TLP **2** (4 & 5): 423–424, July & September 2002. © 2002 Cambridge University Press
DOI: 10.1017/S1471068402001424 Printed in the United Kingdom

Special issue on 'Program development'

MAURICE BRUYNOOGHE
Katholieke Universiteit Leuven, Department of Computer Science, Celestijnenlaan 200A, B 3001 Heverlee Belgium
(*e-mail:* Maurice.Bruynooghe@cs.kuleuven.ac.be)

KUNG-KIU LAU
University of Manchester, Department of Computer Science, Oxford Road, Manchester M13 9PL, UK
(*e-mail:* kung-kiu@cs.man.ac.uk)

This special issue marks the tenth anniversary of the LOPSTR workshop. LOPSTR started in 1991 as a workshop on Logic Program Synthesis and Transformation, but later it broadened its scope to logic-based Program Development in general.

The motivating force behind LOPSTR has been a belief that declarative paradigms such as logic programming are better suited to program development tasks than traditional non-declarative ones such as the imperative paradigm. Specification, synthesis, transformation or specialisation, analysis, verification and debugging can all be given logical foundations, thus providing a unifying framework for the whole development process.

In the past ten years or so, such a theoretical framework has indeed begun to emerge. Even tools have been implemented for analysis, verification and specialisation. However, it is fair to say that so far the focus has largely been on programming-in-the-small. So the future challenge is to apply or extend these techniques to programming-in-the-large, in order to tackle software engineering in the real world.

Returning to this special issue, our aim is to present a collection of papers that reflect significant research efforts over the past ten years. Following an open call, 13 papers were submitted, of which five have met the quality criteria and deadlines for being included in this special issue. It is pleasing, though more by accident than design, that these papers cover nearly the whole development process: synthesis, transformation, analysis and debugging.

The first paper, 'A refinement calculus for logic programs' by Ian Hayes, Robert Colvin, David Hemer, Paul Strooper and Ray Nickson, describes an adaptation of refinement methodologies for imperative program synthesis to the domain of logic program synthesis. They describe a wide-spectrum logic programming language which contains both executable and specification language constructs. They provide a semantics for the language, and then present correctness-preserving refinement rules for manipulating terms of the language into purely executable programs. They also provide an application of their calculus and describe a prototype implementation.

https://doi.org/10.1017/S1471068402001424 Published online by Cambridge University Press

The second paper, 'Logic program specialisation through partial deduction: control issues' by Michael Leuschel and Maurice Bruynooghe, offers a broad overview of techniques developed by many researchers over the past 10 years to control partial deduction of logic programs, in particular to ensure their termination. More specifically, it concerns Prolog with finite failure, transformed by the popular Lloyd-Shepherdson 'partial deduction', a version of specialisation that works by an elaboration of normal Prolog execution.

The third paper, 'A backward analysis for constraint logic programs' by Andy King and Lunjin Lu uses Heyting completion to systematically develop a schema for backward analysis of (constraint) logic programs in the framework of abstract interpretation. To illustrate the schema, the authors use Pos to derive an analysis that will produce sufficient groundness conditions (for queries) under which no insufficiently-instantiated errors will occur at runtime.

The fourth paper, 'Using parametric set constraints for locating errors in CLP programs' by Włodek Drabent, Jan Małuszyński and Paweł Pietrzak, introduces a framework of parametric descriptive directional types for constraint logic programs. A novel technique based on parametric set constraints is the basis of a tool that checks partial correctness of a CLP program with respect to an approximate specification and locates possible program errors if such checks fail. A prototype tool is discussed with simple examples.

The last paper, 'Generic program monitoring by trace analysis' by Erwan Jahier and Mireille Ducassé, describes a generic and efficient programming abstraction for monitors that collect information from execution event traces. The abstraction proposed is a fold on lists of execution events. The genericity of the approach is demonstrated by applying it to a wide range of monitors, ranging from straightforward examples that familiarize the reader, to involved examples such as test-case coverage that are convincing due to their conciseness.

This special issue could not have been created without the help of the authors who contributed high quality papers, and of reviewers who prepared insightful and timely reviews. We thank them all. Finally, a special word of thanks goes to Giorgio Levi, one of the Area-Editors of TPLP, for handling a paper coauthored by one of us.

https://doi.org/10.1017/S1471068402001424 Published online by Cambridge University Press