word forms [7]. At the same time, the chain over the dictionary is considered as a phrase or sentence.

Note that each symbol of the alphabet is considered inseparable in the sense that when constructing chains, its graphic elements (parts of characters) are never used, and any sequence of characters uniquely represents a certain chain.

In practice, this requirement is achieved, for example, by setting a "space" (a standard length gap) between the symbols. This "space" exceeds the length of any of the spaces encountered inside the characters of the alphabet.

Rules of formal grammar should be considered as "*products*" (exit rules) - elementary operations, which, if applied in a definite sequence to the original chain (*axioms*), generate only the right chains. The very sequence of rules used in the process of generating a chain is its rendering. The language thus defined in that way is a formal system. Known examples of formal systems are logical calculations (statements, predicates) that relate to the sections of mathematical logic.

Materials and research results

Generative grammar or, in short, grammar is basically an ordered *set*:

$$G = \langle \mathrm{A}, \Psi, \varepsilon, \mathrm{Z} \rangle, \tag{1}$$

where $\mathrm{A} = \{a_1, a_2, \ldots, a_m\}$ – base terminal alphabet;

Ψ – auxiliary (*after-terminal*) alphabet, the characters of which are indicated by lowercase Greek letters;

$\varepsilon \ (\varepsilon \in \Psi)$ – initial (after-terminal) character;

Z – the final system of variables:

$$\mathrm{Z} = \{u_i \to v_i \mid i = 1, 2, \ldots, k\}, \tag{2}$$

where u_i – is the chain;

$v_i \in \delta(v)$, where $\delta(v)$ – *free semigroup* over the combined alphabet Θ:

$$\Theta = (\mathrm{A} \cup \Psi). \tag{3}$$

In other words, the characters from the main alphabet A are the primary units of the language that is defined. Symbols of the alphabet Ψare *meta-changeable* and used in the presentation of the correct chains (in natural languages, such variables are grammatical classes: noun, verb, etc.).

ε – a meta-changeable axiom from which all the correct chains are built (in the natural languages the grammatical class "sentence" corresponds to the axiom).

Z – a layout of grammar consisting of products (the rules of presentation – the grammatical rules of the determined language).

For example, the generative grammar is:

$$G_0 = \langle \{a, b, c\}, \{\alpha, \beta, \chi\}, \varepsilon, \mathrm{Z}_0 \rangle, \tag{4}$$

and Z_0 has a set of rules:

$$\mathrm{Z}_0 = \begin{cases} \varepsilon \to abc, \\ \varepsilon \to b, \\ \varepsilon \to \alpha\alpha, \\ abc \to c. \end{cases} \tag{5}$$

Definition of the language $L(\mathrm{G})$, produced by generative grammar G, associated with concept of "*rendering*".

Assuming x, y are chains belonging to free semigroup $\delta(v)$.

Chain y is direct output from chain x in grammar G:

$$x \underset{G}{\Rightarrow} y \text{ or } x \Rightarrow y \text{ (where } G \text{ is meant)}, \tag{6}$$

if there is a product $u \to v$ in a scheme Z of the given grammar:

$$\begin{cases} x = x_1 u x_2, \\ y = x_1 v x_2; \end{cases} \tag{7}$$

where $x_1, x_2 \in \delta(v)$.

Then the chain y is obtained as a result of applying outcome $u \to v \in Z$ to a chain x, which means the replacement in the chain x of the selected entry on the left part u of the supplied product to it's right part v.

For example, in grammar G_0;

$$b\varepsilon c \Rightarrow b\alpha\alpha c, \quad abcba \Rightarrow cba, \quad \ldots \tag{8}$$

The chain y can be derived from chain x belonging to a grammar G, $x \overset{*}{\underset{G}{\Rightarrow}} y$ or $x \overset{*}{\Rightarrow} y$ which is similar to (6) if chains x and y coincided or there is such a sequence of chains $z_0, z_1, \ldots, z_k$, that

$$z_0 = x, \quad z_k = y \quad \wedge \quad \forall i \ (1 \le i \le k) \quad z_{i-1} \Rightarrow z_i . \tag{9}$$

The sequence of chains $Q = (z_0, z_1, \ldots, z_k)$ is called "rendering" of the chain y from chain x in grammar G.

For example, in grammar $G_0 \ \varepsilon \overset{*}{\Rightarrow} acc$, where progression is the result of rendering chain acc from chain ε:

$$\langle \varepsilon \Rightarrow abc; abc \Rightarrow a\varepsilon c\,|b \to \varepsilon; a\varepsilon c \Rightarrow aabcc|\varepsilon \to abc;$$

$$aabcc \Rightarrow acc|abc \to c\rangle. \quad (10)$$

It should be added that at each step of the rendering, you can choose any of the products that can be applied at the moment. This means that the sequencing of the use of products in grammar is arbitrary and any products allowed to be applied after another, but within a system of rules.

Thus, the notion of generating grammar is fundamentally different from the notion of "normal algorithm", in which substitutions are of a definite character and are strictly executed in advance of the specified sequence.

Rendering $x \overset{*}{\underset{G}{\Rightarrow}} y$ could be considered as complete if $y \in \delta(\mathrm{A})$ – that is the chain y consists of terminal characters. Any complete rendering ends with use of the products if their right-hand sides are terminal chains. Specified products we will call "*final products*" of the given grammar.

If $x \overset{*}{\underset{G}{\Rightarrow}} y$ and $y \notin \delta(\mathrm{A})$, and there is no rules in system Z that could be applyied to the chain y, then rendering of chain y from chain x in grammar G is called *dead end*.

For example: rendering that generated in (10) is complete in grammar G_0, acc is the end product of grammar G_0.

And the following rendering:

$$\langle aabcabc \Rightarrow a\varepsilon\, abc\,|abc \to \varepsilon; a\varepsilon\, abc \Rightarrow a\varepsilon\, c\,|abc \to c;$$

$$a\varepsilon\, c \Rightarrow a\alpha\alpha\, c\,|\varepsilon \to \alpha\alpha\rangle \quad (11)$$

is the dead end rendering of the chain $a\alpha\alpha c$ from the chain $aabcabc$ in grammar G_0.

Now let's take a look at generative grammar $G = \langle \mathrm{A}, \Psi, \varepsilon, \mathrm{Z}\rangle$ defines the language that matches it.

The chain $x \in \delta(\mathrm{A})$ will be true is there is at least one complete rendering of the chain from axiom ε in grammar G.

In other words the chain x is true if:

$x \in \delta(\mathrm{A})$ – chain x consists of terminal characters;

$\varepsilon \overset{*}{\underset{G}{\Rightarrow}} x$ – such rendering of chain x from axiom ε exists.

Variety of all true chains in grammar G creates a language $L(G)$ that is generated by grammar G. For example, grammar G_0 generates following language:

$$L(G_0) = \{x^n y x^n \cup y^m x y^m | n, m = 0, 1, 2, \ldots\}. \quad (12)$$

Consequently, each grammar $G = \langle \mathrm{A}, \Psi, \varepsilon, \mathrm{Z}\rangle$ clearly corresponds to the language $L(G)$ generated by this grammar.

However, this correspondence is not isomorphic: the same language can be generated by various grammars. This allows you to apply the equivalence relation to the endless set of grammars.

Grammars G and G' can be considerate to be equivalent $(G \Leftrightarrow G')$ if $L(G) = L(G')$ that is, grammar G and G' generate one language.

For example, grammar:

$$G_1 = \langle\{a, b, c\}, \{\varepsilon\}, \varepsilon, \mathrm{Z}_1\rangle, \quad (13)$$

the scheme of which has following set of rules:

$$\mathrm{Z}_1 = \begin{cases} \varepsilon \to abc, \\ \varepsilon \to b, \\ \varepsilon \to c, \end{cases} \quad (14)$$

will generate the language $L(G_1)$, which coincides with language $L(G_0)$ and thereafter means that $G_0 \Leftrightarrow G_1$.

Conclusion

Summing-up the outlined content, we will note that generative and grammar can be widely used in considering the linguistic provision of software systems. Particularly important, the generating grammar tool gains in the event of the necessary reengineering of the program code, which is written in different programming languages.

The paradigm formulated in the article, from the scientific point of view, will be the basis of the methodology of reengineering software systems, and from the practical point of view it will be useful for system programmers who work with multi-language superstructures of software systems that acquire evolutionary development over time and improve in the process of use.

References

[1] V. P. Rudnev, "Heneratyvnaya lynhvystyka" (Slovar' kul'tury XX veka) [Online]. Available: http://www.gumer.info/bibliotek_Buks/Culture/ Rudnev/Dict/_04.php.

[2] Noam Chomsky, "Logical Syntax and Semantics: Their Linguistic Relevance" [Online]. Available: https://chomsky.info/wp-content/uploads/ 195503-.pdf.

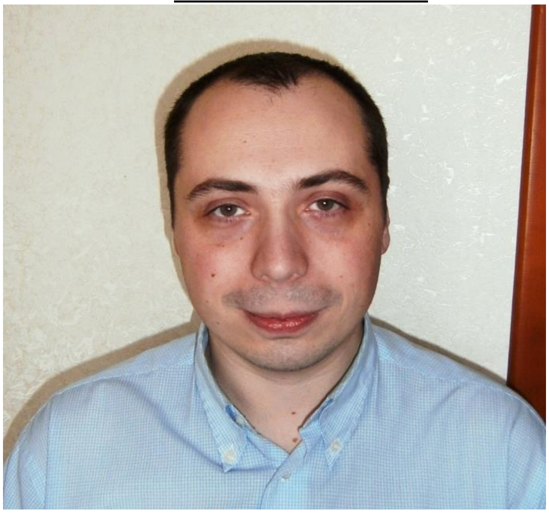
[3] Noam Chomsky, "Three Models for the Description of Language" [Online]. Available: https://chomsky.info/wp-content/uploads/195609-.pdf

[4] wiseGEEK, "What is Generative Grammar?" [Online]. Available: http://www.wisegeek.com/what-is-generative-grammar.htm.

[5] Serhiy Horelov, "Evrystychnyy analiz hramatyky" (Evrystychnyy analiz bud'-yakoyi movy) [Online]. Available: http://www.grammcheck.org/.

[6] V. M. Hlushkov, H. E. Tseytlyn, E. L. Yushchenko, "Alhebra. Yazyky. Prohrammyrovanye". Kyiv, USSR.: Nauk. dumka, 1974. 328 p.

[7] O. F. Fedorenko, S. M. Sukhorol's'ka, O. V. Ruda, "Osnovy linhvistychnykh doslidzhen = Fundamentals of Linguistic Research". Lviv, Ukraine: Tsentr, I. Franko's LNU, 2009. 296 p.



Stanislav Velykodniy – doctoral (Dr. hab.) student, Ph.D. in Computer Science, Associate Professor (Ukraine).

Born December 6, 1981 in the city of Kharkiv. In 2010 he moved to Odessa.

Education.

In 2004 I graduated from the Ukrainian Engineering Pedagogical Academy with a degree in "Automated Control Systems by Industrial Installations" and obtained the qualification of the engineer-educator-researcher.

In 2007, she entered the full-time postgraduate study at Kharkiv National University of Radio Electronics and in 2009 she graduated with a postgraduate degree in defense of her Ph.D. thesis in the specialty "CAD-Systems".

In 2017 he graduated from doctoral studies at the National University "Odessa Maritime Academy".

Scientific and pedagogical activity.

From 2002 to 2010 he worked at various scientific and pedagogical positions of higher educational institutions of Kharkiv.

From 2010 till present time (as of 2017) he worked as associate professor at higher educational institutions of Odessa, in particular at the National University "Odessa Maritime Academy" (since 2010) and the Odessa State Environmental University (from 2013).

Scientific work.

More than 80 scientific and teaching works, including: about 30 scientific articles, 3 certificates of authorship, 2 patents for utility models, 3 textbooks, 2 monographs.

Married. Hobbies: physics, construction, floriculture, cycling.

Olena Tymofieieva – postgraduate student, Department of information technologies, M.Sc. in Computer Science (Ukraine).

Was born March 10, 1985 in Odessa.

In 2000, 2001 graduated from music school violin and piano No. 8 and 2. Got a deed on drawing. Finished teaching courses in disciplines of "Robot and psychology with the children of problem families"; worked as a volunteer with homeless children.

In 2003 graduated from College with a degree "Secretary reviewer" and worked in the specialty.

In 2010 graduated from the Odessa State Environmental University.

Since 2011 worked as a teacher in higher education on computer disciplines. From 2013 there is gratitude and certificates for training of children to the regional Olympiad in Informatics and computer engineering.

In 2014 to 2016 were preparing children for regional Olympiad on computer engineering took the second place and first place.

In 2015 for examination of cabinet on computer disciplines got a deed for a III place in area.

Married. Hobbies: music, sports, construction.

American Scientific Journal
№ (17) / 2017
Vol.1

Layout man: Mark O'Donovan
Layout: Catherine Johnson

Address: 90 st. – Elmhurst AV, Queens, NY, United States
Web-site: http://american-science.com
E-mail: info@american-science.com

Copies: 1000 copies.
Printed in 90 st. – Elmhurst AV, Queens, NY, United States