LAUDATIO TO
Professor László Varga

Varga László professzor emeritus 1931-ben született Sárszentlőrincen, 1956-ban kapott oklevelet az ELTE-n alkalmazott matematikus szakon 1967-ben megszerezte a matematikai tudomány kandidátusa, majd 1977-ben a matematikai tudomány doktora akadémiai fokozatot. 1956-63 között az Optikai Kutató Laboratórium munkatársa, 1963-1979 között az MTA Központi Fizikai Kutatóintézet Számítástudományi Osztályának vezetője. Ezen idő alatt már az ELTE-n is oktatott. 1978-79-ben a Brown University Providence vendégkutatója. 1979-1996-ig az ELTE TTK Általános Számítástudományi Tanszékének tanszékvezető egyetemi tanára. 1986-1989-ig az ELTE TTK gazdasági ügyekkel foglalkozó dékán-helyettese.

Varga László szakmai irányítása mellett indult meg 1972-ben a programozó matematikus képzés az ELTE-n. Fáradhatatlan munkásságának és szakmai hozzáértésének köszönhetően ebből a szakból egy széles spektrumú informatikai képzési rendszer nőtt ki egyetemünkön. Így időközben a nappali programozó matematikus szak mellett megindult a képzés a nappali programtervező matematikus és az informatika tanár szakokon, az esti programozó és programtervező matematikus és a levelező számítástechnika tanári szakokon. A 80-as évek végén elindult a szakinformatikusok képzése is. Az ELTE Informatika Doktori Iskolájának megszervezése, zökkenőmentes beindítása nagyrészt Varga László professzor hatékony szakmai munkájának, szervezőkészségének köszönhető. Napjainkban a különböző informatikai szakokon tanuló hallgatók létszáma meghaladja a 2000 főt. Ezen hallgatói létszám oktatásának személyi és tárgyi feltételeit szinte a nulláról indulva kellett megteremteni. Ebben a menedzseri tevékenységben is döntő részt vállalt a tanszékcsoporton belül, majd az Informatikai Karon.

Oktatói tevékenysége során nagy gondot fordított a tehetséges hallgatók felkarolására, szakmai fejlődésük biztosítására. Oktatói és szervezői feladatai mellett tudatos tanszéképítő munkája eredményeként több tehetséges fiatal került a tanszékére. A fiatal kollegák áldozatkész szakmai menedzselését személyes konzultációk és tanszéki szemináriumok keretében végzi. Ezeken a szemináriumokon nevelkedett fel a tanszék oktatói állományának jelentős része. Számos egyetemi doktori, PhD és kandidátusi értekezés született az ő szakmai irányítása mellett.

Az Informatikai Tanszékcsoport vezetőjeként is sokat vállalt magára. Munkásságával jelentősen hozzájárult ahhoz, hogy az Informatikai Tanszékcsoport – a német egyetemek terminológiáját használva – fakultás szintű egységgé vált. Az ő munkája alapozta meg az Informatikai Kar létrejöttét az Eötvös Loránd Tudományegyetemen 2003-ban.

Varga László dékánhelyettesként is jelentős sikereket ért el a Természettudományi Kar közéletében. A gazdasági területekért felelős dékánhelyettesként sokat dolgozott azon, hogy a kar és rajta keresztül az egyetem struktúrája a dinamikusan változó követelményekhez minél jobban illeszkedjen. Az ELTE Egyetemi Tanácsának választott tagjaként is eredményesen dolgozott az egész egyetem érdekében.

Varga László egyetemi tanár tudományos tevékenysége, eredményei jelentősek. Számos dolgozata jelent meg neves hazai és külföldi fórumokon, többek között az absztrakt adattípusok elmélete és a programok bonyolultsága témakörökben. Az Akadémiai Kiadónál megjelent könyvei (Könyv-nívódíj) és egyetemi jegyzetei az oktatásunk nélkülözhetetlen részeivé váltak. Az objektum elvű programozásról írott könyveit, jegyzeteit széles körben használják a felsőoktatásban. Több nemzetközi konferencián volt programbizottsági tag, elnök, számos külföldi egyetemen tartott előadásokat. Több folyóirat szerkesztőbizottságának tagja. Számos eredményesen lezárult OTKA pályázatnak volt témavezetője. Tevékenyen részt vesz az ELTE Informatika Doktori Bizottságának munkájában. Az MTA különböző bizottságaiban is aktívan dolgozik jelenleg is. Munkásságát a Szent-Györgyi Albert-díj, az Eötvös József díj és az akadémiai könyv-nívódíj mellett a KFKI intézeti első díjjal, a Neumann János Számítógéptudományi Társulat Kalmár László emlékéremmel, az ELTE 1980-ban Kiváló Munkáért kitüntetéssel jutalmazta.

Professor Varga is an acknowledged professor of informatics. He wrote a lot of papers, mainly in the theory and practice of programming. We shall concentrate ourselves to his most interesting results, which we shall subdivide as follows:

1. Structured programming.
2. Object-oriented programming.

1. Structured programming

It is well-known that one of the challenges of a large-system development project 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 costs, maintainability, etc. These needs make large-system development so difficult and risky. In environments for software development the various phases for building an application are well-defined and distinct. The schema for compiled languages is the following. First you have to write a collection of modules, possibly with some interdependencies, and with some dependencies to predefined modules stored in libraries. Second you have to compile the modules, in order to generate machine code and to check type correctness of the modules in strongly type systems. Finally you have to link the various pieces of machine code together, using a global name space to resolve all cross-references. This is the traditional three-phase assembly of software.

In traditional, structured techniques, data is passive. Behaviours associated with a data structure are ‘hard-wired’ into the procedures by means of branching statements that examine some ad hoc characteristic of the data and decide how to act upon it. These branching statements bind input data with the procedures, the code of routines that operate on that data. Nevertheless, this procedural model has allowed us to build very good software systems. For instance the paper [Var-80] concentrates on the problem for developing an abstract program step by step from a given specification using VDL. This paper extends the Hoare’s method to VDL-statements. The proposed deductive technique is illustrated by the example of an abstract graph walk algorithms. VDL was a frequently used tools of software development process during seventies. For instance a general model of an inverse assembler can be defined by VDL. The inverse assembler translates programs in machine code form into assembly form. It is an important tool of program adaptation. In paper [N-V-76] the notion of machine code program, assembly program and inverse assembler are defined by tools of VDL. The

VDL is one of the oldest and best known metalanguages for writing operational definitions of the semantics of a programming language. The central aim is to define an abstract machine for interpreting abstract programs of the subject language. The machine interprets a program by passing through a sequence of discrete states, its mode of operation is independent of the subject language. The allowable state transitions are defined by a set of instruction definitions written in a special notion [Var-76]. Studying of the correctness of programs is a very important research area within the scope of programming methodology. From the second half of the sixties there has been great activity in this field. According to L. Varga's classification there are two different approaches to achieving program correctness. The aim of the *engineering approach* is to develop more efficient software tools and specify standards that can be used in the process of development of programs as a means to improve the reliability and reduce the software cost. This approach is very effective to validate programs in practice, but it is not capable to formally prove the correctness of a program. The *analytic approach* uses program verification methods to ensure that the desired program is correct with respect to its specifications. In case of the posterior verification of programs a program and its specification are given and our task is to prove that the program realizes exactly the mapping stated in the specification. However it turned out, that different programs which realize the same mapping may be essentially different from the point of view of the correctness proof. The difficulty of a proof depends on the complexity of the program. This led to the conclusion that the correctness of a program has to be established during its construction. This is the *constructive approach* to achieving program correctness. The methods initiated by the constructive approach have made a fundamental contribution to the *synthesis of programs* in extracting principles for deriving programs systematically from their specifications. These principles are formulated precisely enough to be carried out by an automatic synthesis system [Var-80]. Unfortunately these methods can be applied for generating only small systems, but from the theoretical point of view they are very useful.

From the second half of the seventies the examination of data structures is getting more and more important. In papers [Var-76, F-V-77] the definition problems of data structures, independently of their concrete representations, are discussed. The abstract concepts of data element and data structures are defined. VDL is used for defining the abstract syntax and semantics of data structures. The VDL graph as a basic VDL data structure and graph manipulation operators are introduced. The properties of the VDL graph are summarised in theorems. The most important result of these papers is that a data structure defined by VDL can be viewed as an abstract data type.

2. Object-oriented programming

The following features make a system object-oriented: *abstraction, abstract data type and knowledge-sharing*. *Abstraction* concentrates on the essential aspects of an entity and ignoring its nonessential properties. This capability is a fundamental part of object-oriented technology. Use of abstraction preserves the freedom to make decisions as long as possible by avoiding premature commitments to details. Object-oriented methodologies model a system by using the objects (entities) and interactions discovered in the system. In information engineering entities are only data; in the object-oriented approach, entities include both data and their associated functionality. These two items are bound together in a structure called class. The class specification captures the details of a class's design. It will show us the structure of a class and let us track the other information we will need to implement it.

An *abstract data type* abstracts away from the concrete representation given by a computation structure. An abstract data type is a class of data types which is closed under renaming of data domains, items and operations and hence independent of representation. Data type is

fundamental concept of specification of software system. A data type is a collection of data domains, designated basic data items, and operations on these domains such that all data items of data domains can be generated from the basic data items by use of the operations. So the data domains are assumed to be countable. From a mathematical point of view data types are algebras. Vice versa it does not make sense to claim that each algebra is also data type, because algebras may have noncountable base sets, like the real numbers. In addition it is often assumed that all data items of a data type are generated by a given set of the operations of this data type. These operations are called constructors. Another set of operations are called non constructors or selectors because these operations can be used to select parts of an object. The term abstract data type has many informal usages in programming and programming methodology. The syntactic structure of an abstract data type *D* is determined its signature, the semantics of *D* is based on the notion of computation structure, i.e. a many-sorted algebra *A* of the given signature [Var-83, Var-87, K-V-01, K-V-03]. So abstract data types can be formally defined in terms of the algebraic specifications. General rules can be given for constructing theorems about a given abstract data type and for proving the theorems. These theorems serve to convince us of that the given specification correctly takes the meaning of our concept [Var-83]. In paper [Var-87] abstract data types are specified by algebraic way and correct implementation of abstract data types is defined. It is shown that a correct implementation satisfies the semantic equations of the given abstract data type. On the other hand it is proved that if an implementation satisfies the semantic equations of an abstract data type then it is a correct implementation. For implementing a data type we may choose an appropriate representation and give concrete data type specification. Hence double specification is given for a data type. In this case the correctness of a concrete specification according to an abstract specification is formulated and sufficient conditions for the correctness of three different double specification is presented [Var-87-88, K-V-01, K-V-03].
In the development of programming systems for large applications the compositional approach to interactive concurrency is generally used. A possible way of ensuring the correctness of the independently developed components in this technology is the use of contract. Inheritance and polymorphism play important roles in modern programming languages such as Java , C++. In the technique of inheritance a class, a subclass inherits from another class, called superclass. The effect is that the subclass comprises the same attributes and same methods, for its methods, but there is nothing that guaranteed for the behavior of a subclass with respect to the behavior of its superclass. The subclass may contain additional attributes and methods, more over some methods could be overwrite too. In [D-K-V-06] the behavioral inheritance contract driven environment is formally defined on the base of substitution principle given by B. Liskov. Methods are given for proving the correctness of subtype with respect to the supertype in the cases of different kinds of data type specification methods.

3. List of Selected Publication

[C-V-88] N. H. Chien and L. Varga, Algebraic Specification of an Abstract High-level Debugger, Annales Univ. Sci. Budapest, Sect. Comp. 9 1988, pp. 33-50.

[D-K-V-06] Ákos Dávid, László Kozma, László Varga, On the Correctness of Data Type Classes Based on Contracts, 6th Joint Conf. on Math. And Comp. Sci., July 12-15, 2006, Pécs, Hungary

[K-V-01] Kozma László, Varga László, Adattípusok osztálya – Definiciók, elezés, példák - ,ELTE TTK Informatikai Tanszékcsoport, Budapest, 2001.

[K-V-03] Kozma László, Varga László, A szoftvertechnológia elméleti kérdései, ELTE Eötvös Kiadó, 2003.
[S-V-01] Sike Sándor, Varga László, Objektum elvű modellalkotás UML-ben, ELTE TTK Informatikai Tanszékcsoport, 2001.
[Var-76] Varga László, Adatszerkezetek absztrakt szintaxisa és szemantikája, Alkalmazott Matematikai Lapok 2, 1976, pp. 41-55.
[Var-79] L. Varga, An Abstract Graph Walk Algorithm, Annales Univ. Sci. Budapest, Sect. Comp. Tomus II. 1979, pp. 63-83.
[Var-80] L. Varga, Synthesis of abstract algorithms, Acta Cybernetica, Tom. 5, Fasc. 1, Szeged 1980, pp. 59-76.
[Var-81] Varga László, Programok analízise és szintézise, Akadémiai Kiadó, Budapest, 1981.
[Var-83] L. Varga, On the verification of abstract data types, Acta Cybernetica, Tom 6, Fasc. 1, Szeged, 1983, pp. 7-12.
[Var-87] L. Varga, Implementation of Abstract Data Types with Correctness Proof, Annales Univ. Sci. Budapest, Sect. Comp. 8, 1987, pp. 109-118.
[Var-87-88] L. Varga, Típusspecifikációk helyességének vizsgálata, Alkalmazott Matematikai Lapok 13, 1987-88, pp. 57-68.
[Var-89] Varga László, A programozási módszertan elmélete I.-II. rész, ELTE TTK, 3.változatlan kiadás, 1989.