

Preface and Overview of Papers

This volume contains a selection of papers presented at the 1999 International Workshop on the Implementation of Functional Languages (IFL '99), held at Lochem in The Netherlands, September 7–10, 1999. This is the 11th in a series of workshops that has lasted for over a decade, having previously been held in The Netherlands, Germany, Sweden, and the UK. It is also the fourth IFL workshop to be published in the Springer-Verlag series of Lecture Notes in Computer Science (selected papers from IFL '98 are published in LNCS 1595).

The workshop has been growing over the years, and the 1999 workshop demonstrated the increasing relevance and applicability of functional programming by attracting delegates from three different industries as well as from the international functional language research community, the majority of whom presented papers at the workshop. We are pleased to be able to present the high quality selection of refereed and revised papers that appear herein.

While the original focus of the workshop was on parallel implementation, it has broadened over time and currently covers a wide spectrum of topics related to the implementation of functional languages, from theory and language design to applications. The workshop therefore represents a cross-section of the active research community. The papers presented in this volume have been grouped under four topic headings as follows:

Applications. Wiering *et al.* have developed a game library in Clean designed for the creation of parallax scrolling platform games; their results show that a functional-programming approach to game creation is more productive than the use of other low-level libraries (due to greater abstraction) and more flexible than the use of game-creation programs (due to greater programming power).

Compilation Techniques. In keeping with the key theme of the workshop, five selected papers deal with abstract machines and compilation techniques.

Chitil presents a new deforestation method (to elide the construction and deconstruction of intermediate data structures) that is targetted at applications of the `foldr` function; his method works across module boundaries and is even able to deforest definitions of functions that consume their own result.

Peyton Jones *et al.* discuss the issues raised by the implementation of memo functions and present a collection of mechanisms that together support user-programmable memo functions. These mechanisms include the novel concept of “stable” names (references that are unaffected by garbage collection) and a new form of weak pointers (which help the programmer to avoid space leaks).

Van Groningen describes a new optimisation technique that improves the execution time of lazy recursive functions that yield multiple results in a

tuple (thereby often creating unnecessary thunks); in some cases, execution time is improved by a factor of two and allocation costs by a factor of four. Grelck *et al.* describe a code-generation optimisation for WITH-loops in the high-performance language SAC. This optimisation exploits the partial-order evaluation semantics of SAC loops to perform extensive code re-ordering and achieves speedups of up to a factor of 16.

Finally, Kluge presents a “reversible” abstract machine; that is, an abstract machine that can both reduce a program to its weak head normal form and then do inverse reductions which reconstruct the initial program term. This facilitates both operational verification of the abstract machine and program debugging.

Language Concepts. The programme committee have selected four papers that address language-level issues such as GUI-building, foreign-language interfacing, reflection, and concurrency-modelling.

Achten and Plasmeijer explain how interactive objects with *local state* are implemented in the Clean object I/O library, using an elegant meta-circular programming technique that relies on lazy evaluation.

Chakravarty introduces a new method by which Haskell programs can access library routines written in C. This new asymmetric method re-uses existing C interface specifications and uses plain Haskell for the marshalling code, thereby providing a much simpler route for Haskell programmers to call C library routines.

Didrich *et al.* report on an extension of the Opal language that allows the programmer to access “reflective” run-time properties of the program (such as run-time type information). The design and implementation of a reflective system poses significant design and engineering problems (for example, how to support full reflection for polymorphic functions, and how to minimise the related system overheads); these problems and the adopted solutions are discussed in depth. In particular, overheads are only incurred where reflections are actually used.

Finally, Reinke introduces a new approach to the modelling of distributed systems with concurrent activities and internal communication. He uses Haskell as the inscription language to describe data objects and their manipulations in the distributed system, and shows how start-up costs for the use of such “Haskell Coloured Petri Nets” (HCPN) can be dramatically reduced by defining a direct translation from HCPN to Haskell.

Parallelism. The exploitation of parallelism has always been a strong theme of the IFL workshops and the programme committee have chosen the following paper to conclude this volume of selected works:

Hammond and Rebón Portillo describe an implementation of “algorithmic skeletons” that define common parallel forms for programs written in Glasgow Parallel Haskell. These skeletons are provided with cost models to guide efficient implementation; the cost models are entirely decoupled from the compiler to aid portability and user control. Both simulated and real results are presented.

The papers published in this volume were selected using a rigorous a-posteriori refereeing process from the 26 papers that were presented at the workshop. The reviewing was shared among the programme committee, which comprised:

Thomas Arts	Ericsson	Sweden
Chris Clack	University College London	UK
Martin Erwig	Fern Universität Hagen	Germany
Ian Holyer	University of Bristol	UK
Pieter Koopman	University of Nijmegen	The Netherlands
Herbert Kuchen	Westfälische Wilhelms-Universität Münster	Germany
Rita Loogen	University of Marlburg	Germany
Greg Michaelson	University of Edinburgh	UK
Marcus Mohnen	University of Aachen	Germany
John Peterson	Yale University	USA
Sven-Bodo Scholz	University of Kiel	Germany
Colin Runciman	University of York	UK

To ensure a rigorous and objective refereeing process, the programme committee membership was drawn from five representative countries and comprised researchers who were unable to attend the workshop in addition to those who were able to participate. In addition to the members named above, the programme committee also benefitted from the assistance of Peter Achten, Kevin Hammond, John van Groningen, Rinus Plasmeijer, and Malcolm Wallace. The editors were supported by Diederik van Arkel as \LaTeX guru.

The overall balance of the papers is representative, both in scope and technical substance, of the contributions made to the Lochem workshop as well as to those that preceded it. Publication in the LNCS series is not only intended to make these contributions more widely known in the computer science community but also to encourage researchers in the field to participate in future workshops, of which the next one will be held in Aachen, Germany, September, 4th–7th, 2000 (for more information see: <http://www-i2.informatik.rwth-aachen.de/ifl2000>).

April 2000

Pieter Koopman and Chris Clack

Implementation of Functional Languages

11th International Workshop, IFL'99 Lochem, The

Netherlands, September 7-10, 1999 Selected Papers

Koopman, P.; Clack, C. (Eds.)

2000, VIII, 198 p., Softcover

ISBN: 978-3-540-67864-9