Fundamenta Informaticae 69 (2006) v-vii
IOS Press

Program Transformation: Theoretical Foundations and Basic Techniques. Part 2

Preface

This issue is the second Special Issue dedicated to the Theoretical Foundations and Basic Techniques for Program Transformation. The first Special Issue has been published in Volume 66, Number 4, April-May 2005.

The Call for Papers included the following topics:

(i) transformation approaches and formalisms: rule-based, calculation-based, and schema-based transformation;

(ii) program transformation in different programming languages: imperative, functional, logic, constraint-based, object-oriented, concurrent, and distributed languages;

(iii) formal properties of transformations: correctness, completeness, and complexity of transformations;

(iv) transformation strategies and techniques for program optimization: composition, accumulation, tupling, specialization, generalization, and parallelization;

(v) interaction of program transformation with related methodologies for assisting software development, such as: program analysis, synthesis, refinement, verification, component-based software construction, software reuse, theorem proving, and meta-programming;

(vi) languages and systems for specifying and applying program transformations; and

(vii) case studies, that is, derivation of non-trivial algorithms from specifications and automated generation of software systems.

Among all papers we have received, twelve papers were selected for publication. Five papers were published in the first Special Issue and the remaining seven papers are published in this second Special Issue.

The paper "Redundant Call Elimination via Tupling" by Wei-Ngan Chin, Siau-Cheng Khoo, and Neil Jones, describes several extensions of the tupling transformation technique, which has been proposed as a means for the elimination of redundant calls in functional programs. The authors consider a first-order functional language and provide a polynomial time program analysis which ensures the termination of the tupling transformation. In the class of programs where the analysis succeeds, the elimination of redundant calls can be performed at compile-time in a fully automatic way.

In the paper "Higher Order Deforestation", Geoff W. Hamilton proposes an extension of deforestation, a well known transformation algorithm which can eliminate intermediate data structures from functional programs. The proposed algorithm handles higher order programs and is shown to terminate for the class of programs where all function definitions are given in a particular syntactical form, called higher order treeless form. One advantage of the extended deforestation algorithm is that, since it uses a syntactically recognisable form, it is often quite simple for the programmer to identify the improvements which will be made.

The paper by Patricia Johann and Janis Voigtländer, entitled "The Impact of seq on Free Theorems-Based Program Transformations", investigates the validity of parametricity theorems and, among them, the so-called free theorems in the case of nonstrict languages which have a polymorphic strict evaluation primitive such as the seq primitive of Haskell. These parametricity theorems, including, for instance, the short cut fusion theorem, are the basis for various program transformation rules. These theorems are shown to hold for a subset of Haskell corresponding to a calculus à la Girard-Reynolds with fixpoints and algebraic datatypes even when seq is present, provided that the relations which occur in the derivations of the theorems are left-closed, total, and admissible.

The paper "Implementing Typeful Program Transformations" by Chiyan Chen, Rui Shi, and Hongwei Xi addresses the problem of a good representation of typed programs so that their transformations can be performed in a safer way. The authors present some programming techniques which allow us to derive program representations in which the type of the object programs as well as the types of the free variables occurring in them, can be reflected in the type of their representations.

The paper by Martin Bravenboer, Arthur van Dam, Karina Olmos, and Eelco Visser, entitled "Program Transformation with Scoped Dynamic Rewrite Rules", proposes the use of dynamic transformation rules and user-definable strategies for improving the power of the program transformation system Stratego. In this system, as in many other transformation systems, a set of rules is used for deriving new, more efficient programs from old, possibly less efficient ones. Dynamic rules can specify transformations which are context-sensitive in the sense that they depend on some variables whose values are determined by the context in which the rules are applied. These rules and the user-definable strategies are illustrated by several examples of program improvement, transformation, and specialization.

In logic programming the logic component of an algorithm is conceptually separated from the control component. The objective of control generation is to derive in an automatic way a computation rule for a logic program which is efficient and yet preserves program correctness. The paper "Control Generation by Program Transformation" by Andy King and Jonathan C. Martin shows how the control generation problem can be tackled by program transformation. The transformation relies on information about the depths of derivations to derive delay declarations which orchestrate the control. In contrast to previous work, the technique proposed in this paper does not require an atom to be completely resolved before another is selected for reduction. This enhancement permits the transformation to introduce control which is flexible and relatively efficient.

Many transformation techniques for logic programs are based on the unfolding and folding transformation rules. Folding can be regarded as a partial inverse of unfolding, in the sense that if a given program is transformed into a new one by unfolding, then the inverse transformation can be realized by folding in a restricted number of cases only. In the paper "A multiple-clause folding rule", David A. Rosenblueth extends the folding rule so as to be able to fold the clauses resulting from any unfolding of a positive literal, thereby making folding a full inverse of unfolding. The extended folding rule is shown to be correct, in the sense that it preserves Dung and Kanchanasut's semantic kernel. The author indicates

some possible applications of the extended folding rule to decompilation, reengineering, and inductive logic programming.

Acknowledgements

We would like to thank the following people who served as referees for these two special issues on Program Transformation: Foto Afrati, Elvira Albert, Arthur Baars, Roland Backhouse, Richard Bird, Annalisa Bossi, James M. Boyle, Wei-Ngan Chin, Agostino Cortesi, Olivier Danvy, Danny De Schreye, Sandro Etalle, Santiago Escobar, Laurent Fribourg, Thom Frühwirth, John Gallagher, Irene Guessarian, Michael Hanus, Ralf Hinze, Zhenjiang Hu, Hideya Iwasaki, Claude Kirchner, Annie Liu, Maria-Chiara Meo, Marc Meister, Ian Miguel, Yasuhiko Minamide, Bernhard Möller, Torben Mogensen, Pierre-Etienne Moreau, Shin-Cheng Mu, Damian Niwinski, C. R. Ramakrishnan, Morten Rhiger, Markus Roggenbach, Abhik Roychoudhury, Colin Runciman, David Sands, David Schmidt, Dietmar Seipel, Ganesh Sittampalam, Doaitse Swierstra, Leonardo Tininini, Tarmo Uustalu, Zhe Yang, Dave Wile, Burkhart Wolff.

Without their kind and professional cooperation the realization of the two special issues would have been impossible.

Alberto Pettorossi
Department of Informatics, Systems, and Production,
University of Rome “Tor Vergata”
Via del Politecnico 1, I-00133 Rome, Italy
pettorossi@info.uniroma2.it

and

Maurizio Proietti
IASI-CNR,
Viale Manzoni 30, I-00185 Rome, Italy
proietti@iasi.rm.cnr.it

September 2005