ELSEVIER

The Journal of Logic Programming 39 (1999) 1–2

THE JOURNAL OF LOGIC PROGRAMMING

Guest editors' introduction

Special issue: synthesis, transformation and analysis of logic programs

Annalisa Bossi [a,1], Yves Deville [b,2]

[a] *Università Ca' Foscari di Venezia, Via Torino 155, 30173 Mestre-Venezia, Italy*
[b] *Université catholique de Louvain, Departement d'Ingénierie Informatique, Place Ste Barbe, 2 B-1348 Louvain-la-Neuve, Belgium*

This Special Issue of JLP is devoted to three basic areas of logic program development: *synthesis*, *transformation* and *analysis* of logic programs. *Program Synthesis*, in a broad way, refers to the elaboration of a program in some systematic manner, starting from a (nonexecutable) specification. Program synthesis mainly focuses on automated or semi-automated synthesis. *Program Transformation* deals with the successive transformations of a given program into equivalent but "better" programs. Usually, the adjective "better" refers to "more efficient" with respect to some operational semantics. The borderline between synthesis and transformation is very thin and rather subjective. A possible difference could be that synthesis starts from specifications written in some richer logical languages. *Program Analysis* aims at proving or providing properties of given programs, such as termination, mode, type, and sharing. Program analysis covers the elaboration of general methods (e.g. abstract interpretation), the analysis of specific properties or classes of programs, and the construction of automated tools.

Since the beginning of logic programming, there has been an ongoing research on these areas. Nowadays, many key concepts have been clarified, theoretical bases have been established, methods have been developed and automated tools have been produced. But we are still in the never-ending process which continuously improves programmers' capability in developing correct, reliable and efficient programs.

This Special Issue on *Synthesis, Transformation and Analysis of Logic Programs* is organized in two parts. The first part, published in this volume, includes papers on program analysis. The second part will propose contributions on synthesis and transformation of logic programs.

[1] E-mail: bossi@dsi.unive.it
[2] E-mail: yde@info.ucl.ac.be.

PII: S0743-1066(98)10031-6

The contributions on *program analysis* published in this volume can be broadly classified under three different, but complementary categories: general methods, specific properties or classes of programs, and automated tools for verification/debugging.

The two first papers present general methods for verification/analysis of logic programs. The first one, *Verification of Logic Programs*, by Dino Pedreschi and Salvatore Ruggieri, aims at providing a unifying framework, in the style of Hoare's logic, for the verification of logic programs. It introduces a proof theory which combines previous proposals for the verification of partial correctness and termination of logic programs, by gaining in expressiveness. A different approach to program analysis is presented in the second paper: *Abstract Diagnosis*, by Marco Comini, Giorgio Levi, Maria Chiara Meo and Giovanna Vitiello. Here, the goal is that of identifying program bugs with respect to the intended behavior, i.e. intended program properties. The technique leads to verification methods of abstract properties described by finite domains.

The following two papers consider specific properties or classes of programs. The paper entitled *Termination of Logic Programs with Delay Declarations*, by Elena Marchiori and Frank Teusink, considers a single property, termination, but in a new context: logic programs with delay declarations. It is well-known that well-moded programs, i.e. which respect some correctness conditions on the input/output connections, allows one to derive efficient methods for verifying diverse properties of logic programs. The paper titled *Layered Modes*, by Sandro Etalle and Maurizio Gabbrielli, extends existing mode systems enlarging the class of programs which can be verified by using modes.

Finally, there are two papers which describe automated tools for verification/debugging. Baudouin Le Charlier, Christophe Leclère, Sabina Rossi and Agostino Cortesi are the authors of the fifth contribution, titled *Automated Verification of Prolog Programs*. They present a generic analyser for Prolog which can be used to check the correctness of Prolog programs with respect to various properties such as modes, types, sharing and termination. The last paper in this issue describes *Opium: An Extendable Trace Analyser for Prolog*, by Mirelle Ducassé. Opium is based on a trace query model. The essential features of this model are ability of modeling execution data, synchronous trace querying and meta-debugging capabilities. The Opium trace analyser offers the capabilities to build abstract tracers and automated debugger facilities.

Many thanks to Maurice Bruynooghe, Editor-in-Chief of JLP, for inviting us to edit this issue, and for his helpful support. We are grateful to the authors for providing high-quality contributions. Finally, we thank all the reviewers for their helpful criticisms, suggestions and advice.

Annalisa Bossi
Venice, Italy

Yves Deville
Louvain-la-Neuve, Belgium