Before we discuss the current level of achievement provided by the AMAST movement for the development of the software technology we would like to emphasize that by software we mean the collection of tools that helps in problem solving with a computer. There are three classes of such tools: (1) tools for the development of the problem model and its solving algorithm; (2) tools for mapping the problem model and its solving algorithm into program and data; and (3) tools for controlling the process of program execution on the given machine. The tools involved in the development of the problem model and its solving algorithm are mostly human, and help in thinking for a purpose, devising means to some desired end (after George Polya, Mathematical Discovery, Vol. 1, p. 118). They could hardly be considered as the objective of a technology. The rest of the tools involved in the process of problem solving with a computer perform mechanical actions and are amenable to technological development.

The algorithms for the translation of the problem and its solving algorithm into program and data are determined by the finite specification of the language used to express problems and algorithms. Thus, the difficulty of programming is in finding finite specifications for infinite mathematical languages, called programming languages, that are convenient for expressing problems and algorithms and that can be mapped automatically into efficient machine language programs. The collection of programming languages implemented on a machine defines the programming support environment of that machine. The process of program execution on a machine can be interrupted by many internal and external agents of the machine. The tools that are able to save the status of an interrupted program, resolve the cause of interrupt, and restart the interrupted program later must also be available. In addition, a machine solving problems is an expensive resource and needs to be exploited efficiently. The tools that ensure efficient exploitation of the machine by parallelizing the operations performed by its components processor, memory, environment, while maintaining the consistency of their interaction according to the computational contents they
perform, must also be provided. The collection of tools that allow the efficient execution of programs by a machine will be called the *execution support environment* of that machine. The two major components of a software technology are:

1. Application software, that designates the collection of software tools that allows the user to handle the process of problem modeling, algorithm development, and program writing, i.e., mapping algorithms into programs within the programming support environment of a given machine.

2. System software, that is a collection of software tools that allow convenient program development (i.e., compiling and testing programs on the given machine) and efficient program execution on that machine.

The nature of the difficulties raised by the development of these two components of the software technology and the methodology used to find ways around these difficulties are different. The problems raised by the development of the application software are exactly the problems raised by problem modeling and algorithm development. Hence, the problems raised by software technology are actually the problems encountered during the development of a system software that allows the industrial development of the application software.

The difficulties in solving problems with a machine are the difficulties of using the notation provided by the programming support environment as well as those of creating control programs that when interpreted make efficient use of the hardware. The tools that belong to the system software of a given machine are specific to that machine and cannot be easily changed. Usually, the life of such a tool is as long as the life of the hardware. Therefore, the system software is very binding for the user. The consequence is that once a hardware system is functional the new research on the system software is almost frozen, while the development of applicative programming methodology is growing. The best example illustrating this situation is provided by the programming methodology in use today, that was developed based on the huge programming experience using primitive languages such as Fortran and Cobol. Since the semantics of these languages depends strongly on the machines on which they are implemented, their programs cannot be ported easily to new languages and machines. This state of the art is probably best reflected by the attitude taken by the computer community between 1970 and 1990, when almost the entire research has been dedicated to the development of a powerful application software (unfortunately bound to given machines and languages) while the research on system software was almost forgotten. At the same time, there was a dramatic evolution of hardware and of computation paradigms that did not necessarily gain user acceptance due to the programming methodology in use. However, the development of new parallel machines and the necessity to manage very large computation tasks show that the only true way out of the difficulties raised by problem solving with a computer is through the development of an appropriate technology for system software. The major drawbacks of the current methodology for problem solving with a computer are:

1. Program development for the new machines is difficult due to the lack of support for incremental program development. A conventional compiler assigns
computation meaning only to the program as a monolithic construct. That is, the compiler cannot map the components of a source language program into executable machine language programs. This would allow the user to test the behavior of program components before integrating them into larger components.

(2) Operating systems do not allow the programmers to interact with the tools they use for program development, that is, one cannot interact with the code generator, optimizer, parallelizer, linker/loader, scheduler, etc., while a program is being developed.

(3) Programs survive the machines and the system software tools used to develop them. However, the conventional software do not provide for moving old programs on new languages and machines.

(4) Historically, processes performing in parallel were visible only to the operating system. Therefore, no appropriate language support for parallel programming has evolved and parallel computations cannot be mapped naturally by the programmer into programs to be executed by a parallel hardware.

(5) The computer user continues to be excluded from the process of system software development. But a tool is good only if it fits its user. This means that the computer manufacturers should furnish their computers with tools that allow system software tool development rather than sealing a system software that may not be appropriate for some user in a certain context.

These drawbacks show the price we pay for growing a computer industry whose two inseparable components, hardware and software, are unbalanced. The hardware development is based on well-established engineering principles while the software follows an ad hoc methodology dependent on the structure of the hardware. This is caused by the way the concept of technology evolved in the context of application of knowledge for touchable things. Since the methodology for problem solving originates in mathematics and handles abstractions rather than touchable things, the software evolved as the untouchable part of the computer. Therefore, the software technology is a phrase that is difficult to really grasp without understanding the concept of abstraction manipulation.

The AMAST movement is promoted by the international conferences on Algebraic Methodology and Software Technology started in 1989 in Iowa City, Iowa. Its goal is to consolidate the trend of looking at algebraic methodology as a foundation for software technology. Unlike other conferences on mathematical foundations of computer science, in which the mathematics is usually enriched with new theories originated in computer science, the trend initiated by AMAST shows computer science developments that originate in mathematics. The algebraic property that leads to the software technology required by the new applications and machines is the incremental nature of the algebraic manipulation. This means that the structure of a computational object is defined in terms of the structure of the components of the object. Using this property, the actions performed by various software tools can be formally defined as structure-preserving mappings called homomorphisms and various generalizations. The most influential mechanisms of such incremental algebraic computations
visible in the recent machine developments are concurrency, pattern matching, rewriting rules, and binding. The structure of the concurrent computational objects is defined in terms of the structure of their components. The structure of the patterns in the fast pattern-matching algorithms is defined in terms of the structure provided by the pattern components. The structure of the algorithms based on rewriting rules is explained in terms of the structure defined by composing operations specified by the signature provided by the rewriting rules. The structure of binding objects to their values in a given environment is explained in terms of structures of binding component objects to component values. The general characteristic of these algebraic mechanisms of computation is that each provides the model for concrete software tools existing in the current software technology. This research trend in software technology has been well covered at the AMAST'91.

Of the 120 papers submitted at the second AMAST conference held on May 22–25, 1991, in Iowa City, Iowa, 30 were selected for presentation at the conference. Short versions of these papers were published in the local proceedings while extended versions make the object of Proceedings of the 2nd AMAST Conference published by Springer, London, in Workshops in Computing series. Of the seven papers chosen to be published in an issue of *Theoretical Computer Science* dedicated to this conference, the reviewers selected the four most representative.

In this issue, R. Janicki and M. Koutny, in the article *Structure of concurrency*, present algebraic models of concurrency where the properties of the concurrent behaviors are modeled in terms of the structure provided by partial orders of the behavioral elements. Y.V. Srinivas, in the article *A sheaf-theoretic approach to pattern matching and related problems*, uses Grothendieck topologies to provide a unifying methodology for the development of pattern-matching algorithms. C. Talcott, in the article *A theory of binding structures and applications to rewriting*, discusses the binding structures using abstract algebras and defines the general notion of parameterized homomorphism. M. Thomas and P. Watson, in the article *Solving divergence in Knuth–Bendix completion by enriching signatures*, use enriching signatures of operations to solve divergence in Knuth–Bendix completion. The characteristics of all these papers is that they show both how to use the algebraic methodology for specific software tool development and provide the applications of their results. We can see that the papers presented at AMAST'91 indeed show progress towards the initial goal of the AMAST movement: *using algebraic methodology for the development of software technology*.

I acknowledge the efforts of all the authors and referees and thank M. Nivat, Editor-in-Chief of *Theoretical Computer Science*, for helping us bring AMAST’s idea to light.

_Teodor Rus_  
_Guest Editor_