Editorial from the issue entitled "Special Issue: PEPM 2010"

Gallagher, John Patrick; Voigtländer, Janis

Published in:
Higher-Order and Symbolic Computation

DOI:
10.1007/s10990-011-9081-0

Publication date:
2011

Document Version
Early version, also known as pre-print

Citation for published version (APA):
Gallagher, J. P., & Voigtländer, J. (2011). Editorial from the issue entitled "Special Issue: PEPM 2010". *Higher-Order and Symbolic Computation*, *23*(3), 273-274. https://doi.org/10.1007/s10990-011-9081-0

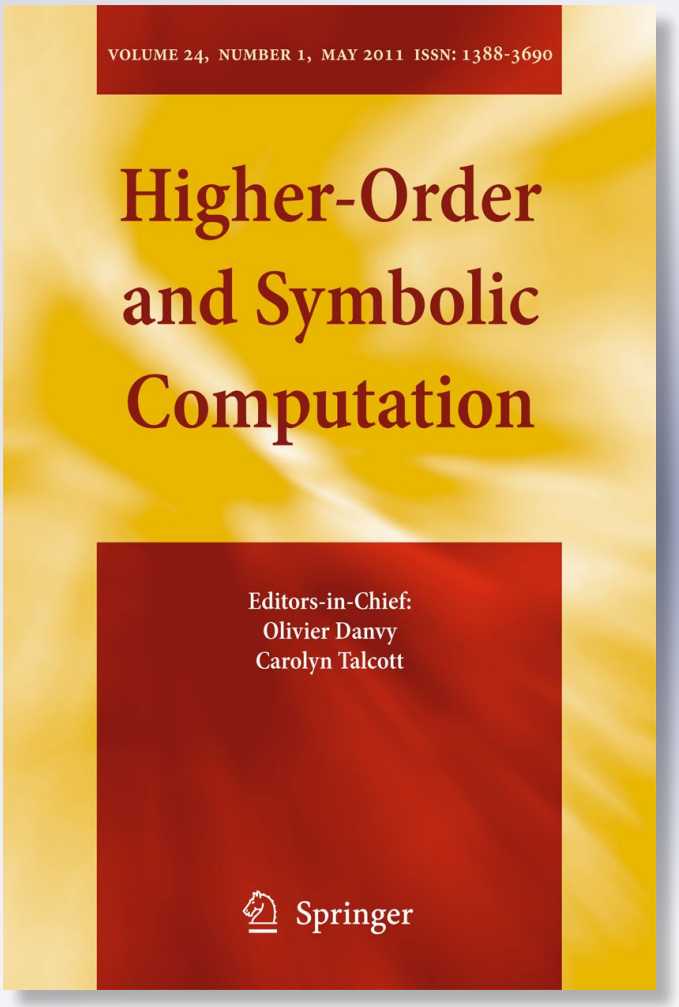
Editorial

John P. Gallagher & Janis Voigtländer

Higher-Order and Symbolic Computation

ISSN 1388-3690
Volume 23
Number 3

Higher-Order Symb Comput (2011)
23:273-274
DOI 10.1007/s10990-011-9081-0



Springer

Higher-Order Symb Comput (2010) 23:273–274
DOI 10.1007/s10990-011-9081-0

EDITORIAL

Editorial

John P. Gallagher · Janis Voigtländer

Published online: 24 November 2011

This special issue of HOSC contains extended versions of selected articles from PEPM'10, the ACM SIGPLAN 2010 Workshop on Partial Evaluation and Program Manipulation, which took place on 18th and 19th January 2010 in Madrid, Spain [1]. All four were rigorously reviewed subject to journal standards by at least three reviewers each.

PEPM's focus is on techniques and supporting theory for the analysis and manipulation of programs, bringing together researchers working in various areas of this general theme. The articles in this special issue reflect this diversity. It collects articles on static analyses, program transformation and generation, type-based programming, and runtime optimization, for imperative, functional, and object-oriented languages.

The article "Context-sensitive analysis without calling-context", by Arun Lakhotia, Davidson R. Boccardo, Anshuman Singh, and Aleardo Manacero Júnior, tackles a well known problem—interprocedural flow analysis—in the relatively new setting of obfuscated code. Classical interprocedural analysis requires some notion of calling context in order to be accurate, since the same procedure can be called from different places and in different states. The obfuscation of call and return instructions and of procedure boundaries renders established techniques for interprocedural analysis (in particular Sharir and Pnueli's method) inapplicable or ineffective. The approach proposed in the article is to introduce a new, more low-level notion of context, called stack-context, that is based purely on the state of the stack. Using the abstract interpretation framework, the authors develop abstractions of

J.P. Gallagher
Roskilde University, Universitetsvej 1, 4000 Roskilde, Denmark
e-mail: jpg@ruc.dk

J.P. Gallagher
IMDEA Software Institute, Campus Montegancedo, 28660 Boadilla del Monte, Madrid, Spain

J. Voigtländer (✉)
University of Bonn, Römerstraße 164, 53117 Bonn, Germany
e-mail: jv@informatik.uni-bonn.de

semantics incorporating the stack-context, yielding analyses that are on a par with classical techniques for unobfuscated code while giving more precise results for obfuscated code.

In "Making 'stricterness' more relevant", Stefan Holdermans and Jurriaan Hage extend strictness analysis for lazy functional languages like Haskell and Clean to take into account properly and effectively programmer annotations that deliberately increase the strictness of selected functions. Such annotations are already crucial to giving the programmer a means of dynamically avoiding certain kinds of performance problems in a lazy setting, but promise even more benefits if the compiler can statically pick up and widely propagate (and then use to drive code optimizations) information about the effected change in evaluation order. Adopting relevance typing, and adapting it appropriately by introducing a concept of applicativeness of expressions, the authors provide the foundation for performing such static propagation both safely and vigorously.

The topics addressed by "Generic multiset programming with discrimination-based joins and symbolic Cartesian products", by Fritz Henglein and Ken Friis Larsen, are dynamic symbolic computation on the one hand, and specific algorithmic techniques for executing common relational algebra operations on the other. The results are embodied in a generic Haskell library supporting efficient SQL-style queries on rich data types and with user-defined functions, predicates, and equivalence and ordering relations. The combination of discrimination-based joins (an algorithmic idea previously introduced by Henglein) and manipulation of symbolic representations of multiset union and Cartesian products at runtime leads to good performance. Optimizations are achieved on the fly that require much more dedicated effort in standard database query planning approaches. The approach makes elegant use of HOT (higher-order and typed) functional programming techniques like Generalized Algebraic Data Types.

Another impressive instance of HOT programming is "Mnemonics: Type-safe bytecode generation at run time", by Johannes Rudolph and Peter Thiemann. They develop a Scala library for generating bytecode for the Java Virtual Machine. The library, as an embedded typed domain-specific language, guarantees that only well formed bytecode is generated. That is, structural constraints that bytecode has to obey concerning the relationship between instructions are automatically enforced, to a large degree by the type correctness of the generating program. Thus, a separate verification pass needed in other approaches either alongside the generation of bytecode, or when loading generated bytecode, is redundant and can be avoided.

We share, with all the authors who submitted papers to this special issue, gratitude towards the reviewers for sharp, intense, and helpful reviewing. We would also like to express our sincere appreciation to Patricia Johann as HOSC associate editor for guiding us along the process of preparing this special issue.

References

1. Gallagher, J.P., Voigtländer, J. (eds.): Partial Evaluation and Program Manipulation, Proceedings. ACM Press, New York (2010)