УДК 3.378

TEACHING THEORY OF PROGRAMMING: METHODOLOGY AND BASIC NOTIONS

Nikitchenko N.S.,
Department of Theory and Technology of Programming,
National Taras Shevchenko University of Kyiv

Обгрунтовується включення в навчальні плани курсу "Теорії програмування". Визначаються його методологічні аспекти та основні поняття. Запропонований підхід призводить до простої структури курсу, його єдності з такими спорідненими дисциплінами як математична логіка та теорія алгоритмів.

An idea of including into computing curricula an integral course "Theory of Programming" is proposed. Its main methodological aspects and basic notions are discussed. Such an approach leads to a simple structure of the course, its integrity with such programming related courses as mathematical logic and computability theory.

Introduction

The Computing Curricula 2001 [1] treats programming as a main course for computer science specialists. But strangely enough we cannot find in this curriculum a course under the title "Theory of programming" which should present theoretical foundations of programming. Instead, a number of units split among different courses are proposed which cover theoretical aspects of programming. These are: Programming Paradigms, Programming Language Design, Syntax and Semantics, Formal Specification, etc. As a consequence, students lack a unified view on a general process of programming and their theoretical knowledge is primarily restricted to programming languages concepts. Therefore in this presentation we will advocate the idea of including into curricula an integral course "Theory of Programming" and present its main methodological aspects and basic notions.

1. Methodological Aspects of Theory of Programming

Speaking about certain theory we cannot avoid questions concerning its methodological foundations. This is also true for theory of programming. There can be several approaches to answer such questions. The first is ad hoc one which specifies methodological principles only when it is required by some special topics. The second is specialized one which specifies methodological principles of programming and uses them as a basis for theory development. And the third approach tries to integrate general (philosophical) methodological principles with special ones and present them in an explicit form.

Our approach is the third one. Strictly speaking this approach is based on a special branch of philosophy which is called gnoseology (or epistemology). Gnoseology is a theory of cognition, which aims to examine the nature of knowledge, its limits, how it is derived and how it is to be validated and tested. Thus, our approach to theory of programming can be also defined as a *gnoseology-based approach*. This is to be supported by the principle of gnoseology-based theory: *theory of programming should be developed according to the main principles, laws and methods of gnoseology*.

This principle is a weak form of a Hegel's idea of a theory as applied logic. Therefore we will also call theory of programming as *programology*.

Other general methodological principles can be explained by the following considerations. When we investigate some objects, the first idea that strikes us is numerous connections of objects with each other. It follows from this that any thing has a lot of facets (aspects) and that a thing itself can be considered as the totality of all its aspects. But we cannot immediately investigate this totality of aspects therefore we start with some aspects (chosen due to abstraction), then consider some other aspects with their relations to the first aspects (thus making concretization). Considered aspects should be essential ones and are chosen according to practical and theoretical criteria.

Despite the simplicity and coarseness of the above considerations they lead to the following principles of gnoseology.

Principle of universal connection: everything in the world is connected with something else.

Principle of development from abstract to concrete: development is definitely oriented change of the object (notion) from abstract to concrete (from simple to complex, from a lower level to a higher one, from the old to the new).

Triadic principle of development: one of the main schemes of development is specified as a development triad *thesis – antithesis – synthesis*.

Principle of unity of theory and practice: theory and practice should be considered as influencing each other. A variant of this principle is the principle of *union of logical and historical development*.

Let us make two notes: 1) the above are not all principles required for theory construction; we chose only those principles which demand a special attention in theory of programming; 2) all methodological principles are not absolute, they have a relative nature.

It follows from the above formulated principles that developing theory of programming we will construct a hierarchy of definitions (systems) of various abstractions levels and generality which reflect the main properties of programming.

Having formulated the main methodological principles we can now go to development of the notion (concept) structure of theory of programming.

2. Basic Notions of Theory of Programming

In constructing theory of programming we will use notions of three types:

1) categories;
2) scientific notions;
3) formal notions.

Categories can be considered as the most general features, characteristic of things. They are studied in gnoseology. Examples are: subject and object; abstract and concrete; internal and external; quality, quantity, and measure; essence and phenomenon; individual, general, and particular; whole and part; content and form; cause and effect; etc. Scientific notions present special features of thing, reflecting a domain under investigation. Formal notions have formalized definitions which specify the notion constructs and the rules of their interpretations. In programming formal notions form a basis for automatization of various phases of programming process.

In theory of programming these types of notions are explicated with orientation on computing (sphere of informatization) as applied domain (practice). This statement is called *a principle of programming orientation*. This principle is a specialization of the principle of unity of theory and practice. The considered situation can be presented in Figure 1.

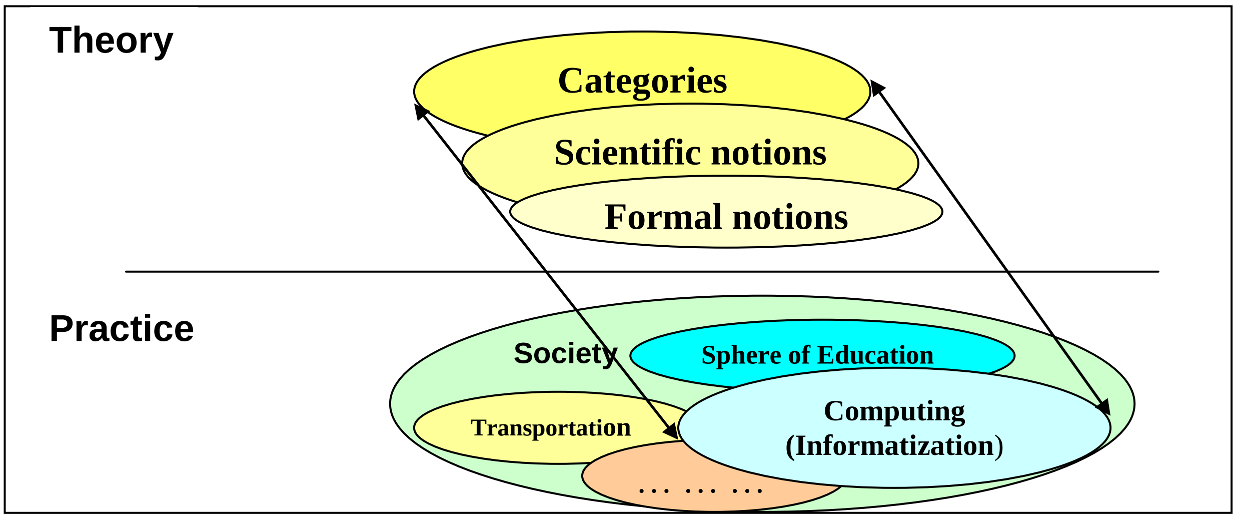

Figure 1. *Explication of notions should be oriented on computing (sphere of informatization)*

Now we can start developing the main (scientific) notions of theory of programming. The following development triads are specified [2]:

1) user – problem – program (*program triad*),
2) program – process of execution – user (*execution triad*),
3) problem – program – process of programming (*programming triad*).

In aggregate these basic notions form the *programming pentad* (Figure 2). All notions in this pentad are connected with various relations. The aim of theory of programming is to study these notions in integrity of their essential aspects.

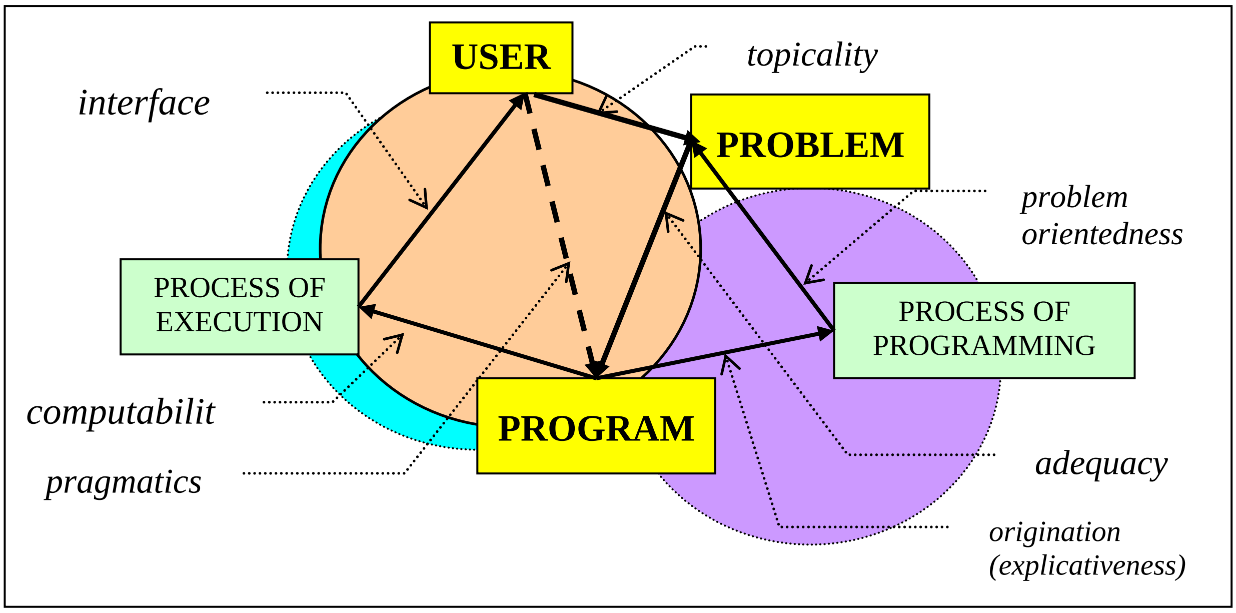

Figure 2. Pentad of the main basic notions of programming

On the next level of abstraction we should elucidate all notions of the programming pentad [2]. But here we present only the notion of program (Figure 3).

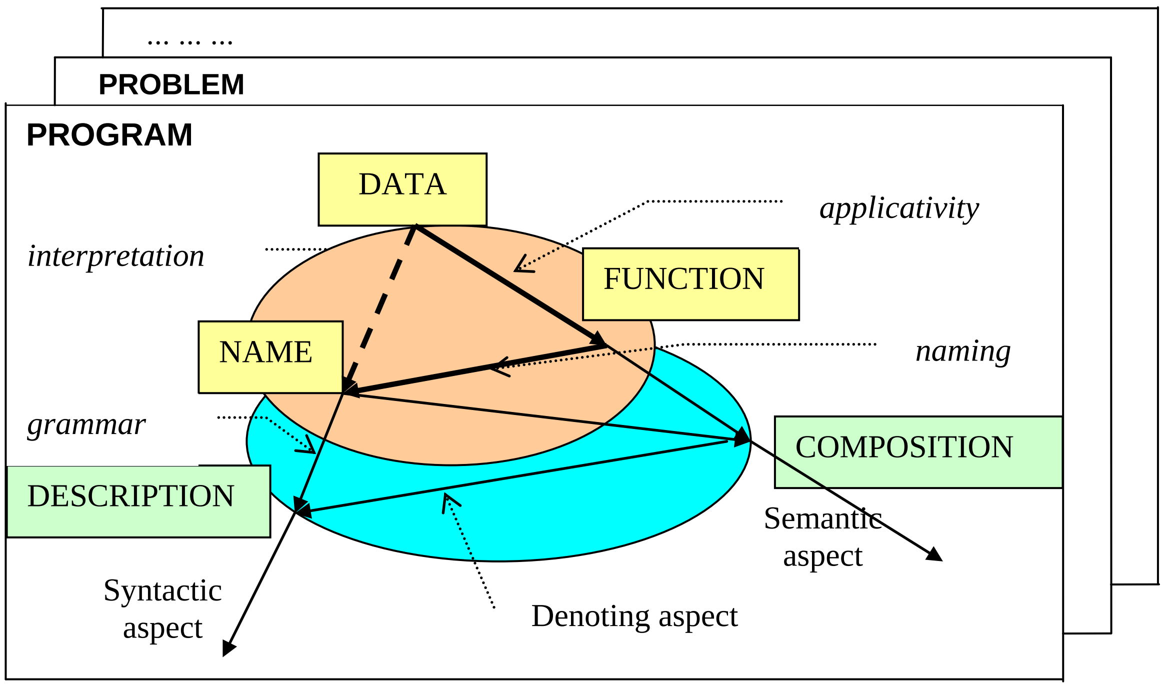

Figure 3. *Pentad of the main basic program notions*

This diagram presents

1) the triad of the main program notions (*data – function – name*),

2) the pentad of the main program notions (data – function – name – composition – description).

All notions in this pentad are connected with various relations.

Our aim is to study and formalize all these notions. To do this we should formulate some specific methodological principles. First, we involve into consideration intensional and extensional aspects of notions.

The distinction of these aspects was known from ancient times. Aristotle in his *Posterior Analytics* already specified this distinction though he did not use explicitly the above terms. Many logicians since that time examined the questions of intension/extension dichotomy. A second wind to these investigations was given by G. Frege with his famous *meaning triangle* and R. Carnap with his *intensional/extensional investigations*. Though the dichotomy under discussion was studied primarily in logic, semiotics, and linguistics, last years it also was considered in informatics (computer science). In such its branches as artificial intelligence, data and knowledge bases, etc., the intensional and extensional aspects play an important role. Still, we can admit a special and restricted usage of these notions, especially in formalized (mathematical) theories. The reason of such a situation is clear: in the last century mathematical development was based on an extensional approach. This approach was supported by the very first axiom of set theory – the extensionality axiom: two sets are equal if they consist of the same elements. N. Bourbaki in his numerous treatises aimed to write a thorough unified account of all mathematics based on (extensional) set theory. We can see now more and more facts when a pure extensional orientation becomes restrictive for further development of mathematics and informatics. Therefore a more thorough attention must be paid to intensional aspects. Thus, we can formulate the following principle.

Principle of *integrity of intensional and extensional aspects*: the notions should be presented in the integrity of their intensional and extensional aspects. The intensional aspects in this integrity play a leading role.

Objects are presented by their descriptions, thus letting us to speak about semantic and syntactic aspects as projections of categories content and form. Principle of a *leading role of semantics over syntax* states that syntax can be considered as dependent upon semantics, though finally the both aspects should be studied in their integrity specified by denoting aspects.

Up to now we did not speak about objects and descriptions structure. Taking into account an object (and its description) structuring we can formulate the *compositionality principle*: objects are constructed from other objects with the help of compositions. As a consequence, semantics of a description is a composition of semantics of its constituent parts (Frege's compositionality principle).

And at last (but not least) we formulate the *nominativity principle*: nominative (naming) relations are the primary constructs.

Let us admit that the formulated principles do not cover many epistemic (methodological) principles that should be used in theory of programming. They rather emphasize those moments to which more attention should be paid while investigating notions in theory of programming.

On the next level of abstraction we can develop and formalize the notions of the program pentad: data, functions, names, compositions, and descriptions. Here we present only a diagram for data development (Figure 4).

This notion is developed according to the following triads of categories:

- whole (W) – parts (P) – synthesis (H as Hierarchy)
- abstract (A) – concrete (C) – synthesis (S).

Thus, we get 9 levels of data types.

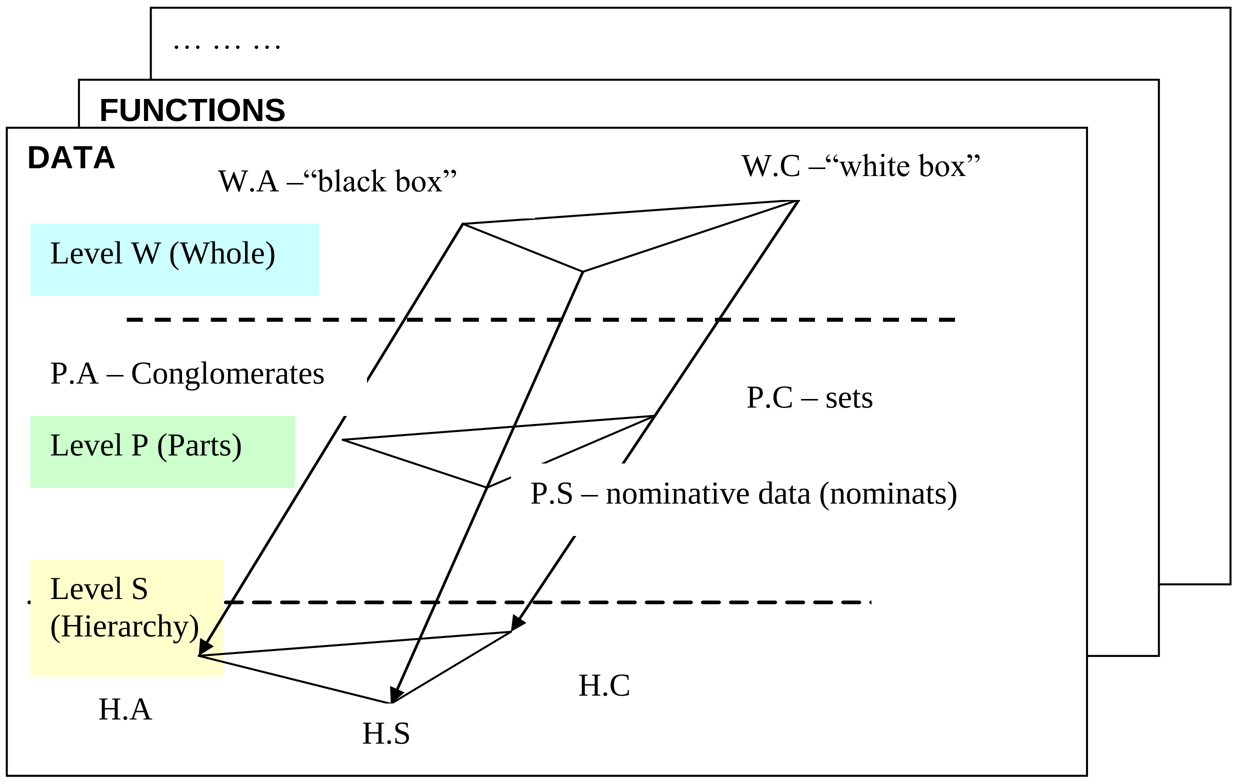

Figure 4. *Diagram of development of the notion of data*

The developed notion of data permits to say that data structures used in programming (records, arrays, files, relations, trees, etc.) can be specified as concretizations of the considered types of data. And the nominats (nominative data) play a main role in data processing.

The constructed hierarchical system of notions is a subject for formalization on a basis of the formulated methodological principles. In particular, a notion of program can be formalized as a composition nominative system which consists of semantic, syntactic and denoting systems. Such composition nominative systems can be constructed on various levels of abstractions in integrity of their intensional and extensional aspects [3]. Such systems specify various program aspects. Let us consider the main of such aspects.

3. Main Aspects of Programs

Traditionally, the main aspects of programs are specified as semiotics aspects: *pragmatic, semantics*, *syntax*.

The system of program notions constructed here permits to define a more elaborated system of program aspects. In the first approximation this system which we call a system of *essentialistic program aspects* consists of two types of aspects:

1) external program aspects: adequacy, pragmatics, computability, and origination;
2) internal aspects: semantics, syntax, and denoting relation.

It is important to integrate the external and internal aspects of programs thus giving indeed a concrete definition of the notion of program.

Here we will not go into further details (see [2, 3]). Let us only admit that the presented system of program notions/aspects and their formalization via composition nominative systems have rather general nature and can be also used in such disciplines as mathematical logic, computability theory, and universal algebra [3, 4].

Conclusion

The proposed methodological approach for developing theory of programming seems to be useful in teaching due to the following:

1) it is based on a small number of methodological principles thus specifying a clear structure of the theory;
2) it proposes a number of theory levels starting from simple to more elaborate thus giving possibility to present more complex concepts on later stages of education;
3) it leads to simple formal program models thus permitting their thorough investigation;
4) it can be used also in teaching logic and computability thus presenting a real integrity of programming related courses.

REFERENCES

1. *Computing Curricula 2001, Computer Science*.– IEEE Computer Society Press and ACM Press, December 15, 2001.– 236 p.
2. *Nikitchenko N.S.* A Composition Nominative Approach to Program Semantics.– Technical Report IT-TR: 1998-020.– Technical University of Denmark.– 1998.– 103 p.
3. *Nikitchenko N.S.* Abstract Computability of Non-deterministic Programs over Various Data Structures // Perspectives of System Informatics (Proc. of Andrei Ershov Fourth Int. Conf.).– Novosibirsk: A.P. Ershov Institute of Informatics Systems, 2001.– P. 246–251.
4. *Nikitchenko N.S., Shkilniak S.S.* Basics of Mathematical Logic.– Kiev university, 2006. – 246 p. (In Ukrainian)