MPC 2000

FIFTH INTERNATIONAL CONFERENCE ON

MATHEMATICS OF PROGRAM CONSTRUCTION

3–5th July 2000

Ponte de Lima, Portugal

`http://www.di.uminho.pt/mpc2000`

Conference Programme

Monday July 3 2000

8.50 - 9.00

Welcome

SESSION 1. Chair: *Roland Backhouse* (University of Nottingham)

9.00 - 10.00

Invited talk: *Integrating Programming, Properties, and Validation*
Mark Jones (Oregon Graduate Institute, USA)

Abstract: Current program development environments provide excellent support for many desirable aspects of modern software applications such as performance and interoperability, but almost no support for features that could directly enhance correctness and reliability. In this talk, I will describe the first steps that we are making in a project to develop a new kind of program development environment. Our goal is to produce a tool that actively supports and encourages its users in thinking about, stating, and validating key properties of software as an integral part of the programming process.

The environment that we are designing will allow programmers to assert properties of program elements as part of their source code, capturing intuitions and insights about its behavior at the time it is written. These property assertions will also provide an opportunity to give more precise interfaces to software components and libraries. Even by themselves, assertions can provide valuable documentation, and can be type checked to ensure a base level of consistency with executable portions of the program. Critically, however, our environment will allow property assertions to be annotated with "certificates" that provide evidence of validity. By adopting a generic interface, many different forms of certificate will be supported, offering a wide range of validation options—from low-cost instrumentation and automated testing, to machine-assisted proof and formal methods. Individual properties and certificates may pass through several points on this spectrum as development progresses, and as higher levels of assurance are required. To complete the environment, a suite of "property management" tools will provide users with facilities to browse or report on the status of properties and associated certificates within a program, and to explore different validation strategies.

We plan to evaluate our system by applying it to some real-world, security related problems, and we hope to demonstrate that it can contribute to the development of more robust and dependable software applications. For example, a tighter integration between programming and validation should provide the mechanisms that we need to move towards a model for accountability in software, and away from the ubiquitous "as is" warranties that we see today. More specifically, this could allow vendors to provide and sell software on the strength of firm *guarantees* about its behavior, and allow users to identify the responsible parties when a software system fails.

10.00 - 10.30

Break

SESSION 2. Chair: *Johan Jeuring* (Utrecht University)

10.30 - 11.15

Polytypic values possess polykinded types
Ralf Hinze (Institute für Informatik III, Universität Bonn)

Abstract: A polytypic value is one that is defined by induction on the structure of types. In Haskell types are assigned so-called kinds that distinguish between manifest types like the type of integers and functions on types like the list type constructor. Previous approaches to polytypic programming were restricted in that they only allowed to parameterize values by types of one fixed kind. In this paper we show how to define values that are indexed by types of arbitrary kinds. It turns out that these polytypic values possess types that are indexed by kinds. We present several examples that demonstrate that the additional flexibility is useful in practice. One paradigmatic example is the mapping function, which describes the functorial action on arrows. A single polytypic definition yields mapping functions for datatypes of arbitrary kinds including first- and higher-order functors. Polytypic values enjoy polytypic properties. Using kind-indexed logical relations we prove among other things that the polytypic mapping function satisfies suitable generalizations of the functorial laws.

11.15 - 12.00

The Zip Calculus
Mark Tullsen (Department of Computer Science, Yale University)

Abstract: Many have recognized the need for genericity in programming and program transformation. Genericity over data types has been achieved with polymorphism. Genericity over type constructors, often called polytypism, is an area of active research. This paper proposes that another kind of genericity is needed: genericity over the length of tuples. Untyped languages allow for such genericity but typed languages do not (except for languages allowing dependent types). The contribution of this paper is to present the "zip calculus," a typed lambda calculus that provides genericity over the length of tuples and yet does not require the full generality of dependent types.

12.00 - 14.00

Lunch

SESSION 3. Chair: *Lindsay Groves* (Victoria University of Wellington)

14.00 - 14.45

Separation and Reduction
Ernie Cohen (Telecordia Technologies)

Abstract: We present some new theorems that equate an iteration to a sequential composition of stronger iterations, and use these theorems to simplify and generalize a number of known techniques for pretending atomicity in concurrent programs.

14.45 - 15.30

Reasoning about Non-terminating Loops Using Deadline Commands
Ian Hayes (Dept. of Computer Science and Electrical Engineering, The University of Queensland, Brisbane)

Abstract: It is common for a real-time process to consist of a nonterminating loop monitoring an input and controlling an output. Hence a real-time program development method needs to support nonterminating loops. Earlier work on real-time program development has produced a real-time refinement calculus that makes use of a novel deadline command which allows timing constraints to be embedded in real-time programs. The addition of the deadline command to the real-time programming language gives the significant advantage of providing a real-time programming language that is machine independent. This allows a more abstract approach to the program development process.

In this paper we add possibly nonterminating loops to the refinement calculus. First we examine the semantics of possibly nonterminating loops, and use them to reason directly about a simple example. Then we develop simpler refinement rules that make use of a loop invariant.

15.30 - 16.00

Break

SESSION 4. Chair: *Christian Lengauer* (Universität Passau)

16.00 - 16.45

Quantum Programming
J. Sanders and *P. Zuliani* (PRG, Oxford University Computing Laboratory, Oxford)

Abstract: In this paper a programming language, *qGCL*, is presented for the expression of quantum algorithms. It contains the features required to program a 'universal' quantum computer (including initialisation and observation), has a formal semantics and body of laws, and provides a refinement calculus supporting the verification and derivation of programs against their specifications. A representative selection of quantum algorithms are expressed in the language and one of them is derived from its specification.

16.45-17.30

"*Hot topic*" Session. Coordination: *Richard Bird* (Oxford University Computing Laboratory)

Tuesday July 4 2000

SESSION 5. Chair: *Lambert Meertens* (Kestrel Institute, USA)

9.00 - 10.00

Invited talk: *Regular Expressions Revisited: a Coinductive Approach to Streams, Automata, and Power Series*
Jan Rutten (CWI, The Netherlands)

Abstract: Regular expressions are a standard means for denoting formal languages that are recognizable by finite automata. Much less familiar is the use of syntactic expressions for (formal) power series. Power series generalize languages by assigning to words multiplicities in any semiring (such as the reals) rather than just Booleans, and include as a special case the set of streams (infinite sequences). Here we shall define an extended set of regular expressions with multiplicities in an arbitrary semiring. The semantics of such expressions will be defined coinductively, allowing for the use of a syntactic coinductive proof principle. To each expression will be assigned a nondeterministic automaton with multiplicities, which usually is a rather efficient representation of the power series denoted by the expression. Much of the above will be illustrated for the special case of streams of real numbers; other examples include automata and languages (sets of words), and task-resource systems (using the max-plus semiring). The coinductive definitions mentioned above take the shape of what we have called behavioural differential equations, on the basis of which we develop, as a motivating example, a theory of streams in a calculus-like fashion.
Our perspective is essentially coalgebraic. More precisely, the set of all formal power series, including the set of languages and the set of streams as special instances, is a final coalgebra. This fact is the basis for both the coinduction definition and the coinduction proof principle.
For general background on coalgebra, see [JR97] and [Rut96c]. The proceedings of the recently established workshop series CMCS (Coalgebraic Methods in Computer Science), contained in Volumes 11, 19, and 33 of Elsevier's Electronic Notes in Theoretical Computer Science, give a good impression of many of the latest developments in coalgebraic studies. References related to the theory summarized above are [Rut98b], dealing with automata and languages, and [Rut99], on formal power series. A technical report on behavioural differential equations is in preparation.

10.00 - 10.30

Break

SESSION 6. Chair: *Bernhard Möller* (Universität Augsburg)

10.30 - 11.15

Proving pointer programs in Hoare Logic
Richard Bornat (Department of Computer Science, Queen Mary and Westfield College,University of London)

Abstract: It is possible, but difficult, to reason in Hoare logic about programs which address and modify data structures defined by pointers. The challenge is to approach the simplicity of Hoare logic's treatment of variable assignment, where substitution affects only relevant assertion formulæ. The axiom of assignment to object components treats each component name as a pointer**?**indexed array. This permits a formal treatment of inductively defined data structures in the heap but tends to produce instances of modified component mappings in arguments to inductively defined assertions. The major weapons against these troublesome mappings are assertions which describe spatial separation of data structures. Three example proofs are sketched.

11.15 - 12.00

On Guarded Commands with Fair Choice
Emil Sekerinski (Department of Computing and Software, McMaster University, Hamilton, Ontario)

Abstract: For the purpose of program development, fairness is typically formalized by verification rules or, alternatively, through refinement rules. In this paper we give an account of (weak) fairness in an algebraic style, extending recently proposed algebraic accounts of iterations and loops using the predicate transformer model of statements.

12.00 - 14.00

Lunch

Wednesday July 5 2000

SESSION 7. Chair: *J.N. Oliveira* (Universidade do Minho)

9.00 - 10.00

Invited talk: *Formal Methods and Dependability*
Cliff Jones (University of Newcastle, UK)

Abstract: This paper sets out a programme of work in the area of dependability. The research is to be pursued under the aegis of a six-year *Inter-Disciplinary Research Collaboration* funded by the UK Engineering and Physical Sciences Research Council. The aim is to to consider computer-based systems which comprise humans as well as hardware and software. The aim here is to indicate how formal methods ideas, coupled with structuring proposals, can help address a problem which clearly also requires social science input.

10.00 - 10.30

Break

SESSION 8. Chair: *Oege de Moor* (Oxford University Computing Laboratory)

10.30 - 11.15

Liberating Data Refinement
Eerke Boiten and *John Derrick* (Computing Laboratory, University of Kent, Canterbury)

Abstract: Traditional rules for refinement of abstract data types suggest a software development process in which much of the detail has to be present already in the initial specification. In particular, the set of available operations and their interfaces need to be fixed. In contrast, many formal and informal software development methods rely on changes of granularity and require introduction of detail in a gradual way during the development process.
This paper discusses several generalisations and extensions of the traditional refinement rules, which are compatible with each other and, more importantly, with the semantic grounding of data refinement. Together they should provide a semantic justification for a larger spectrum of development steps.
The discussion takes place in the context of the formal specification language Z and its relational underpinnings.

11.15 - 12.00

Theorems about composition
Michel Charpentier (Computer Science Department, University of New Hampshire, Durham) and *K. Mani Chandy* (Computer Science Department, California Institute of Technology, Pasadena)

Abstract: Compositional designs require component specifications that can be composed: Designers have to be able to deduce system properties from components specifications. On the other hand, components specifications should be abstract enough to allow component reuse and to hide substantial parts of correctness proofs in components verifications. Part of the problem is that too abstract specifications do not contain enough information to be composed. Therefore, the right balance between abstraction and composability must be found. This paper explores the systematic construction of abstract specifications that can be composed through specific forms of composition called *existential* and *universal*.

12.00 - 14.00

Lunch

SESSION 9. Chair: *Jeremy Gibbons* (Oxford University Computing Laboratory)

14.00 - 14.45

The Universal Resolving Algorithm: Inverse Computation in a Functional Language
Sergei Abramov (Program Systems Institute, Russian Academy of Sciences) and *Robert Glück* (Department of Information and Computer Science, Waseda University, Tokyo)

Abstract: We present an algorithm for inverse computation in a first-order functional language based on the notion of a perfect process tree. The Universal Resolving Algorithm (URA) introduced in this paper is sound and complete, and computes each solution, if it exists, in finite time. The algorithm has been implemented for TSG, a typed dialect of S-Graph, and shows some remarkable results for the inverse computation of functional programs such as pattern matching and the inverse interpretation of While-programs.

14.45 - 15.30

Metacomputation-based Compiler Architecture
William L. Harrison (Department of Computer Science, Indiana University, Bloomington) and *Samuel N. Kamin* (Department of Computer Science, University of Illinois, Urbana)

Abstract: This paper presents a modular and extensible style of language specification based on metacomputations. This style uses two monads to factor the static and dynamic parts of the specification, thereby staging the specification and achieving strong binding-time separation. Because metacomputations are defined in terms of monads, they can be constructed modularly and extensibly using monad transformers. A number of language constructs are specified: expressions, control-flow, imperative features, and block structure. Metacomputation-style specification lends itself to semantics-directed compilation, which we demonstrate by creating a modular compiler for a block-structured, imperative while language.

15.30 - 16.00

Break

SESSION 10. Chair: *Eugenio Moggi* (DISI, Univ. di Genova)

16.00 - 16.45

A Meta Language for Programming with Bound Names Modulo Renaming (Preliminary Report)
Andrew M. Pitts (Cambridge University Computer Laboratory, Cambridge) and *Murdoch J. Gabbay* (Department of Pure Mathematics and Mathematical Statistics, Cambridge University)

Abstract: This paper describes work in progress on the design of an ML-style metalanguage Fresh-ML for programming with recursively defined functions on user-defined, concrete data types whose constructors may involve variable binding. Up to operational equivalence, values of such Fresh-ML data types can faithfully encode terms modulo $alpha$-conversion for a wide range of object languages in a straightforward fashion. The design of Fresh-ML is 'semantically driven', in that it arises from the model of variable binding in set theory with atoms given by the authors in [GP99]. The language has a type constructor for abstractions over names (= atoms) and facilities for declaring locally fresh names. Moreover, recursive definitions can use a form of pattern-matching on bound names in abstractions. The crucial point is that the Fresh-ML type system ensures that these features can only be used in well-typed programs in ways that are insensitive to renaming of bound names.

16.45-17.30

"*Hot topic*" Session. Coordination: *Richard Bird* (Oxford University Computing Laboratory)

17.30-

Closing session - End of MATHEMATICS OF PROGRAM CONSTRUCTION 2000