ELSEVIER

The Journal of Logic and Algebraic Programming 51 (2002) 123–124

THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING

www.elsevier.com/locate/jlap

Guest editor's introduction

C.A. Middelburg*

Computing Science Department, Eindhoven University of Technology, P.O. Box 80126, 5600 MB Eindhoven, The Netherlands

Program algebra is a novel algebraic framework for sequential programming. It is intended to contribute to a better understanding of sequential programming. The work on program algebra was initiated by Bergstra. This special issue aims to give a concise overview of the work done on program algebra so far.

Starting from simple algebraic foundations, an original and systematic analysis of a number of basic and more advanced programming language constructs has been carried out in a precise, but moderately formal way. In addition, a systematic and detailed analysis of how the behaviour of a program and a state machine can interact has been carried out. An interesting extension of the framework with primitives for object-based programming, based on a novel model of computation, is studied as well.

In the first paper, Bergstra and Loots introduce the basic program algebra PGA (ProGram Algebra). The terms of PGA denote single pass instruction sequences. The behaviour represented by single pass instruction sequences is described in terms of the constants and operators of the simple process algebra BPPA (Basic Polarized Process Algebra). The terms of PGA can be regarded as programs in a very simple basic programming language called PGLA. It is demonstrated how a number of more advanced programming languages can be understood by mappings to PGLA, called program algebra projections, in a step-by-step fashion.

In the second paper, Ponse extends program algebra PGA with a unit instruction operator which wraps a PGA program into a unit of length one. This extension is useful to introduce composed tests. The extended version is called $\mathrm{PGA}_{\mathrm{u}}$. Both a projection of $\mathrm{PGA}_{\mathrm{u}}$ into PGA that maps a program one instruction at a time from left to right and a projection that maps a program as a whole are defined.

The third paper complements the first paper in an interesting way. Bergstra and Ponse investigate how a state machine may affect the behaviour of a program and how the behaviour of a program may transform a state machine into another one. The particular programming language used, called PGLE, originates from the first paper. Operators corresponding to the two above-mentioned ways in which the behaviour of a program and a state machine may interact are introduced. It is demonstrated that there are PGLE programs using state machines for which PGLE programs not using state machines are disappointingly long or impossible.

* Tel.: +31-40-247-5157; fax: +31-40-247-5361.
E-mail address: keesm@win.tue.nl

PII: S1567-8326(02)00017-6

In the fourth paper, Bergstra and Bethke first introduce a model of computation suitable for object-based programs. In this model, a state resembles a fluid consisting of a collection of molecules, each molecule being a collection of atoms with bindings between them. A computation is an evolution of the fluid by a chain of reactions. Next, they extend the programming language PGLEc, which is much like PGLE, with a collection of molecular programming primitives. An assertion language to express properties of the fluid during a computation is proposed.

Editorial remark by J.A. Bergstra and J.V. Tucker

This is the first of our special issues. It is the policy of *The Journal of Logic and Algebraic Programming* to encourage special issue contributions reporting or launching a coherent topic, or originating mainly from a single research group or geographic site, and, of course, to consider editors as part of the authoring community. We welcome suggestions for further special issues of the Journal.