JFP **18** (5 & 6): 565–566, 2008. © 2008 Cambridge University Press
doi:10.1017/S0956796808006965 First published online 8 September 2008 Printed in the United Kingdom

Editorial

The Eleventh ACM SIGPLAN International Conference on Functional Programming (ICFP 2006) took place on September 18–20, 2006 in Portland, Oregon. ICFP 2006 provides a forum for researchers and developers to hear about the latest work on the design, implementation, principles, and uses of functional programming. The conference covers the entire spectrum of work, from theory to practice, and beyond. There were 74 submissions, of which the program committee selected 24 for presentation at the conference. Following the conference, the program committee selected eight papers for which the authors were invited to submit extended versions for this special issue of JFP. The seven papers that appear in this volume were reviewed, revised, and accepted following standard JFP procedures. These papers cover a wide range of topics related to functional programming, including programming, program verification, language design, language development, program transformation, and program analysis. This range of topics reflects the variety of the work presented at the ICFP conference.

The special issue opens with A Pattern for Almost Compositional Functions by Björn Bringert and Aarne Ranta. This paper addresses a problem familiar to all functional programmers, that of folding an operation over a complex set of data structures, where specific operations are needed at only a few kinds of data objects. They propose a collection of operators that abstract over this kind of programming, and consider how they may be used in both Haskell and Java. The paper concludes with a survey of previous approaches to this problem and an analysis of the relationship between the proposed operators and this previous work.

The second paper is Modular Development of Certified Program Verifiers with a Proof Assistant by Adam Chlipala. This paper provides a case study in the use of the proof assistant Coq to develop correct programming language tools, in this case a verifier for x86 binaries. Rather than simply defining an implementation of a verifier and proving it correct, the paper proposes a collection of modules that transforms a verifier that takes one set of issues into account into a verifier that takes a richer set of issues into account, such as converting from a traditional type system to a type system that understands machine code calling conventions. These modules should be useful in constructing other verifiers, although more modules may be needed for more complex features, such as first-class pointers. The complete source code is available at the JFP web site.

The third paper is Transactional Events by Kevin Donnelly and Matthew Fluet. Concurrency is becoming increasingly important to achieve adequate performance for many applications, and thus it is essential to find language abstractions that allow concurrent programming in an easy and safe way. Transactions prevent unwanted interactions between threads via shared variables, while improving the degree of concurrency as compared to solutions such as global locking. Events are a useful device for structuring the communication between threads. This paper proposes a

combination of these features, implemented as a library for Haskell. This approach addresses some limitations in previous approaches to concurrency in functional languages, such as the inability to implement a three-way rendezvous. The complete source code associated with this work is available at the JFP website.

The fourth paper is Building Language Towers with Ziggurat by David Fisher and Olin Shivers. This paper proposes a framework based on the use of Scheme S-expressions, Scheme macros, and objects providing lazy delegation to allow the implementation of new programming languages as extensions of more basic ones. The use of objects with lazy delegation allows the extensions to be very fine-grained, possibly introducing only a few new constructs at a time. In contrast to other macro-based approaches, in Ziggurat, analysis rules can be associated with the new constructs, allowing the generation of understandable feedback to the user. A number of analyses, such as termination analysis and various kinds of type checking, are presented.

The fifth paper is Algebraic Fusion of Functions with an Accumulating Parameter and Its Improvement by Shin-ya Katsumata and Susumu Nishimura. It is well known that most fusion techniques, i.e., techniques to eliminate intermediate data structures in compositions of functions, do not apply well to functions that construct their result in an accumulating parameter. The authors present a framework for performing fusions involving such functions, and show that some other recently proposed solutions to this problem are instances of this framework.

The sixth paper is Exploiting Reachability and Cardinality in Higher-Order Flow Analysis by Matthew Might and Olin Shivers. The goal of this paper is to improve the precision of abstract interpretation, in particular of flow analysis. The authors consider two optimizations: abstract counting, which allows them to detect the case when only one concrete value is associated with an abstract value, and abstract garbage collection, which allows them to remove concrete values associated with a particular abstraction. Abstract counting provides more informative results, allowing optimizations such as inlining, when the analysis shows that only one function definition can flow to a given call site. Abstract garbage collection provides significant performance improvements. The analysis is proved sound, and implementation issues are described.

The final paper is Hoare Type Theory, Polymorphism and Separation by Aleksandar Nanevski, Greg Morrisett, and Lars Birkedal. The authors present a type system that combines dependent types, which provide the degree of precision that is necessary to find bugs such as array bounds violations, with Hoare logic, which enables reasoning about imperative programs. The type system allows reasoning about both small footprint and large footprint specifications, where the logic describes only the relevant part of the heap or the entire heap, respectively. The paper provides typing rules and proves their soundness.

In conclusion, I would like to thank everyone who participated in the reviewing process, both for ICFP 2006 and for this special issue. Their timely and constructive feedback has contributed significantly to the quality of all of the ICFP 2006 papers.

Julia Lawall
Special Issue Editor