Lecture Notes in Computer Science 755

Bernhard Möller Helmut Partsch
Steve Schuman (Eds.)

Formal Program Development

IFIP TC2/WG 2.1 State-of-the-Art Report

Springer-Verlag

Bernhard Möller Helmut Partsch
Steve Schuman (Eds.)

Formal Program Development

IFIP TC2/WG 2.1 State-of-the-Art Report

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Volume Editors

Bernhard Möller
Institut für Mathematik, Universität Augsburg
Universitätsstr. 2, D-86135 Augsburg, Germany

Helmut Partsch
Fakultät für Informatik, Universität Ulm
Oberer Eselsberg, D-89069 Ulm, Germany

Steve Schuman
Department of Mathematics, University of Surrey
Guildford, Surrey GU2 5XH, United Kingdom

CR Subject Classification (1991): F.3.1, D.2.1, D.2.4, D.2.2, D.1.1, D.2.10, G.2.m, I.1.3, D.2.6

ISBN 3-540-57499-9 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57499-9 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera-ready by author
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Lecture Notes in Computer Science 755
Edited by G. Goos and J. Hartmanis

Preface

Since the late 1960s, much has been done to establish the new field of software engineering. The common objective has been to overcome the well-documented difficulties of software development by adopting the methods, techniques and professional practices of more traditional engineering disciplines. Such efforts have led to significant improvements insofar as project organization and management are concerned, but unfortunately, too many programs produced today still do not behave as expected. However, the last decade has also seen the emergence of a number of approaches wherein software development is viewed as a fully *formal* activity. All of these approaches have *correctness* as their primary focus – that is, their aim is to obtain programs which provably satisfy some given *specification* (a formal statement of the problem to be solved). Thus research in this area necessarily encompasses two principal concerns:

– formal specification of solutions to problems, and
– formal development/calculation of programs from such specifications.

These two topics form the core interests now represented within IFIP Working Group 2.1 on *Algorithmic Languages and Calculi.*

In former times, IFIP Working Group 2.1 was mainly concerned with definition of the *algorithmic languages* ALGOL 60 and ALGOL 68, for which it still has international responsibility. In those days the syntax and semantics of programming languages were at the forefront of computing science research. Since 1975, the Working Group has increasingly focused on *systematic approaches* to programming in its broader sense, and on appropriate *concepts and notations* to support such approaches. Today, the *calculation of programs from specifications* constitutes the central theme of the group's work. This is reflected in its official Aim and Scope, which are as follows:

Aim:
To explore and evaluate new ideas in the field of programming, possibly leading to the design of new languages.

Scope:
1. The study of calculation of programs from specifications.
2. The design of notations for such calculation.

3. The formulation of algorithm theories, using such notations.
4. The investigation of software support for program derivation.
5. Continuing responsibility for ALGOL 60 and ALGOL 68.

For some time the group had felt that its work had reached a state to be presented to a wider audience in the form of in-depth surveys of the various strains of thought. This met well with the emergence of a global activity of IFIP of sponsoring State-of-the-Art Seminars in developing countries. As a result, such a seminar was conceived as to provide access to the foremost front of research on *Formal Program Development.* This book contains background texts for the seminar lectures.

The first actual presentation of the seminar took place in January 1992 near Rio de Janeiro, Brazil. It was hosted by Armando Haeberer from the Pontifíca Universidade Católica at Rio de Janeiro and took place in most splendid tropical surroundings on Itacuruçá Island. We are most grateful to Armando for making this seminar possible and for his excellent arrangements. We also express our gratitude to IBM-Brazil, Conselho Naçional de Pesquisa e Disenvolvimiento and Pontifíca Universidade Católica at Rio de Janeiro for their generous support. Finally we wish to thank the referees for their detailed evaluations and M. Russling for his help in preparing the manuscript for this volume.

Augsburg, Ulm and Surrey, July 1993

Bernhard Möller, Helmut Partsch, Steve Schuman

Lecture Notes in Computer Science

For information about Vols. 1–685
please contact your bookseller or Springer-Verlag

Vol. 686: J. Mira, J. Cabestany, A. Prieto (Eds.), New Trends in Neural Computation. Proceedings, 1993. XVII, 746 pages. 1993.

Vol. 687: H. H. Barrett, A. F. Gmitro (Eds.), Information Processing in Medical Imaging. Proceedings, 1993. XVI, 567 pages. 1993.

Vol. 688: M. Gauthier (Ed.), Ada-Europe '93. Proceedings, 1993. VIII, 353 pages. 1993.

Vol. 689: J. Komorowski, Z. W. Ras (Eds.), Methodologies for Intelligent Systems. Proceedings, 1993. XI, 653 pages. 1993. (Subseries LNAI).

Vol. 690: C. Kirchner (Ed.), Rewriting Techniques and Applications. Proceedings, 1993. XI, 488 pages. 1993.

Vol. 691: M. Ajmone Marsan (Ed.), Application and Theory of Petri Nets 1993. Proceedings, 1993. IX, 591 pages. 1993.

Vol. 692: D. Abel, B.C. Ooi (Eds.), Advances in Spatial Databases. Proceedings, 1993. XIII, 529 pages. 1993.

Vol. 693: P. E. Lauer (Ed.), Functional Programming, Concurrency, Simulation and Automated Reasoning. Proceedings, 1991/1992. XI, 398 pages. 1993.

Vol. 694: A. Bode, M. Reeve, G. Wolf (Eds.), PARLE '93. Parallel Architectures and Languages Europe. Proceedings, 1993. XVII, 770 pages. 1993.

Vol. 695: E. P. Klement, W. Slany (Eds.), Fuzzy Logic in Artificial Intelligence. Proceedings, 1993. VIII, 192 pages. 1993. (Subseries LNAI).

Vol. 696: M. Worboys, A. F. Grundy (Eds.), Advances in Databases. Proceedings, 1993. X, 276 pages. 1993.

Vol. 697: C. Courcoubetis (Ed.), Computer Aided Verification. Proceedings, 1993. IX, 504 pages. 1993.

Vol. 698: A. Voronkov (Ed.), Logic Programming and Automated Reasoning. Proceedings, 1993. XIII, 386 pages. 1993. (Subseries LNAI).

Vol. 699: G. W. Mineau, B. Moulin, J. F. Sowa (Eds.), Conceptual Graphs for Knowledge Representation. Proceedings, 1993. IX, 451 pages. 1993. (Subseries LNAI).

Vol. 700: A. Lingas, R. Karlsson, S. Carlsson (Eds.), Automata, Languages and Programming. Proceedings, 1993. XII, 697 pages. 1993.

Vol. 701: P. Atzeni (Ed.), LOGIDATA+: Deductive Databases with Complex Objects. VIII, 273 pages. 1993.

Vol. 702: E. Börger, G. Jäger, H. Kleine Büning, S. Martini, M. M. Richter (Eds.), Computer Science Logic. Proceedings, 1992. VIII, 439 pages. 1993.

Vol. 703: M. de Berg, Ray Shooting, Depth Orders and Hidden Surface Removal. X, 201 pages. 1993.

Vol. 704: F. N. Paulisch, The Design of an Extendible Graph Editor. XV, 184 pages. 1993.

Vol. 705: H. Grünbacher, R. W. Hartenstein (Eds.), Field-Programmable Gate Arrays. Proceedings, 1992. VIII, 218 pages. 1993.

Vol. 706: H. D. Rombach, V. R. Basili, R. W. Selby (Eds.), Experimental Software Engineering Issues. Proceedings, 1992. XVIII, 261 pages. 1993.

Vol. 707: O. M. Nierstrasz (Ed.), ECOOP '93 – Object-Oriented Programming. Proceedings, 1993. XI, 531 pages. 1993.

Vol. 708: C. Laugier (Ed.), Geometric Reasoning for Perception and Action. Proceedings, 1991. VIII, 281 pages. 1993.

Vol. 709: F. Dehne, J.-R. Sack, N. Santoro, S. Whitesides (Eds.), Algorithms and Data Structures. Proceedings, 1993. XII, 634 pages. 1993.

Vol. 710: Z. Ésik (Ed.), Fundamentals of Computation Theory. Proceedings, 1993. IX, 471 pages. 1993.

Vol. 711: A. M. Borzyszkowski, S. Sokołowski (Eds.), Mathematical Foundations of Computer Science 1993. Proceedings, 1993. XIII, 782 pages. 1993.

Vol. 712: P. V. Rangan (Ed.), Network and Operating System Support for Digital Audio and Video. Proceedings, 1992. X, 416 pages. 1993.

Vol. 713: G. Gottlob, A. Leitsch, D. Mundici (Eds.), Computational Logic and Proof Theory. Proceedings, 1993. XI, 348 pages. 1993.

Vol. 714: M. Bruynooghe, J. Penjam (Eds.), Programming Language Implementation and Logic Programming. Proceedings, 1993. XI, 421 pages. 1993.

Vol. 715: E. Best (Ed.), CONCUR'93. Proceedings, 1993. IX, 541 pages. 1993.

Vol. 716: A. U. Frank, I. Campari (Eds.), Spatial Information Theory. Proceedings, 1993. XI, 478 pages. 1993.

Vol. 717: I. Sommerville, M. Paul (Eds.), Software Engineering – ESEC '93. Proceedings, 1993. XII, 516 pages. 1993.

Vol. 718: J. Seberry, Y. Zheng (Eds.), Advances in Cryptology – AUSCRYPT '92. Proceedings, 1992. XIII, 543 pages. 1993.

Vol. 719: D. Chetverikov, W.G. Kropatsch (Eds.), Computer Analysis of Images and Patterns. Proceedings, 1993. XVI, 857 pages. 1993.

Vol. 720: V.Mařík, J. Lažanský, R.R. Wagner (Eds.), Database and Expert Systems Applications. Proceedings, 1993. XV, 768 pages. 1993.

Vol. 721: J. Fitch (Ed.), Design and Implementation of Symbolic Computation Systems. Proceedings, 1992. VIII, 215 pages. 1993.

Vol. 722: A. Miola (Ed.), Design and Implementation of Symbolic Computation Systems. Proceedings, 1993. XII, 384 pages. 1993.

Vol. 723: N. Aussenac, G. Boy, B. Gaines, M. Linster, J.-G. Ganascia, Y. Kodratoff (Eds.), Knowledge Acquisition for Knowledge-Based Systems. Proceedings, 1993. XIII, 446 pages. 1993. (Subseries LNAI).

Vol. 724: P. Cousot, M. Falaschi, G. Filè, A. Rauzy (Eds.), Static Analysis. Proceedings, 1993. IX, 283 pages. 1993.

Vol. 725: A. Schiper (Ed.), Distributed Algorithms. Proceedings, 1993. VIII, 325 pages. 1993.

Vol. 726: T. Lengauer (Ed.), Algorithms – ESA '93. Proceedings, 1993. IX, 419 pages. 1993

Vol. 727: M. Filgueiras, L. Damas (Eds.), Progress in Artificial Intelligence. Proceedings, 1993. X, 362 pages. 1993. (Subseries LNAI).

Vol. 728: P. Torasso (Ed.), Advances in Artificial Intelligence. Proceedings, 1993. XI, 336 pages. 1993. (Subseries LNAI).

Vol. 729: L. Donatiello, R. Nelson (Eds.), Performance Evaluation of Computer and Communication Systems. Proceedings, 1993. VIII, 675 pages. 1993.

Vol. 730: D. B. Lomet (Ed.), Foundations of Data Organization and Algorithms. Proceedings, 1993. XII, 412 pages. 1993.

Vol. 731: A. Schill (Ed.), DCE – The OSF Distributed Computing Environment. Proceedings, 1993. VIII, 285 pages. 1993.

Vol. 732: A. Bode, M. Dal Cin (Eds.), Parallel Computer Architectures. IX, 311 pages. 1993.

Vol. 733: Th. Grechenig, M. Tscheligi (Eds.), Human Computer Interaction. Proceedings, 1993. XIV, 450 pages. 1993.

Vol. 734: J. Volkert (Ed.), Parallel Computation. Proceedings, 1993. VIII, 248 pages. 1993.

Vol. 735: D. Bjørner, M. Broy, I. V. Pottosin (Eds.), Formal Methods in Programming and Their Applications. Proceedings, 1993. IX, 434 pages. 1993.

Vol. 736: R. L. Grossman, A. Nerode, A. P. Ravn, H. Rischel (Eds.), Hybrid Systems. VIII, 474 pages. 1993.

Vol. 737: J. Calmet, J. A. Campbell (Eds.), Artificial Intelligence and Symbolic Mathematical Computing. Proceedings, 1992. VIII, 305 pages. 1993.

Vol. 738: M. Weber, M. Simons, Ch. Lafontaine, The Generic Development Language Deva. XI, 246 pages. 1993.

Vol. 739: H. Imai, R. L. Rivest, T. Matsumoto (Eds.), Advances in Cryptology – ASIACRYPT '91. X, 499 pages. 1993.

Vol. 740: E. F. Brickell (Ed.), Advances in Cryptology – CRYPTO '92. Proceedings, 1992. X, 593 pages. 1993.

Vol. 741: B. Preneel, R. Govaerts, J. Vandewalle (Eds.), Computer Security and Industrial Cryptography. Proceedings, 1991. VIII, 275 pages. 1993.

Vol. 742: S. Nishio, A. Yonezawa (Eds.), Object Technologies for Advanced Software. Proceedings, 1993. X, 543 pages. 1993.

Vol. 743: S. Doshita, K. Furukawa, K. P. Jantke, T. Nishida (Eds.), Algorithmic Learning Theory. Proceedings, 1992. X, 260 pages. 1993. (Subseries LNAI)

Vol. 744: K. P. Jantke, T. Yokomori, S. Kobayashi, E. Tomita (Eds.), Algorithmic Learning Theory. Proceedings, 1993. XI, 423 pages. 1993. (Subseries LNAI)

Vol. 745: V. Roberto (Ed.), Intelligent Perceptual Systems. VIII, 378 pages. 1993. (Subseries LNAI)

Vol. 746: A. S. Tanguiane, Artificial Perception and Music Recognition. XV, 210 pages. 1993. (Subseries LNAI).

Vol. 747: M. Clarke, R. Kruse, S. Moral (Eds.), Symbolic and Quantitative Approaches to Reasoning and Uncertainty. Proceedings, 1993. X, 390 pages. 1993.

Vol. 748: R. H. Halstead Jr., T. Ito (Eds.), Parallel Symbolic Computing: Languages, Systems, and Applications. Proceedings, 1992. X, 419 pages. 1993.

Vol. 749: P. A. Fritzson (Ed.), Automated and Algorithmic Debugging. Proceedings, 1993. VIII, 369 pages. 1993.

Vol. 750: J. L. Díaz-Herrera (Ed.), Software Engineering Education. Proceedings, 1994. XII, 601 pages. 1994.

Vol. 751: B. Jähne, Spatio-Temporal Image Processing. XII, 208 pages. 1993.

Vol. 752: T. W. Finin, C. K. Nicholas, Y. Yesha (Eds.), Information and Knowledge Management. Proceedings, 1992. VII, 142 pages. 1993.

Vol. 753: L. J. Bass, J. Gornostaev, C. Unger (Eds.), Human-Computer Interaction. Proceedings, 1993. X, 388 pages. 1993.

Vol. 754: H. D. Pfeiffer, T. E. Nagle (Eds.), Conceptual Structures: Theory and Implementation. Proceedings, 1992. IX, 327 pages. 1993. (Subseries LNAI).

Vol. 755: B. Möller, H. Partsch, S. Schuman (Eds.), Formal Program Development. Proceedings. VII, 371 pages. 1993.

Vol. 756: J. Pieprzyk, B. Sadeghiyan, Design of Hashing Algorithms. XV, 194 pages. 1993.

Vol. 757: U. Banerjee, D. Gelernter, A. Nicolau, D. Padua (Eds.), Languages and Compilers for Parallel Computing. Proceedings, 1992. X, 576 pages. 1993.

Vol. 758: M. Teillaud, Towards Dynamic Randomized Algorithms in Computational Geometry. IX, 157 pages. 1993.

Vol. 759: N. R. Adam, B. K. Bhargava (Eds.), Advanced Database Systems. XV, 451 pages. 1993.

Vol. 760: S. Ceri, K. Tanaka, S. Tsur (Eds.), Deductive and Object-Oriented Databases. Proceedings, 1993. XII, 488 pages. 1993.

Vol. 761: R. K. Shyamasundar (Ed.), Foundations of Software Technology and Theoretical Computer Science. Proceedings, 1993. XIV, 456 pages. 1993.

Vol. 762: K. W. Ng, P. Raghavan, N. V. Balasubramanian, F. Y. L. Chin (Eds.), Algorithms and Computation. Proceedings, 1993. XIII, 542 pages. 1993.

Table of Contents

Introduction

Bernhard Möller[1], *Helmut A. Partsch*[2], *Stephen A. Schuman*[3]

[1] Institut für Mathematik, Universität Augsburg, D-86135 Augsburg, Germany
[2] Fakultät für Informatik, Universität Ulm, D-89069 Ulm, Germany
[3] Dept. of Mathematics, University of Surrey, Guildford, Surrey GU2 5XH, U.K.

1 The Topics

The book attempts to survey the area of *Formal Program Development.* The most important subthemes of this area are

- formal specifications (as starting points for subsequent calculations),
- particular calculi and their theoretical foundations (inclusive of development calculi),
- rules and strategies (contents of calculi) used in such calculations, and
- systems to support these formal calculations.

1.1 Formal Specifications

A *formal specification* is intended to give a precise description of a problem to be solved by some piece of software. Research on formal specification deals mainly with appropriate concepts and the corresponding language constructs, including their theoretical foundations. These aspects deal with the use of specification constructs in describing concrete problems, with the construction of specifications in a systematic way. Additional aspects of this research are concerned with methodological issues such as acquisition, evolution, development and validation of formal specifications. Today, there is no commonly agreed general-purpose specification formalism. Rather, there is a spectrum of specific techniques based on different theoretical foundations and aimed at describing particular classes of problems (or systems) in the most appropriate way. In this book, formal specifications are explicitly addressed in the contributions by PARTSCH, PEPPER and FEATHER. In the algebraic approach, as dealt with in the papers by PARTSCH and PEPPER, a software system is specified in terms of its components through a hierarchically related structure of *algebraic types.* In each of these types, objects, object classes and operations are defined through algebraic axioms. These axioms describe the essential properties of the individual operations without refering to their operational realization. Propositions on the overall behaviour may then be inferred from the individual axioms, the interactions of the various operations and the hierarchical relationships between the component types.

In the behaviour-oriented approach, as discussed in FEATHER's contribution, a system is specified in terms of its possible events and its reactions to those events. Static

characteristics of the system concerned are captured in the notion of state, and sequences of states (or state-changes) describe the system's dynamic behaviour. Particular aspects within this overall framework are constraints on states and their maintenance during state-changes, inference mechanisms to extract information on prior and future states, as well as demons which cause certain changes of state.

The issue of formal specification is also implicitly addressed by some of the other contributions focussed mainly on calculi. Here too, different formalisms are used, partly due to the specific semantic framework, but also influenced by the class of problems considered. PETTOROSSI/PROIETTI's and BOYLE/HARMER's derivations start from functional specifications (ML and pure LISP with data abstraction, respectively), as do those by BIRD/DE MOOR and SWIERSTRA/DE MOOR. SMITH's specifications include set theoretic data types, notations from first-order logic, as well as specifications by pre/post conditions. MORGAN and PETTOROSSI/PROIETTI use logical specifications (in Z and Prolog, respectively). MÖLLER also deals with set-theoretic specifications.

1.2 Calculi

Calculi are the main focus of WG 2.1's current activities. Therefore, various calculi and their theoretical foundations are comprehensively dealt with in this book. Common to all these calculi is the intent to provide a kind of *mathematics of program construction.* Differences arise mainly in their underlying theoretical basis and, consequently, in the concrete rules of the particular calculus.

As to the theoretical foundations, in all approaches theories are imported from mathematics. Predicate logic and implicational reasoning form the basis of MORGAN's refinement calculus. PEPPER's calculus also uses first-order logic, but in the form of Gentzen style rules of natural deduction. SMITH's approach to algorithm design is essentially based on the idea of translating one theory into another. Set theory with relational theory and formal language theory as subtopics is the basis of MÖLLER's calculus. Relational theories are also used in the calculi of BACKHOUSE/HOOGENDIJK and BIRD/DE MOOR. In addition, they adapt notions from category theory, which is a source of ideas for SINTZOFF's general development calculus as well.

The practically important part of each calculus lies in the formulation and use of its associated rules and strategies. The rules and strategies discussed by PEPPER, PETTOROSSI/PROIETTI and SWIERSTRA/DE MOOR are general in the sense that they are largely language-independent and applicable to a variety of problems – although particular languages and problem domains are used in their presentation. BOYLE/HARMER's rules also are applicable to a variety of problems, but they are mainly syntactical and, as such, language-specific. In contrast to this, BIRD/DE MOOR, MÖLLER and SMITH present problem-oriented rules, i.e., language-independent rules targeted towards particular classes of problems.

1.3 Systems

There are many experimental systems which support formal program development. Likewise, a number of aspects amenable to automation are addressed by such systems. In the context of this book, emphasis is given to those aspects related to the central activity,

viz. calculation of programs; the relevant literature should be consulted for information on support for other important aspects of software development. From the wide range of possible approaches, two extreme positions w.r.t. the way programs are calculated are represented here. BOYLE/HARMER's system is essentially driven by syntactic rewriting, according to built-in strategies and without user interaction. In SMITH's system the user is offered a choice between various built-in strategies for algorithm design, from which the system then automatically derives the requested program using its knowledge-base and inference mechanism.

2 Additional Interests of WG 2.1

In addition to the topics explicitly addressed by the contributions in this book, there are other areas represented within WG 2.1 but only marginally reflected here. Nevertheless, it seems worthwhile to at least mention them, in order to give an impression of the breadth of the group's interests. Moreover, many of these latter topics are subjects of ongoing research, building on the results summarized in this book, rather than being state of the art.

In addition to the semantic issues explicitly addressed in the context of calculi, theoretical aspects of languages such as type theory, non-determinism, parallelism, concurrency and distributed systems are discussed within WG 2.1, as are many more pragmatic issues of language design.

With respect to methodology, reuse and adaptation of designs or developments and transformation towards parallel execution are fairly recent research topics. On the borderline between theory and methodology, the incorporation of efficiency considerations and the formalization of reasoning must also be mentioned.

The issue of system support is considered within the Working Group in a much wider sense, covering nearly all aspects related to automating the production of software. Such interests include optimizing compilers, integrated development environments, end-user interfaces, all kinds of language-oriented tools as well as their underlying databases or knowledge-bases.

Although this book gives a fairly comprehensive account of the state of the art in formal program development, not all subjects could be treated in depth (owing to its mainly tutorial objectives). However, further information may be obtained by pursuing the references provided here. In particular, the reader is referred to reports of ongoing research in this area which are contained in the proceedings of the IFIP Working Conferences organized by WG 2.1.

3 Summaries of the Contributions in this Book

BACKHOUSE/HOOGENDIJK's paper introduces an *algebra of data types* oriented towards the calculation of *polymorphic functions and relations.* Their approach is similar to theories of types in a functional setting, but differs in including *non-determinism.* Moreover, it achieves a uniform treatment of data and control structures. The major goal of the paper is to construct a framework in which a large class of type manipulation problems can be reduced to straightforward calculation. Economical notation and elegant programming

laws are used to express powerful fundamental concepts. Particular emphasis is laid on comparing and contrasting the calculus with the *Bird-Meertens formalism* as used in the contribution by SWIERSTRA/DE MOOR.

BIRD/DE MOOR introduce a calculus based on a categorical setting and involving relational concepts and the theory of free inductive data types. They state and prove a general theorem about problems treatable by *greedy algorithms*. The use of the calculus and the theorem are demonstrated with the *minimum lateness problem*, a job-scheduling application. BIRD/DE MOOR view greedy algorithms as refinements of *dynamic programming*. This latter paradigm is applicable if the principle of optimality applies (i.e., an optimal solution to a problem can be composed of optimal solutions to subproblems if a certain monotonicity condition holds). Whereas dynamic programming decomposes the input in all possible ways, a greedy algorithm considers only one (usually unbalanced) decomposition and reduces the input in each step as much as possible. BIRD/DE MOOR's work has an obvious relationship with SMITH's studies on divide and conquer algorithms.

BOYLE/HARMER's paper deals with the use of automated program transformations as realized in the *TAMPR system*. The purpose of the TAMPR transformations discussed in this paper is the derivation of efficient *vectorizable programs* from functional specifications in pure LISP enhanced by a data abstraction mechanism. TAMPR is built on Chomsky's idea of a transformational grammar; the laws of the underlying program algebra are expressed as rewrite rules. TAMPR also tackles some general problems that are inherent to functional languages, e.g., the speed and storage costs of higher-order functions and the lack of selective updating. It has been used to derive a large number of realistic programs, several of which are in everyday use. The particular application area considered in BOYLE/HARMER's contribution is *numerical solutions* to a practical fluid dynamics problem, stated in terms of hyperbolic partial differential equations. The results obtained are remarkable. For instance, the derived program given in the paper runs faster on a CRAY X-MP vector supercomputer than its hand-coded counterpart.

FEATHER gives a guided tour of the language *Gist*, a representative of formalisms for behaviour-oriented specification. This kind of specification is based on the notions of state and state-changes, which is motivated by the observation that it is difficult to express *ongoing behaviours* in a functional setting. Such approaches address the need to describe the *interactions* of a system with its environment, and to express complex behavioural requirements. Apart from considering the advantages of constructing a formal specification in general, FEATHER deals mainly with constructs appropriate for describing such properties. Among others, the following specification constructs are identified as useful for these purposes: sequences of states (to denote behaviours), access to information from prior and future states, *nondeterminism*, *constraints* and *demons*. In addition, operations on such specifications are discussed. Such operations are construction and maintenance (modification and re-use) of the specification, presentation and analysis (paraphrasing, symbolic evaluation, prototyping) and the transition to an implementation. The discussions are illustrated by a package router (i.e., a mechanism to sort postal packages into one of several bins according to their destinations), and an elevator system (for bringing passengers to their destinations in a multi-story building). The paper is rounded off by a brief comparison with related approaches.

MÖLLER introduces some operators and laws of an algebra of *formal languages*, a subalgebra of which corresponds to the algebra of *multiary relations*. Central operations

are join and composition, in particular in their iterated forms where they describe sets of paths and reachability. The common algebraic structure of these iterations is that of a *Kleene algebra*, which allows the statement of a powerful uniform induction principle. Using this structure, a number of lemmas on paths in subgraphs are proved in a very concise way. The algebra is then used in the formal specification and derivation of some *graph and pointer algorithms*. The examples treated are cycle detection and reachability in graphs, in-situ concatenation and reversal of singly-linked lists, a copying algorithm for general pointer structures and parts of a garbage collection problem.

MORGAN gives an introduction to his refinement calculus, a method of deriving *imperative programs* and presenting their developments. While based on Dijkstra's calculus of weakest preconditions, MORGAN's approach has a number of novel characteristics. In contrast with Dijkstra's original approach, weakest preconditions do not appear explicitly in program developments. Moreover, specifications (which may contain predicates) and programs are not distinguished semantically, and may therefore be mixed. For the derivations, use is made of an extensible collection of *refinement laws* derived from a basic refinement relation between programs. Particular emphasis is laid on *literate programming* when constructing program developments using the refinement calculus starting from specifications in Z. In addition to simple examples (e.g., swap of two variables), a complete derivation of a square root program is presented.

PARTSCH introduces the theoretical foundations and the major concepts of an algebraically based formalism for *problem specification* (using essentially the one developed within the CIP project as a representative). This approach is based on the notion of an *algebraic type*, which defines objects, object classes, and operations on these object classes by means of algebraic axioms. Particular emphasis is laid on the use of such a formalism for the specification of concrete problems, including the *methodological aspects* of formalization. Many standard examples (sets, sequences, bags, maps) are covered. In addition, more comprehensive examples are treated such as various formalizations of finite directed graphs (mainly to illustrate the formalization process), a bounded buffer (as part of a simple communication system) and the *cube problem* (a not too sophisticated puzzle that shows many pitfalls of formalization).

PEPPER's contribution models the programming activity as a deduction in a formal *calculus for program development* based on concepts from algebra and logic. In this framework, programming is viewed as a process that successively extends the program under consideration by adding new axioms or theorems to it. In this setting, axioms constitute design decisions, whereas theorems make deducible knowledge explicit. Technically, the power of the approach is achieved by combining concepts from two related areas: algebraic specifications are used to represent programs, and *Gentzen style rules* of natural deduction are used to represent derivation processes. The paper presents the *algebraic framework* and a collection of characteristic derivation rules illustrated by various examples such as binary logarithm, fast integer division and majority voting.

PETTOROSSI/PROIETTI give an overview of traditional transformational methods, focusing on the *rules and strategies* approach (as opposed to the schematic or dictionary approach). Their contribution deals with the transformation of *functional programs*, formulated in a variant of ML, and using basic rules such as definition, unfold, fold, together with laws of the underlying data algebra. Various strategies are surveyed, such as elimination of composition (to avoid intermediate data structures), tupling (to avoid repeated

visits of data structures, thus enabling on-the-fly garbage collection), generalization and a particular strategy for *online programs*. The topic of transforming *logic programs* is also covered. In this context, strategies similar to the functional case are given. The treatment is illustrated by a large collection of examples, such as evensum, Towers of Hanoi, factorial, collecting tree leaves, Fibonacci numbers, palindrome recognition, Hilbert curves, common sublists, minimal leaf replacement and prime numbers.

SINTZOFF's contribution on typing endomorphisms proposes the kernel of a *meta-calculus* for formal program derivations. This kernel is described in terms of an intuitive semantics and of corresponding algebraic concepts inspired by the theory of cartesian closed categories. It defines operations for composing deductions in such a way that typing is defined as an endomorphism on deductions. The approach is compared with others based on typed λ-calculi and is related to the language DEVA.

SMITH's contribution is composed of two parts, one on the *automation of program construction* (as implemented in the KIDS system) and another one on a *general theory of algorithm design*. The *KIDS system* provides knowledge-based support for the derivation of correct and efficient programs from specifications. The specification language includes set theoretic data types, notations from first-order logic and extensions that support specifications by pre/post conditions. The KIDS system has components for performing algorithm design, deductive inference, program simplification, partial evaluation, finite differencing optimizations, data type refinement and case analysis. All of the KIDS operations are automatic except the algorithm design tactics, which at present require some user interaction. The use of KIDS is traced in deriving a scheduling algorithm as a representative for the many programs that have been derived using that environment. This derivation illustrates various aspects such as design, deductive inference, simplification, finite differencing, partial evaluation, data type refinement and other techniques. The second part discusses the theory of algorithm design used in KIDS. Important concepts are problem theories, algorithm theories, program schemes as parameterized theories, design as interpretation between theories (theory morphisms), algorithm design tactics and refinement hierarchies of algorithm theories.

SWIERSTRA/DE MOOR demonstrate a number of techniques that may be used in calculating algorithms for sequence-oriented problems, using the *Bird-Meertens formalism*. Their central theme is the use of *virtual data structures*, which allow optimization by reasoning at the level of function compositions and then eliminating intermediate data structures at the final transformation step. These ideas are illustrated by two segment problems on lists, viz. the maximum segment sum and the length of a longest low segment.

Elements of a Relational Theory of Datatypes

Roland Backhouse and Paul Hoogendijk

Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

Abstract

The "Boom hierarchy" is a hierarchy of types that begins at the level of trees and includes lists, bags and sets. This hierarchy forms the basis for the calculus of total functions developed by Bird and Meertens, and which has become known as the "Bird-Meertens formalism".

This paper describes a hierarchy of types that logically precedes the Boom hierarchy. We show how the basic operators of the Bird-Meertens formalism (map, reduce and filter) can be introduced in a logical sequence by beginning with a very simple structure and successively refining that structure.

The context of this work is a relational theory of datatypes, rather than a calculus of total functions. Elements of the theory necessary to the later discussion are summarised at the beginning of the paper.

1 Introduction

This paper reports on an experiment into the design of a programming algebra. The algebra is an algebra of datatypes oriented towards the calculation of polymorphic functions and relations. Its design draws most inspiration from earlier research into theories of type in a functional setting but differs from those theories in including an element of indeterminacy. The selection of results chosen for presentation here has been made on the basis of level of correlation with the work of other members of IFIP Working Group 2.1. Other published selections from the work of the research team can be found in references [18, 19, 24, 26].

The goal of our work is to reduce a large class of type-manipulation problems to straightforward calculation. The hope is that within the next century it will become feasible to pose a large variety of such problems in school-leaving examinations alongside problems in, say, the differential calculus (with the implication that they are at the same level of difficulty). In order to achieve this goal it is vital to design a programming algebra in which the combination of economical notation with elegant programming laws is used to express powerful, fundamental concepts.

Fluidity of calculation is considerably enhanced by attention to two design considerations. The first is that the operators in one's algebra should be *total functions*: their use should not be hedged with conditions on the type of their arguments, however simple

those conditions may be. The second is that calculational rules should involve a minimum of bound variables (at most four being our yardstick) and no complicated nestings of universal and/or existential quantifications.

The axiomatic form of the calculus of relations developed by De Morgan, Peirce, Schröder, Tarski and others has both these attributes par excellence as well as offering mechanisms for modelling the indeterminacy that is pervasive in programming problems. It has been chosen for these reasons as the basis for our experiment.

The contribution made in [5, 4, 3] is to extend the calculus of relations with the so-called "polynomial relators". That is, axioms are added defining a unit type, "junction" and "split" operators, and then it is shown how, via the latter two operators, disjoint sum and cartesian product are defined. Sum and product are so-called "relators" (a corruption of the categorical notion of functor), and, with these as building blocks, new relators can be constructed by composition and by the construction of fixed points.

In line with our design principles the junction and split operators are total functions: this in contrast to most category-theory-inspired theories of type where type restrictions are imposed on the corresponding operators. A consequence is that the laws in our system have a recognisably different character to the laws in other systems. Instead of global type restrictions on the variables in the laws the restrictions appear — where unavoidable — in the laws themselves. One of our experimental objectives has been to explore to what extent this would impede or enhance calculations. Our experience is that this design decision was fortunate. Only occasionally do type restrictions occur in our formulae and these act as a welcome reminder to the user of the calculus, and not as a tiresome detail. In this paper only one such type restriction occurs — in the very last theorem.

The main concern of the current paper is to compare and contrast the calculus to the so-called "Bird-Meertens Formalism". This formalism (to be more precise, our own conception of it) is a calculus of total functions based on a small number of primitives and a hierarchy of types including trees and lists. The theory was set out in an inspiring paper by Meertens [23] and has been further refined and applied in a number of papers by Bird and Meertens [9, 10, 13, 11, 14].

Essentially there are just three primitive operators in the theory — "reduce", "map" and "filter". These operators are defined at each level of a hierarchy of types called the "Boom hierarchy" [1] after H.J. Boom to whom Meertens attributes the concept.

The Boom hierarchy begins at the level of trees and subsequently specialises to lists, (finite) bags and sets. In this report we describe a hierarchy of types that logically precedes the Boom hierarchy and in which all three primitive operators of the Bird-Meertens formalism can be defined. We call the hierarchy a hierarchy of "freebies" because all types within the hierarchy are described by "free" algebras (i.e. algebras free of laws). How the Boom hierarchy itself is captured in the spec calculus is described in a companion paper [18].

Space limitations have dictated the form and content of this paper. The first eight sections prepare the reader for section 9 in which the main contribution of the paper

[1] For the record: Doaitse Swierstra appears to have been responsible for coining the name "Bird-Meertens Formalism" when he cracked a joke comparing "BMF" to "BNF" — Backus-Naur Form — at a workshop in Nijmegen in April, 1988. The name "Boom hierarchy" was suggested to Roland Backhouse by Richard Bird at the same workshop.