Special Issue
Automatic Programming
—Foreword of the Guest Editors

1. Introductory Remarks

The design and implementation of correct software meeting given requirements continues to be a most relevant, practical, and scientifically challenging problem. There are many lines of research directed towards solving this problem. Automatic Programming is one among those. It is the study of techniques for generating executable code from information which may be fragmentary and may only indirectly specify the target behavior.

The field is based on the idea that ultimately we need to engage the machine itself in the process of programming machines, since only machines offer the important property needed for this task which is the ability to work without making mistakes. The "Automatic" in the name does not necessarily refer to a full automation of programming, but rather to a considerably higher degree of automation than in other lines of software production pursued by the literature. The earliest developments in the field were the first assemblers and compilers for digital computers. The more recent work has concentrated on the synthesis of algorithms from specifications, high level commands, or examples. The current special issue presents a variety of new techniques utilizing transformations, rewriting techniques, logical deduction, and other mathematical methods.

We would like to express our thanks to Bruno Buchberger for inviting us to edit this issue. We also wish to thank the many authors who contributed their papers and the reviewers who read them. Only through the extensive efforts of a large number of people was it possible to bring this issue into being.

2. Discussion of the Individual Contributions

The first set of papers follow an important line of research - the creation of executable code through the transformation or rewriting of the original specifications.

The paper by Dershowitz and Reddy describes an equational approach which transforms a specification into a program using an ordered rewriting mechanism. The method is formal in that it generates programs that are provably compatible with the given specifications. It also includes some heuristics which aid in efficient convergence to a solution.

The work by Boulanger and Bruynooghe presents a uniform technique for covering a large class of fold/unfold transformations which can be generated out of a preceding interpretation phase in which the source program is analyzed. The paper by van Deursen, Klint and Tip addresses the topic of generating interactive programming environments from language specifications. Their technique uses term rewriting for the processing of the specifications. In particular, they focus on the problem of tracking the origin of some term and then proposing a solution to it.

The paper by Clauss and Mongenot addresses the problem of synthesizing efficient processor arrays from a problem specification in terms of affine recurrence equations.
They develop a two step approach which includes novel partitioning techniques reducing the number of processors needed and resulting in more efficient parallel solutions. This work contributes to the aspect of synthesizing parallel machinery where automation seems even more mandatory than in the sequential case.

Smith has developed a generalized view of his previously well refined approach to transformational automatic programming as implemented in the KIDS system. The new view formats the synthesis process as a search for a mapping, or morphism, from the specification to known theories of algorithms. The discovery of the morphism provides the instantiations needed to enable the construction of the program. The advantage of the method is that the variety of synthesis and refinement tactics of previously developed systems now can be seen in a uniform format.

The second set of papers represent the programming-by-proving approach to synthesis. Logical specifications are proved and then a functional proof is extracted from the derived proof which is compiled into an executable program. The paper by Paulin-Mohring and Werner is an example of this approach. Specifications are proved in the proof development environment Coq. Then a functional proof is extracted and compiled into an executable ML program.

In the same vein, the paper by Takayama proposes a calculus which may be seen as a restricted version of Martin-Löf’s intuitionistic type theory. With the restriction, he aims at a more direct extraction of executable functional programs and demonstrates that the formalism is still expressive enough for dealing with parameterized specifications. In fact, a higher degree of automation can be achieved this way.

The work by Omodeo, Parlamento and Policriti addresses the problem to what extent set theory could be interpreted constructively, so that algorithms could be extracted from high level specifications involving set constructs.

The next paper belongs in a class by itself. It is the Chadha and Plaisted article which addresses the classic problem of creating loop invariants needed for proving the partial correctness of a program. The method uses an unskolemization technique and is complete in the sense that if a loop invariant exists relative to a given set of first order axioms, then the algorithm will find it.

Dershowitz and Lee have extended ideas of E. Shapiro in logic program debugging. Their methodology requires the user to give both the specifications for the target program and the possibly erroneous program. The specification is used to generate test cases and to locate faulty statements. It is also used with a series of heuristics to aid in program repair.

The final two papers examine the possibility of using examples in the synthesis process. Flener and Deville have found a way to synthesize logic programs from a combination of incomplete specifications and examples. The goal is to relax the requirement that the user provide perfection in the description by allowing the use of examples to disambiguate the incompleteness. They have instantiated their theory in a system to generate divide-and-conquer programs from a generalized schema.

Fahmy and Biermann have developed a synthesis-from-examples technique for so called “real time acceptors”. The methodology is based upon the observation that the behavior of a real time program, as represented by a behavior graph, is effectively the product of the control structure graph and the data structure graph as defined in the paper. Synthesis is thus done by constructing the behavior graph and factoring it into its two components, the control and data structure graphs.

In summary, this set of twelve papers contributes to a growing and important field of
computer science both by continuing some fruitful existing topics and by opening some new ones. The results of this work will be to both enhance our understanding of the theory of programming and to lead to practical systems that help people do their work.

Wolfgang Bibel
Alan W. Biermann