Fundamenta Informaticae 66 (2005) i-iii
IOS Press

Program Transformation: Theoretical Foundations and Basic Techniques. Part 1

Preface

Over the last three decades the program transformation methodology has been proved to be a powerful technique for deriving programs from specifications, verifying program properties, specializing programs w.r.t. their context of use, and deriving more efficient program versions from less efficient ones. The transformation methodology has been first proposed in the area of functional programming by R. M. Burstall and J. Darlington in their seminal paper "A Transformation System for Developing Recursive Programs" [1]. Then H. Tamaki and T. Sato applied the transformation methodology also to the area of logic programming [4] and since then, many other papers and research efforts have been devoted to the development of a variety of approaches to program transformation in functional and logic programming and in other programming paradigms as well. For a survey which illustrates the basic ideas of program transformation and covers the early work in the area, the reader may refer to [3].

More recently, people have been considering the application of transformation techniques to the case of "programming in the large". Developments in this direction may be found in the companion Special Issue of Science of Computer Programming [2] edited by Ralf Lämmel. That Special Issue is more oriented towards the applications of the program transformation methodology in software engineering.

The Call for Papers for our Special Issue 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, five were selected for this Special Issue. Other papers will be published in forthcoming special issues.

The paper "Point-free Program Transformation" by Alcino Cunha and Jorge Sousa Pinto focuses on the calculation-based approach to program transformation, by which programs are developed in an algebraic style via equational reasoning. The authors revisit many well-known transformation strategies, like tupling, fusion, and accumulation, by using a pure point-free calculus à la Backus. Several algebraic laws that capture typical transformation patterns are proved. The article also presents a number of fully worked out, old and new, examples.

Proving program properties is an important ingredient of any program transformation technique and, thus, proof methods are very relevant to program transformation. "Proof Methods for Corecursive Programs" by Jeremy Gibbons and Graham Hutton, is about some methods for proving properties of corecursive programs, that is, functions whose range is a recursively defined type which is the greatest solution of some equation. Typical instances of corecursive programs include functions that produce infinite lists or infinite trees. This article is a tutorial introduction to the four main proof methods for corecursive programs: fixpoint induction, approximation lemma, coinduction, and fusion. This article is mainly addressed to functional programmers, but it is also accessible to a larger audience.

Program inversion is a classical program transformation task that was first considered by E.W.Dijkstra. Given a program P that computes an injective function f, program inversion consists in deriving a program, denoted $inv(P)$, that computes the inverse function f^{-1}. In "A Method for Automatic Program Inversion Based on LR(0) Parsing", Robert Glück and Masahiko Kawabe present an automatic method for program inversion. Their method first derives a nondeterministic program which can be viewed as a context-free grammar. Then, by applying LR-based parsing methods, the nondeterministic program is transformed into a deterministic one.

The paper "There and Back Again" by Olivier Danvy and Mayer Goldberg, is about a programming pattern, called TABA, where a recursive function traverses a data structure at call time and another data structure at return time. Typical instances of this pattern are programs for computing symbolic convolutions and multiplying polynomials. Through various examples, the authors illustrate how the TABA programming pattern can be combined with other programming patterns like, for instance, dynamic programming. The paper also illustrates the synergism between TABA and various program transformation techniques, like the continuation passing style and the defunctionalization transformations.

In the paper "Infinite Unfolding and Transformations of Nondeterministic Programs", Björn Lisper presents some new results on the correctness of transformations of nondeterministic programs. These results are based on the idea that two programs are equivalent if they can be symbolically unfolded to the same, possibly infinite, term. The author provides a sufficient condition that ensures the correctness of unfold/fold transformations of nondeterministic programs.

Alberto Pettorossi **and** **Maurizio Proietti**

Department of Informatics
Systems, and Production
University of Rome "Tor Vergata"
Via del Politecnico 1, I-00133 Rome
Italy

IASI-CNR
Viale Manzoni 30, I-00185 Rome
Italy

April 2005

References

[1] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, January 1977.

[2] Ralf Lämmel, editor. *Special Issue of Science of Computer Programming on Program Transformation*, Vol. 52 (1–3). Elsevier, August 2004.

[3] A. Pettorossi and M. Proietti. Rules and strategies for transforming functional and logic programs. *ACM Computing Surveys*, 28(2):360–414, 1996.

[4] H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In S.-Å. Tärnlund, editor, *Proceedings of the Second International Conference on Logic Programming*, pages 127–138, Uppsala, Sweden, 1984. Uppsala University.