
ELSEVIER

Science of Computer Programming 43 (2002) 91–92

Science of Computer Programming

www.elsevier.com/locate/scico

Preface

The construction of sophisticated computer software is a highly creative but, customarily, highly error-prone activity. Mathematically based programming methodologies offer a scientifically sound basis for improving programmer skills and hence the quality and reliability of the software on which our quality of life increasingly depends.

The series of conferences on Mathematics of Program Construction (MPC) was begun in 1989 as a forum for the exchange of ideas and research on the development and application of mathematically based methodologies for program construction. The aim of the conference is to embrace a broad range of activities (from foundational studies through language design to program specification and construction) and so bring together different communities, the binding theme being the quest for the combination of precision and concision in design methodologies for error-free computer software.

Traditionally, the authors of the very best papers presented at the conference and satellite workshops have been asked to submit revised and/or extended journal papers for subsequent publication in a special issue of Science of Computer Programming, these papers being required to have significant added value over and above the conference publication and being subjected to the usual refereeing process. This process led to the publication of the four papers in this volume, which stem from the MPC 2000 conference and satellite workshops.

The four papers are excellent examples of what the conference is about and the high standards it aims to maintain. Each not only develops the mathematical foundations of a novel contribution to programming methodology but also shows how the techniques are applied in non-trivial programming applications.

The use of type systems has long been recognized as important to increasing the reliability of programs. This is true in all programming paradigms but is particularly well developed in functional programming languages. The language O'Haskell is a development of the functional language Haskell which includes object-oriented and concurrent programming features. Johan Nordlander's paper "Polymorphic subtyping in O'Haskell" shows how the language is furnished with a safe, polymorphic type system supported by a powerful, partial type inference algorithm. This is illustrated by elements of a statically typed interface to Ousterhout's graphical tool kit, Tk.

Ralf Hinze's paper "Polytypic values possess polykinded types" presents a significant advance on our understanding of the nature of so-called "polytypic" programs and how they are constructed. Polytypic programs are programs that are defined by induction on the structure of types. Examples are the automatically "derived" programs in Haskell, like the equality function. Hinze shows how polytypic values have types that are indexed by kinds (cf. parametrically polymorphic functions have types that are

0167-6423/02/$ - see front matter
PII: S0167-6423(02)00027-8

indexed by types). He then shows how the kind information is used to systematically construct the definitions of polytypic programs. Reasoning about such programs, using logical relations, is also discussed.

The paper "Reasoning about real-time repetitions: terminating and nonterminating" by Ian Hayes extends the use of invariant properties (well-known to be crucial in the design of, e.g. repetitions and communicating processes) to real-time, non-terminating repetitions. The method is applied to the design of a program to sort items on a conveyor belt according to size. A key property of the methodology is that it reduces to the standard loop-invariant rule in the case of terminating non-real-time repetitions.

The construction of programs by inverting existing programs is an attractive idea that has been studied intermittently almost since the inception of computing as a science. The paper "The universal resolving algorithm and its correctness: inverse computation in a functional language" by Sergei Abramov and Robert Glück aims to mechanize the process. The universal resolving algorithm is an algorithm for inverse computation in a first-order functional language. The algorithm is based on constructing a "perfect process tree" which represents the computation of a program with partially specified input by a tree of all computation traces. The paper provides a detailed formal justification of the algorithm and illustrates its application to pattern matching.

Our thanks go to the program committees of the MPC conference and its satellite workshops for helping us make the selection of papers for this special issue, to the authors and referees of the papers for their timely cooperation, and to the managing editors of SCP for their support in producing this special issue.

Roland Backhouse
University of Nottingham
Nottingham, UK
E-mail address: rcb@cs.nott.ac.uk

José Oliveira
Universidade do Minho
Braga, Portugal
E-mail address: jno@di.uminho.pt