ELSEVIER The Journal of Logic Programming 41 (1999) 139–140

THE JOURNAL OF LOGIC PROGRAMMING

www.elsevier.com/locate/jlpr

Guest editors' introduction

Special issue: synthesis, transformation and analysis of logic programs 2

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

[a] *Università Ca' Foscari di Venezia, Via Torino 155, 30173 Mestre-Venezia, Italy*

[b] *Université catholique de Louvain, Place Ste Barbe, 2 B-1348 Louvain-la-Neuve, Belgium*

This volume contains the second part of the Special Issue on *Synthesis, Transformation and Analysis of Logic Programs*. The first part on *program analysis* appeared in Vol 39(1–3). The second part includes contributions on *program synthesis* and *program transformation*.

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.

There are four contributions on *program synthesis and program transformation* published in this volume.

The first paper, *Inductive synthesis of recursive logic programs*: *achievements and prospects*, by Pierre Flener and Serap Yılmaz overviews the achievements of inductive synthesis of logic programs from incomplete specifications. This paper focusses on the synthesis of *recursive* programs. It also debates the practical applicability of these techniques in two application areas: knowledge discovery and software engineering.

The borderline between synthesis and transformation, which is very thin, is even thinner in a logic programming context. In fact, in this environment the same transformations we can use to get a more efficient program from an initial one can be applied to logic specifications which are not executable, as they are. This is shown in the

[*] Corresponding author. Tel.: +39-041-2908421; fax: +39-041-2908419; e-mail: bossi@dsi.unive.it

[1] E-mail: yde@info.ucl.ac.be

second paper of this volume *Synthesis and transformation of logic programs* using unfold/fold proofs by Alberto Pettorossi and Maurizio Proietti.

We refer to program specialization when the transformation aims at gaining efficiency by exploiting the fact that the program will be employed in a certain context, that is, for a restricted set of input values. The optimization is generally achieved by partially evaluating the program with respect to the given input set. In the case of logic programs, this takes the form of *partial deduction*, which can be seen as a subset of the class of unfold/fold transformations where *unfolding* is the basic transformation rule. In *Conjunctive partial deduction*: *foundations*, *control*, *algorithms and experiments*, Danny De Schreye, Robert Glück, Jesper Jørgensen, Michael Leuschel, Bern Martens and Morten Heine Sørensen present a framework for program specialization which extends conventional partial deduction techniques by incorporating more rules of the unfold/fold approach.

A different extension to partial evaluation is proposed by German Puebla and Manuel Hermenegildo, in *Abstract multiple specialization and its application to program parallelization*. Here, the authors consider the case when the set of possible inputs is unknown or infinite and show how a form of specialization can still be performed in such cases by means of abstract interpretation.

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.