Annales Univ. Sci. Budapest., Sect. Comp. 52 (2021) 7–11

TO THE MEMORY OF PROFESSOR LÁSZLÓ VARGA

(1 May 1931 – 19 September 2020)

László Kozma (Budapest, Hungary)

László Varga was born in Sárszentlőrinc in Hungary, in 1931. He started his higher education studies at Eötvös Lorand University in applied mathematics and received a diploma in 1956. He started his scientific carrier in mathematics at Central Research Institute for Physics in Budapest. He earned the so called cadidate degree in Hungarian Academy of Sciences (HAS) in 1967 and the doctor of science degree in mathematics in HAS in 1977. He continued his scientific carrier and started his higher educational activities at Faculty of Science at Eötvös Loránd University during seventies. He was a visiting professor at Brown University Providence in USA for one year in 1978-79. He had several so-called aspirants and PhD students who defended successfully their theses. He was the first director of group of informatics departments at Faculty of Science at Eötvös Loránd University and was the head of Department of General Computer Science for four years, 1986-1989. László Varga was one of the leaders of a team who elaborated the study programs of programming mathematics profession and the program designer profession. He had several high level lectures including theory of programming, software technology etc. His research accomplishments in computer science and activity in informatics education have been recognized by several merits including the Hungarian Order of Merit Knight's Cross (Civil) in 2001, the Hungarian Merit Officer's Cross (Civil) in 2009, the Eötvös József Prize in 2006, Kalmár Award by John von Neumann Computer Society, Szent-Györgyi Award. László Varga was awarded an honorary doctor of Eötvös Loránd University in 2010.

The results of László Varga's research work covering a wide field of theory and practice of programming prove very useful and important. These results range from clearing up the problems of machine coding, over analysing advantages and limits of structured and object-oriented programming, to problems of component oriented programming.

He was among the first Hungarian mathematicians who started to study the problems of programming of the first Hungarian made computer M-3 during end of fifties.

https://doi.org/10.71352/ac.52.007

One of the main problems of the software engineers is the rapid development of hardware. The development of software could not keep level up with it. These facts press the software engineers for frequently developing new software technologies and methods. It is well-known that one of the challenges of a large-system development is the need to regularly evaluate the emerging system from a number of different perspectives e.g. from manufacturability, safety, performance, ergonomics, modifiability interoperability, competitive position production cost, maintainability, reliability etc. These needs make the large-system development so difficult and risky. The constituents of a software development process are: analysing a given problem, specifying a new system, designing that system, building it and verifying that it was built as specified.

László Varga published several papers, books and lectures notes on how to increase reliability of a program developed by structured programming technology [Var-81, Var-89].

Beside the programs' structures László Varga was carefully studying the problems of data structures independently of their concrete representation. In his paper [Var-76] an extension of Vienna Definition Language (VDL) was used for defining the abstract syntax and semantics of data structures. The basic VDL data structures were discussed from the viewpoint of practical considerations, and the VDL graph, as a basic VDL data structure, and graph manipulation operators were introduced. The properties of the VDL graph were summarized in theorems. In an other paper, his purpose was to create a deductive technique for developing abstract programs systematically by stepwise refinement from given specifications using VDL. The VDL graph was defined as an abstraction of a class of data structures. The VDL graph specifies a connected graph which has one entry node at least, but may have several terminal nodes, and there must be a path from one entry node at least to a terminal node through every node in the graph. The deductive technique was illustrated by some examples e.g. by an abstract graph walk algorithm, by an application of VDL-graph for specifying a linkage editor and an inverse assembler model, respectively [Var-80].

Abstract data type is a fundamental concept of the specification of a software system including object-oriented systems and component oriented system as well. In his paper [Var-87] abstract data types are specified by algebraic specification and the appropriate concrete implementations of abstract types are defined by algebraic way, too. The notion of correct representation and the notion of correct implementation are defined. For a concrete data type which is a correct representation of the given abstract data type, it is shown that a correct implementation satisfies the semantic equations of the given abstract data type. On the other hand, if an implementation satisfies the semantic equations

of an abstract data type and the representation is correct then it is a correct implementation.

To generalize the above results: for implementing an abstract data type, we may choose an appropriate representation and give a concrete data type specification, so a so-called double specification is given for a data type. In this case a correctness of a concrete specification according to an abstract specification can be formulated, and sufficient conditions for correctness of three different double specifications can be presented [Var-87-88]. A double specification has an abstract and a concrete specification part. An abstract specification is to serve the user's view, and a concrete specification is to serve the implementer's view. In the case a double specification, a verification of the correctness of an implementation according to the concept which is in our mind about the given data type can be carried out in two steps. First we show that an abstract specification correctly reflects the concept [Var-83], and next we verify the correctness of a concrete specification according to the abstract one [Var-87, Var-87-88]. An abstract specification can be given by algebraic way or by pre- and post-conditions. A concrete specification can be given by algebraic way or by pre- and post-conditions or by procedures. The following pairs are relevant from point of view of double specifications:

Abstract:	**Concrete:**
algebraic	algebraic
algebraic	pre- and post-condition
pre- and post-condition	pre- and post-condition
pre- and post-condition	procedural

We can define an abstract data types with coarse granularity by algebraic way independently from any programming language. Pre- and post-condition specification and procedural one are language dependent specifications [Var-87-88].

In accordance with the above results, the designer of a software system can define a technology for creating correct implementation of an abstract data type step by step [K-V-2003]:

First the designer can specify a data type by algebraic specification with coarse granularity, and then he can show that the selected abstract specification correctly reflects the user's concept [Var-83]. Then the concrete implementation is specified by algebraic way or by pre- and post-conditions. The designer can verify the correctness of the concrete specification according to the abstract one [Var-87, Var-87-88].

Second step: The designer now selects the above concrete specification as the new abstract one, and selects a new concrete specification with finer gra-

nularity. The designer can prove the correctness of the new implementation [Var-87, Var-87-88].

This step can be repeated by as many times as it is necessary.

Finally, during the last step, the designer or the programmer can select the last concrete specification with pre- and post-conditions as the abstract one, and the concrete implementation can be specified by procedures with finest granularity. He can prove the correctness of the implementation [Var-87, Var-87-88].

Object-oriented programming is a methodology for describing modules, objects and classes of objects. Its essence is the subdivision of a system into objects which are integrated units of data and procedures [S-V-2001]. These units usually are concurrent objects. Concurrent objects can be instants of shared types. These shared types must also be specified, represented and implemented. The paper [K-V-93] presents a methodology with some examples for developing shared types to support intra-object concurrency. This means that an object can generate its answers for the other objects concurrently.

For developing large software systems, the compositional approach is generally used. A possible way of ensuring the correctness of independently developed components in this technology is the use of contracts which are formal agreements expressing each party's rights and obligations. Defining a pre-condition and a post-condition, or an invariant for a routine is a way to define a contract that binds the routine to its caller. This concept was later extended to component-based world. Component-based applications are generally distributed systems, and the verification of such systems is extremely complicated, usually done on two levels. In paper [D-K-V-2006] the behavioural inheritance contract driven environment was formally defined in the base of the Substitution Principle given by B. H. Liskov and J. M. Wing. A single method was given for proving the correctness of a subtype with respect to its super type in the case of two kinds of data type specification methods. The method is illustrated by some examples. Using type inheritance and contracts together not only increases the reliability of a program or a system, but also simplifies its verification procedure.

The central questions of the research work and the educational activities of László Varga were the conceptual understanding of software. He became one of the best-known researchers and professors in Hungarian informatics communities.

List of selected publications

Books and Lecture Notes

[Var-81] **L. Varga,** *Analysis and Synthesis of Programs* (Hungarian), Akadémiai Kiadó, Budapest, 1981.

[Var-89] **L. Varga,** *Theory of Programming Methods, I.-II.* (Hungarian), Tankönyvkiadó, Budapest, 1989.

[S-V-2001] **S. Sike and L. Varga,** *Object-oriented Modeling in UML – Exercises Library with Definitions* (Hungarian), Eötvös Loránd University, Faculty of Science, Budapest, 2001.

[K-V-2001] **L. Kozma and L. Varga,** *Data Type Classes – Definitions, Analysis, Examples* (Hungarian), Eötvös Loránd University, Faculty of Science, Budapest, 2001.

[K-V-2003] **L. Kozma and L. Varga,** *Theoretical Issues of Software Technology* (Hungarian), ELTE Eötvös Kiadó, Budapest, 2003.

Papers

[Var-76] **L. Varga,** Abstract syntax and semantics of data structures (Hungarian), *Alkalmazott Matematikai Lapok*, **2(1-2)** (1976), 41–55.

[Var-80] **L. Varga,** Synthesis of abstract algorithms, *Acta Cybernetica*, Szeged, **5(1)** (1980), 59–76.

[Var-83] **L. Varga,** On the verification of abstract data types, *Acta Cybernetica*, Szeged, **6(1)** (1983), 7–12.

[Var-87] **L. Varga,** Implementation of abstract data types with correctness proof, *Annales Univ. Sci. Budapest., Sect. Comp.*, **8** (1987), 109–118.

[Var-87-88] **L. Varga,** Study of the correctness of data type specifications (Hungarian), *Alkalmazott Matematikai Lapok*, **13(1-2)** (1987/88), 57–68.

[K-V-93] **L. Kozma and L. Varga,** A methodology for the development of shared object classes, *International Conference on Applied Informatics*, Eger, Hungary, 23–26 August 1993.

[D-K-V-2006] **Á. Dávid, L. Kozma and L. Varga,** On the correctness of data type classes based on contracts, *PU.M.A.*, **17(3-4)** (2006), 251–261.

L. Kozma
Faculty of Informatics
Eötvös Loránd University
H-1117 Budapest
Pázmány Péter sétány 1/C
Hungary
kozma@ik.elte.hu