*Bull. Nov. Comp. Center*, Comp. Science, 28 (2008), 31–42


# On the problem of computer language classification

I. S. Anureev, E. V. Bodin, L. V. Gorodnyaya,
A. G. Marchuk, F. A. Murzin, N. V. Shilov

*To Svyatoslav Lavrov textbook*
*"Universal Programming Language (ALGOL-60)"*
*used to teach a generation of programmers*
*in the Soviet Union for a decade.*

**Abstract.** During the semicentenial history of Computer Science and Information Technologies, several thousands of computer languages have been created. Computer language universe includes languages for different purposes: programming languages, specification languages, modeling languages, languages for knowledge representation, etc. In each of these branches of computer languages it is possible to track several approaches (imperative, declarative, object-oriented, etc.), disciplines of processing (sequential, non-deterministic, parallel, distributed, etc.), formalized models (from Turing machines up to logic inference machines). Computer languages have a different potential to be integrated to different models of software process and information processing. The development of computer languages influences the development of new and evolution of existing programming systems and information technologies. On the contrary, the demand for new software and information technologies leads to design of new computer languages. Computer paradigms are the basis for classification of the computer languages. They are based on joint attributes which allow us to distinct and select a certain branch in the computer language universe. Currently the number of essentially different paradigms has reached several dozens. Study and precise specification of computer paradigms (including new ones) are called to improve the choice of appropriate computer languages for new software projects and information technologies. The paper presents a so-called "semantics and pragmatics view" – an approach to computer languages paradigms and classification, that is based on a unified formal semantics (Abstract State Machines) and an open ontology for pragmatists.

## 1. Classification problem

A computer language is any language designed or used for automatic "information processing", i.e. data and process handling and management. During the semicentenial history of Computer Science and Information Technology, several thousands of computer languages have been created. Due to the number of the existing computer languages, there is a necessity for their systematization and classification.

At the initial stage of programming and information technology history (years 1950-65), it was possible to classify computer languages chronologically with annotations a-là Herodotus "The Histories", i.e. including lists of authors, their purposes, personal related stories, etc. (Please refer to [37] for a story of this kind.) The matter is that at the first stage all computer languages were languages of imperative programming for von Neumann's computers.

But since the late 1960-s, the approach in the style of "Father of History" becomes unacceptable. Hereinafter the variety of computer languages includes not only programming languages but also specification languages, data representation languages, etc. Some of these branches include not only imperative, but also declarative languages (functional in particular). Between the middle of 1970-s and the early 1980-s, some new approaches to computer language design appeared (logical and object-oriented, for example). In this period classification of computer languages could be done

- either in "Linnaeus style" as a taxonomy tree (i.e. Kingdom – Phylum – Class – Order – Family - Subfamily – Genus – Species ),
- or in "Mendeleyev style" as a "periodic table", where the rows represent classification of purpose (programming, specification, data representation, etc.), and the columns represent classification of approaches (imperative, functional, logical, etc.).

However, the 1990-s and the beginning of a new millennium became the time of rapid growth of existing and new branches of computer languages. For example, knowledge representation languages, languages for parallel/concurrent computations, languages for distributed and multi-agent systems, etc. Each of new computer languages has its own syntax (sometimes very specific), a certain model of information processing (i.e. semantics or a virtual machine), and its pragmatics (i.e. the sphere of application and distribution). The process of rapid generation of new computer languages will continue till new spheres of human activities will be computerized. So, from classification viewpoint, the situation in computer languages radically differs from natural sciences: in biology or chemistry the situation is static, while in computer languages the situation is rather dynamic. Due to this argument, classification approaches adopted in Natural Science cannot be adequately applied to computer languages.

At the same time, classification of already developed and new computer languages is a very important problem for computer science, since it could benefit software engineering and information technology by a sound framework for computer language choice for components of new program and information systems. Therefore a modern and practical classification of computer languages should be based not only on the analysis of code-generation mechanisms, productivity, reliability and efficiency, but also on the sphere

of application of languages, i.e. on their pragmatics. Also, a modern classification should be (whenever possible) steady and opened for changes, when new branches, approaches, or models of computer languages appear.

## 2. Computer language paradigmization

We call the alternative approaches to information processing, accumulated and fixed in the form of computer languages, the paradigms of computer languages. Computer paradigms are a basis of classification of computer languages, a set of the general attributes which allow us to specify a certain branch in the variety of computer languages. Characteristics of paradigms at the level of their basic means for information processing could provide designers, developers and users of software and information systems with a sound basis for comparison of systems, estimation of their potential, usability, etc. [13, 28]. Paradigmization means characterization of paradigms.

Robert Floyd was the first who had explicitly used the concept of "paradigm" in relation to computer languages. In particular, in his Turing Award Lecture [12] in 1978, he applied this concept only to programming languages. He referred to Thomas Kuhn's well-known book [17], published only 8 years before in 1970. According to T.Kuhn, the term "paradigm" means a method, an approach to formulation of problems and the ways to solve them. In modern philosophy the word paradigm is used as a pattern in any scientific discipline or other epistemological context. The word "paradigm" itself has Greek origin and means "example" or "pattern". Initially it was used in linguistics for classes of elements with similarities.

The importance of study of computer languages and their paradigmization is widely recognized in Computer Science and Information Technology. In particular, some information about the most known paradigms can be found in *The Encyclopedia of Programming Languages* [44] and in a special journal *Computer Languages* published by Elsevier. Many prominent scientists contributed to paradigmatization of computer languages: Edsger Dijkstra [6, 7], Robert Floyd[11, 12], Antony Hoare [7, 14, 15, 16], John McCarthy [21], Bertrand Meyer [22, 23], Robin Milner [24, 25, 26], Niklaus Wirth [14, 41], and others. We also would like to mention the contribution to the field made by A. P. Ershov [8, 9, 10] and S. S. Lavrov [19, 20] in the Soviet Union.

Currently, the number of essentially different paradigms only in programming languages is already close to several dozens (see, for example, the list of "programming paradigms" in [42]). And though there are rather small groups of computer languages (e.g., Hardware Description Languages), many groups are "crowded" (e.g., Specification Languages) and others go through the period of explosion and migration (e.g., Markup Languages).

The development of computer languages influences the design of new

programming systems and information technologies and evolution of current ones. On the contrary, appearance of new models of computation, and evolution of programming systems and information technologies lead to appearance of new computer languages. Many new parallel programming languages, languages of specification and modeling of multiagent systems, languages for object-oriented modeling and design, languages of knowledge representation, etc., appearing over the past 10-15 years, do not fit in the current classification and require its essential revision.

In spite of active research on systematization and classification of computer languages, experts have difficulties in putting some languages into one particular paradigm (e.g., Eiffel [23], Erlang [2], Python [36], etc.). There is a demand for paradigms for Language Of Temporal Ordering Specification LOTOS [39], Business Process Modeling Notation BPMN [34], Unified Medical Language System UMLS, [38] and many others.

## 3. Semantic and pragmatic view

For natural and artificial languages (including computer languages), the terms "syntax", "semantics" and "pragmatics" are used to categorize descriptions of language characteristics. Syntax is an orthography of a language. The meaning of syntactically-correct constructs is provided through language semantics. Pragmatics is a practice of use of meaning-bearing syntactically-correct constructs. Therefore the approach based on "specific features" of syntax, semantics and pragmatics could be natural for specifications of paradigmization and classification of computer languages. In the next paragraphs we discuss how this approach shows its worth in programming languages.

Pragmatics or programmer's practice was chosen by R. Floyd as a basis for development of the existing programming languages and design of new ones. He said in his Turing lecture [12]:

> "To the designer of programming languages, I say: unless you can support the paradigms I use when I program, or at least support my extending your language into one that does support my programming methods, I don't need your shiny new languages [...]"

It is possible to draw a conclusion that R. Floyd considered programmer's practice as a basis of paradigmization of programming languages. If we recall that the deterministic structured programming paradigm [7] prevailed up to the end of 1970-s, it becomes clear why nondeterministic structured programming [6] and concurrent and parallel communicating processes [15, 24, 35] emerged right at the end of this decade as new paradigms. At this time multitasking turned from a marginal problem (synchronization between

single CPU and few peripheral devices) into an actual problem of system and industrial programming.

Speaking about pragmatics of programming languages, it is natural to discuss the so-called "esoteric programming languages" [45, 46]. These languages are designed to validate some theoretical concepts or models of computation without any care about practice of programming. So, for example, the "programming language for orangutans" Ook! [47] uses the alphabet of three "symbols", represented by the word "ook" (which can be pronounced even by orangutans) with appropriate emotional tinge: "Ook?", "Ook." or "Ook!". In fact, Ook! is a structured Turing-complete programming language for counter machines (Minsky register machines) [27]. Thereby, reckoning a programming language among esoteric ones is first of all based on its pragmatics (as a practically useless language) and is also supported by its semantics – a special model of the "process of computation".

The semantic approach is not restricted by esoteric programming languages. The definitions of functional programming languages LISP [21] and ML [26] are based on semantics of these languages in terms of $\lambda$-calculus – the applicative equational theory of functions originated by Alonzo Church.

On the contrary, classification based on syntax has practically lost any value by virtue of the development of effective and powerful translators. Certainly, it is very important for a compiler implementation whether a particular language has regular, context-free or context-sensitive syntax. But the main thing for a programmer is flexibility and naturalness (from a human standpoint) of syntax of a language (including a reasonable portion of so-called "syntactic sugar") rather than the formal characteristics of its grammar.

It follows from the above discussion that the semantic and pragmatic view could be a reasonable approach to classification of programming languages. So it is rather natural to try to expand this approach to classification of computer languages. Let us discuss below what should be done for this expansion to be successful.

## 4. Unifying formal semantics

The role of formal semantics in the programming language paradigmization and classification of programming languages is very important [40]. The major problem with semantics of computer languages is a different formalism and a different level of formalization adopted for particular languages. For example,

- the functional language LISP has been founded on the base of a very precise denotational semantics in terms of $\lambda$-calculus,
- a structured subset of a high-level imperative language Pascal has an

operational and axiomatic semantics that are consistent with respect to each other [14],

- but formal semantics for a representative subset of an imperative macroassembler C is still a research challenge [33, 32].

This difference makes extremely hard to compare semantics of different computer languages.

Nevertheless, a sound formalism unification and more rigorous comparison of formal semantics are very essential for better classification of computer languages and a proper paradigm definition. The problem can be solved in 2 stages as follows.

- Two-dimensional stratification of sample[1] computer languages. Each of these languages should be stratified in levels and layers.
  - The level hierarchy could be a human-friendly semantic representation. It should comprise 2-3 levels that may be called educational (or learner's), basic (or user's), and professional. They are either dialects or subsets of the language. The educational level should be a dialect of the language for a first-time study of its basic concepts and features; it may be a subset of the language for casual users of the language. The basic level should be a subset for regular users of the language, it requires some skills and experience. The professional level is the language itself, it is intended for advanced and experienced users. Sometimes educational or basic level could be omitted. For example, the Ontology Web Language (OWL) [48, 49, 50, 51, 52, 53] comprises OWL-DL (basic level) and OWL-full (professional level).
  - The layer hierarchy is a formal-oriented semantic representation. It may comprise up to 3 layers for the basic level and (optionally) for other levels. These layers may be called kernel, intermediate and complete. The kernel layer should have a virtual machine semantics and provide tools for implementation of the intermediate layer; the intermediate layer in turn should provide tools for the complete layer. In both cases implementation should be of a transformation-type. Please refer to [30, 31, 32] for an example of a 3-layer hierarchy for programming languages of C-family.
- Extensive use of a powerful operational virtual machine semantics at the kernel layer of sample computer languages.
  - Specification of the kernel-layer computer languages by means of a unified and powerful semantic formalism like Abstract State Machines (AMS) [4] or Ontology Transition Systems (OTS) [1].

[1] We discuss what is a "sample computer language" in the next section 5.

- Animation (i.e. a sound and consistent implementation) of semantics of the kernel-layer of sample computer languages by means of type-free functional programming languages (LISP, for instance).
- Unfolding the animation to semantically sound and consistent compilers for the kernel-layer computer languages by means of bootstrapping technique typical for functional programming.

## 5. Evolving open pragmatics

Semantic and pragmatic view also assumes the development of a formalized framework for reasoning about computer languages pragmatics. But in contrast to a highly mathematical formal semantics, pragmatics rely upon a highly informal knowledge and experience of people and communities that are involved in compute language design, implementation, promotion, usage and evolution. In other words, we need to formalize expert knowledge about computer languages, about related concepts, and about relations between computer languages. This naturally leads to the idea to represent this knowledge about computer language pragmatics as an ontology.

"Ontology is the theory of objects and their ties. Ontology provides criteria for distinguishing various types of objects (concrete and abstract, existent and non-existent, real and ideal, independent and dependent) and their ties (relations, dependencies and predication)" [5]. Roughly speaking, an ontology is a partial formalization of a "knowledge" about a particular problem domain (computer languages, for instance). This "knowledge" could be an empirical fact, a mathematical theorem, a personal belief, expert resolution, shared or common viewpoint of a group. The most popular computer language for ontology representation is the Ontology Web Language OWL supported by WWW-consortium [48, 49, 50, 51, 52, 53]. The use of OWL (OWL-Lite and OWL-DL, in particular) for ontology representation provides an opportunity to use also Description Logic (DL) reasoners [3] for automatic consistency checking of an ontology. Consistency is very important for evolving ontologies that change in time, and for open ontologies that are open for editing. (A good example of an evolving and open ontology is Wikipedia, the free encyclopedia [43]).

In the proposed ontology for pragmatics of computer languages, objects should be computer languages (including their levels and layers that are described in the previous section 4), concepts (in terms of DL) or classes (in terms of OWL) – collections of computer languages that share a joint paradigm, ties (roles in terms of DL or properties in terms of OWL) – relations between computer languages. For example, LISP, PROLOG, SDL, LOTOS, BPMN, UMLT, as well as C, C-light and C-kernel, OWL-Lite, OWL-DL and OWL-full should be objects of the ontology.

We have already discussed a number of examples of concepts/classes in the proposed ontology: "functional languages", "specification languages", "executable languages", etc. All listed examples may be elementary concepts/ classes. Non-elementary concepts/classes can be constructed by different means that are supported by OWL and DL, in particular, by means of standard set-theoretic operations "complement", "union" and "intersection". For example, the concept "executable specification languages" is the intersection of two concepts – "executable languages" and "specification languages".

But the proposed ontology should have a special elementary concept/class for "sample languages" that comprises few (but one at least) representatives for every elementary concept/class. Of course, all elementary concepts/classes (including "sample languages") should be formatted on the base of expert knowledge and open for editing. A special feature of the proposed ontology should be the following constraint: every legal non-empty concept/class should contain a sample language (one at least). A background intuition is straightforward: if an expert can not point out any representative example of a paradigm, then the paradigm should be empty.

Roles/properties in the proposed ontology may be very natural also: "is dialect of", "is layer of", "uses syntax of", etc. For example: "REAL is dialect of SDL", "C-light is layer of C", "OWL uses syntax of XML". All listed examples are elementary roles/properties. The standard relational algebra operations "inverse", "complement", "union", "intersection", "composition" and "transitive closure" can be used and are meaningful for construction of new roles/properties. For example, "uses syntax of dialect of" is the composition of "uses syntax of" and "is a dialect of": REAL executional specifications [29] "uses syntax of dialect of" SDL [39].

Universal and existential quantifier restrictions that are used in OWL and DL for construction of new classes/concepts also could get a natural and useful meaning. An example of the existential restriction (in DL notation): a concept $\exists\ (uses\ syntax\ of\ .\ ((markup\ language)\ \sqcap \neg\{XML\})$ consists of all computer languages that are markup languages but do not use syntax of Extensible Markup Language XML [54]; an example of a language of this kind is LaTeX[18]. An example of the universal restriction and terminological sentence (in DL notation also) follows: a sentence $XML:\ \forall\ (is\ dialect\ of)\ .\ \neg(\{ML\})$ expresses the fact that XML is a dialect of any computer language but the functional programming "Meta Language" ML [26].

We would like to emphasize that the proposed ontology for pragmatics of computer languages should be an open evolving ontology, but with a semi-automatic maintenance by a group of experts, provided with automatic DL-based consistency checking tools.

We believe that the proposed semantics and pragmatics approach to clas-

sification of computer languages will provide researchers and engineers with a sound and easy to maintain and update framework for the new language design and an appropriate language choice for new software and information technology projects.

## References

[1] Anureev I.S. Ontological Transition Systems // Bull. Novosibirsk Comp. Center. Ser. Computer Science. – Novosibirsk, 2007. – Iss. 26. – P. 1–17.

[2] Armstrong J. Programming Erlang. Software for a Concurrent World. – Pragmatic Programmers, 2007.

[3] Baader F., Calvanese D., Nardi D., McGuinness D., and Patel-Schneider P., editors. The Description Logic Handbook: Theory, Implementation and Applications. – Cambridge University Press, 2003.

[4] Börger E. and Stärk R. Abstract State Machines: A Method for High-Level System Design and Analysis. – Springer-Verlag, 2003.

[5] Corazzon R. Ontology. A Resource Guide for Philosophers. – Available at `http://www.formalontology.it/`.

[6] Dijkstra E.W. A Discipline of Programming. – Prentice-Hall, 1976.

[7] Dahl O-J., Dijkstra E.W., Hoare C.A.R. Structured Programming. – Academic Press, 1972.

[8] Ershov A.P. Algorithmic Programming Languages // Vestnik Akademii Nauk SSSR. – 1968. – N 3. – P. 58-63. (In Russian)

[9] Ershov A.P., Pokrovsky S.B. Evolution of Programming Languages // 2-nd Soviet Conf. on Operational Research, Petrozavodsk, May 10-14, 1976. – P. 39-54. (In Russian)

[10] Ershov A.P. Mixed Computation: Potential Applications and Problems for Study // Theor. Comput. Sci. – 1982. – Vol. 18., N 1. – P. 41-67.

[11] Floyd R.W. Assigning meanings to programs // Proc. Symp. Appl. Math. American Math. Society. – 1967. – Vol. 19. – P. 19–32.

[12] Floyd R.W. The paradiggms of Programming // Communs ACM. – 1979. – Vol. 22. – P. 455–460.

[13] Gorodnyay L.V. Programming Paradigms // Internet University of Information Technology. – Available at http://www.intuit.ru, 2007. (In Russian)

[14] Hoare C.A.R., Wirth N. An Axiomatic Definition of the Programming Language PASCAL // Acta Informatica. – 1973. – Vol. 2, N 4. – P. 335–355.

[15] Hoare, C. A. R. Communicating Sequential Processes. – Prentice Hall, 1978.

[16] Hoare C. A. R. The Verifying Compiler: A Grand Challenge for Computing Research // Lect. Notes Comput. Sci. – 2003. – Vol. 2890. – P. 1–12.

[17] Kuhn T.S. The structure of Scientific Revolutions. – Univ. of Chicago Press, 1970.

[18] Lamport L. LaTeX: A document preparation system: User's guide and reference. – Addison-Wesley Professional, 1994.

[19] Lavrov S.S. Universal Programming Language (ALGOL 60). – Nauka Publishers, 1972. (In Russian)

[20] Lavrov S.S. Basic Concepts and Constructs of Programming Languages. – Finance and Statistics Publishers, 1982. (In Russian).

[21] McCarthy J. Recursive Functions of Symbolic Expressions and Their Computation by Machine // Communs ACM. – 1960. – Vol. 3, N 4. – P. 184–195.

[22] Meyer B. Introduction to the Theory of Programming Languages. – Prentice Hall, 1990.

[23] Meyer B. Eiffel: The Language. – Prentice Hall, 1991.

[24] Milner R. A Calculus for Communicating Systems. – Lect. Notes Comput. Sci. – 1980. – Vol. 92. – 171 p.

[25] Milner R. Communication and Concurrency. – Prentice Hall, 1989.

[26] Milner R., Tofte M., Harper R., MacQueen D. The Definition of Standard ML (Revised). – MIT Press, 1997.

[27] Minsky M. Recursive Unsolvability of Post's Problem of 'Tag' // Annals of Mathematics. – 1961. – N 74(3). – P. 437–453.

[28] Nepejvoda N.N. Styles and methods of programming. – Internet University of Information Technology. – 2005. (In Russian)

[29] Nepomniaschy V.A., Shilov N.V., Bodin E.V., Kozura V.E. Basic-REAL: integrated approach for design, specification and verification of distributed systems // Lect. Notes Comput. Sci. – 2002. – Vol. 2335. – P. 69–88.

[30] Nepomniaschy V.A., Anureev I.S., Dubranovskii I.V., Promsky A.V. Towards verification of C# programs: A three-level approach // Programming and Computer Software. – 2006. – N 32(4). – P. 190–202.

[31] Nepomniaschy V.A., Anureev I.S., Mihailov I.N., Promsky A.V. Towards Verification of C Programs. C-Light Language and Its Formal Semantics // Programming and Computer Software. – 2002. – N 28(6). – P. 314–323.

[32] Nepomniaschy V.A., Anureev I.S., Promsky A.V. Towards Verification of C Programs: Axiomatic Semantics of the C-kernel Language // Programming and Computer Software. – 2003. – N 29(6). – P. 338–350.

[33] Norrish M. C formalised in HOL: PhD Thesis. – 1998. – (Tech. Rep. / Univ. of Cambridge, Computer Laboratory; N 453).

[34] Roj J., Owen M. BPMN and Business Process Management Popkin Software, 2003. – Available at
http://www.bpmn.org/Documents/6AD5D16960.BPMN_and_BPM.pdf

[35] Roscoe A. W. The Theory and Practice of Concurrency. – Prentice Hall, 1997.

[36] The Python Language Reference Mannual (version 2.5) / Ed. by G. van Rossum, F.L. Drake. – Network Theory Limited, 2006.

[37] Ritchie D.M. The development of the C language // ACM SIGPLAN Notices. – 1993. – Vol. 28, N 3. – P. 201–208.

[38] Smith B., Kumar A., Schulze-Kremer S. Revising the UMLS Semantic Network / Ed. by M. Fieschi, et al. – Medinfo, Amsterdam, IOS Press, 2004. – 1700 p.

[39] Using Formal Description Techniques – An Introduction to Estelle, LOTOS and SDL /Ed. by K. J. Turner. – John Wiley and Sons, 1993.

[40] Turner R., Eden A.H. Towards a Programming Language Ontology // Computation, Information, CognitionThe Nexus and the Liminal. – Cambridge Scholars Press, 2007. – P. 147–159.

[41] Wirth N. Algorithms + Data Structures = Programs. – Prentice-Hall, 1976.

[42] Programming paradigm. From Wikipedia, the free encyclopedia. – Available at http://en.wikipedia.org/wiki/Programming_paradigm

[43] Wikipedia, the free encyclopedia. – Available at http://en.wikipedia.org

[44] The Encyclopedia of Programming Languages. – Available at http://hopl.murdoch.edu.au/

[45] Esoteric programming language. From Wikipedia, the free encyclopedia. – Available at
http://en.wikipedia.org/wiki/Esoteric_programming_language

[46] Esolang wiki. – Available at http://esolangs.org/wiki/Main_Page

[47] Ook! – Available at http://www.dangermouse.net/esoteric/ook.html

[48] Web Ontology Language (OWL) Use Cases and Requirements, W3C Recommendation, February 10, 2004. – Available at http://www.w3.org/TR/webont-req

[49] OWL Web Ontology Language Reference, W3C Recommendation, February 10, 2004. – Available at http://www.w3.org/TR/owl-ref

[50] OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation, February 10, 2004. – Available at http://www.w3.org/TR/owl-absyn

[51] OWL Web Ontology Language Overview, W3C Recommendation, February 10, 2004. – Available at http://www.w3.org/TR/owl-features

[52] OWL Web Ontology Language Test Cases, W3C Recommendation, February 10, 2004. – Available at http://www.w3.org/TR/owl-test

[53] OWL Web Ontology Language Guide, W3C Recommendation, February 10, 2004. – Available at http://www.w3.org/TR/owl-guide

[54] Extensible Markup Language (XML) 1.0 – Available at http://www.w3.org/TR/2006/REC-xml-20060816