Lecture Notes in Computer Science 693
Edited by G. Goos and J. Hartmanis

Advisory Board: W. Brauer D. Gries J. Stoer

Peter E. Lauer (Ed.)

Functional Programming, Concurrency, Simulation and Automated Reasoning

International Lecture Series 1991-1992
McMaster University, Hamilton, Ontario, Canada

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

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
D-76131 Karlsruhe, FRG

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Volume Editor

Peter E. Lauer
Department of Computer Science and Systems, McMaster University
1280 Main Street West, Hamilton, Ontario L8S 4K1, Canada

CR Subject Classification (1991): D.1-3, F.3

ISBN 3-540-56883-2 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-56883-2 Springer-Verlag New York Berlin Heidelberg

Printed in Germany

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

Preface

This collection of papers arose from a series of lectures, given in the Department of Computer Science and Systems, McMaster University, Hamilton, Ontario, Canada, during 1991-92, at the invitation of Peter Lauer. The series was co-ordinated by Peter Lauer, Jeffery Zucker and Ryszard Janicki. The lectures were intended to familiarize workers in Computer Science and other disciplines with some of the most exciting advanced computer based systems for the conceptualization, design, implementation, simulation, and logical analysis of applications in these disciplines. The papers are mostly the work of individuals who were among the originators of the systems presented.

We hope that this volume will make it easier for colleagues at other universities and research establishments to evaluate the utility of these systems for their application areas. We also hope that this volume will be of paramount utility to graduate students in the various disciplines.

The collection of papers presents some strong motivational points for the use of theory based systems in the areas of functional programming, concurrency, simulation, and automated reasoning, highlighting some of their advantages and disadvantages relative to conventional systems.

At the editors invitation, the authors kindly agreed to furnish newly written papers on theory based systems which provide a guide into some of the major operational systems and which might form a useful basis for assessing knowledge and skills required for their informed use.

The four topic areas were selected for various related reasons.

Functional programming rather than procedural programming was chosen because it provides a good level of abstraction from the standpoints of the user, the tractability of full formalization of semantics, and providing good practical implementations, allowing for computer supported experimentation with concepts expressed in this basically declarative style.

Concurrency rather than sequentiality was chosen as basic because we feel that this is closer to real-world systems and human thought processes, and avoids the artificial introduction of sequentiality constraints due to one's sequential model and not due to the nature of the system modelled.

Simulation is used here as a synonym for modelling or prototyping and was chosen since it is a means to enhance understanding of complex situations and dynamically changing systems, and a basis for experimental study of such systems. Furthermore, simulation may be used to validate whether a computer implementation of some real-world situation is adequate for the purpose for which it was designed.

Automated reasoning, which we take to include not only fully automated theorem provers but especially interactive definition debuggers and proof checkers, was chosen because it relieves the user from tedious, time consuming and error prone activities involved in checking whether chains of inferences and logical conclusions about the system are justified. We feel that a similar advantage to that obtained by the presence of syntax checking in compilers, for developing error free programs, can be obtained by the presence of proof checkers for developing error free system models, and ultimately trustworthy computer systems implementing them.

The issue is how to make existing theory based systems more accessible to users of various kinds and at different levels.

Theory based systems have the advantages of precision, trustworthiness, and generality. They can be used effectively to enhance learning. However, they have the disadvantage of relative inefficiency in operation, and a greater learning gap to be closed by the user.

Conventional systems have the advantages of efficiency and a reputedly smaller learning gap, and they can also be used effectively to enhance learning, are more familiar to users, and have extensive application development.

However, familiarity with conventional computer systems may not be as much of an advantage as might appear at first sight. There is a more basic kind of familiarity which users have with theory based systems which is often overlooked and which, if exploited, has a much greater payoff than the exploitation of familiarity with conventional computer concepts. For example, familiarity with high school algebra, which can be relatively safely presupposed in all adults who have graduated from high school, makes for an easy road to computer systems based on the algebraic approach and its concomitant equational style of reasoning. The simple realisation that the objects of the algebra need not just be numbers, but can essentially come from any inductive domain, allows users to transfer the same algebraic understanding from the domain of numbers to domains such as programs, data, machines, and even systems as a whole. The same style of equational reasoning remains valid throughout. This permits frequent transfer of knowledge from one domain to another by a mathematical equivalent of analogical thinking.

Furthermore, conventional computer oriented concepts are rather far removed from human ways of thinking about real world systems, except in the case of the object oriented paradigm, whereas the algebraic approach has many aspects in common with the object oriented approach and hence can make similar claims to real world closeness. On the whole, theory based systems could be considered closer to real world situations, since they are descriptive and try to introduce the least amount of model specific formalism possible, whereas conventional systems force upon the user all the details of computer oriented models, including particularly the need to express concepts algorithmically and usually sequentially.

So it seems far from obvious that conventional systems are closer to real world situations, and hence to the non-computer specialist user, than theory based systems.

Even if the gaps for both were the same, there would still be the greater payoff from investing time in learning to understand and use theory based systems, since one obtains ability for very general knowledge transfer from domain to domain. One only needs to compare the general applicability of the results of one year's study of C++, which is the least amount of time required to become proficient in that complex language, with the general applicability of the results of one year's study of general algebraic topics.

Mind-set for this Series of Lectures and Papers.

At the outset of the lecture series, I formulated some general thoughts about the current intellectual environment of advanced system theory as it relates to computer science. Authors of papers were aware of this mind-set and have taken it into account in orienting their papers for inclusion in this volume. Since this original mind-set may be of interest to the general readership it is included here.

1. **Convergence of theoretical computer science and mathematics.** Formal and theoretical systems developed in computer science and mathematics are increasingly converging, as are the interests of researchers in both areas. This is witnessed by the regular occurrence of such conferences as the Annual IEEE Symposium on Logic in Computer Science, the International Workshops on Mathematical Foundations of Programming Semantics, and the special section on Logic, Mathematics and Computer Science of the International Congress of Logic, Methodology and Philosophy of Science. In addition four new journals have appeared in the past year, *Mathematical Structures in Computer Science* (Cambridge University Press), the *Journal of Logic and Computing* (Oxford University Press), the *International Journal of Foundations of Computer Science* (IOS Press), and *Category Theory for Computer Science* (Prentice Hall).
2. **Theory based environments are transforming system development.** Practical computer based realizations of such theory based systems are rapidly appearing, and promise radically to transform the entire process of software development, from conceptualization to implementation, permitting rigorous formulation and verification of most aspects of the process (see the paper by Peter Lauer in this volume).
3. **Environments must be efficient and semantically sound.** A practical environment for the rigorous development of software must be based on an efficiently executable programming notation which enjoys as clear and sound a semantics as the more abstract, and usually more mathematical and often non-executable notations used to express requirements, specifications, designs, etc.
4. **Functional languages best achieve efficiency and semantic clarity.** To date, functional programming languages (see the papers by David MacQueen on SML, and by R. Frost and S. Karamatos, in this volume) are the most successful in achieving efficiency comparable to the most efficient procedural languages such as C, while at the same time permitting the formulation of a clear mathematical semantics, which sometimes, for example, in the case of OBJ3 (see the paper by Tim Winkler in this volume), coincides with the actual operational (run time) semantics of the language, which is based on the notion of rewriting (see the paper by Nachum Dershowitz, in this volume).
5. **Domains of interest conceived analogously in mathematics and computing.** Mathematicians and logicians tend to characterize domains of interest by giving a structure consisting of some domains, and a number of operations or functions, and possibly relations, on these domains. The meanings of the functions and relations are then stated axiomatically, for instance as equations or inequalities.

   Increasingly, computer scientists tend to characterize executable representations by defining concrete or abstract data types, which essentially correspond to the mathematician's notion of (algebraic) structure, except that the meanings of the functions and relations are defined operationally in terms of language primitives which directly translate to executable machine code.

   This similarity of characterization of domains of interest inspired the proponents of the algebraic specification methods to work towards a new style of software development which would be pervaded by sound mathematical principles and supported by powerful mathematical tools (see the papers by Tim Winkler on

OBJ3, and by Stephen Garland, John Guttag and James Horning on LARCH in this volume).

6. **Type theory fits the general needs of domain independent systems.** Adequate support for reasoning about arbitrary formal systems (including executable notation) requires more than the ability to express domain specific information. It requires powerful logical systems in which to formulate, develop, analyze and compare different domain specific formalisms. Modern type theory has proved to be extremely fruitful when applied for this purpose. In fact, solutions of problems in computer science using type theoretical notions have greatly stimulated research into the lambda calculus and type theory by mathematicians and logicians, and have even contributed new developments in these areas. (See the papers by Douglas Howe on Nuprl, by K. van Hee, P. Rambags and P. Verkoulen on ExSpect, and by Sentot Kromodimoeljo, Bill Pase, Mark Saaltink, Dan Craigen and Irwin Meisels on EVES, in this volume.)
7. **Other disciplines have need of advanced theory based systems.** There exist a number of very interesting prototypical computer based systems which support rigorous and systematic development of executable software from specifications (e.g., OBJ3, LARCH, ExSpect, EVES, and IDEF/CPN, which are all presented in papers in this volume). Graduate students in computer science and other disciplines such as engineering, business, linguistics, philosophy, etc., need to gain experience with such systems so that they can usefully employ them in the process of producing dependable (verified) application oriented software. But this presupposes, especially in the case of non-mathematicians, that much of the theoretical underpinning of the system is hidden from the user, and that the remaining theory is taught in an appropriate manner and at the right time.
8. **SML is widely used to implement such advanced systems.** Standard ML and its extensions are proving to be the functional programming languages of preference for implementing many of the most advanced systems of the kind we have been discussing (see the papers by David MacQueen and John Ophel on SML, in this volume). This is in part due to the fact that they can be made to produce quite efficient code, while at the same time having a very well defined mathematical semantics. In fact, most of the systems covered during the series of talks are implemented in SML or in LISP, or use SML as part of their programming interface.
9. **Concurrency gives efficiency and conciseness.** Concurrency is of importance for efficiency reasons, but also due to the fact that decomposition into relatively independent concurrent subsystems often leads to much shorter code and increased clarity. Standard ML has been extended to support concurrency in a number of ways, which ensure that the advantages of functional programming are preserved (see the paper on Concurrent ML by John Reppy, and the general paper by David MacQueen on SML in this volume). On the other hand, concurrency introduces additional complexity into the problem of correctly conceptualizing the possible behaviours of the system and proving the correctness of the algorithms involved. To manage this complexity, the need for rigour and formality in proving properties of the system is even greater than in the case of sequential and centralized systems. Systems such as the Concurrency Workbench (see the paper by Rance Cleaveland, and the preparatory papers by Jeffery

Zucker in this volume), the IDEF/CPN system from Metasoft Corporation (see the paper by Jawahar Malhotra and Robert Shapiro, as well as the paper by Robert Shapiro, Valerio Pinci and Roberto Mameli in this volume), give much support to this endeavor. The effective implementation of concurrent systems is also difficult, as is the effective exploitation of parallel architectures by programmers. Mathematically well founded mechanical schemes for synthesising concurrent programs from programs that do not specify concurrency or communication also promise to reduce the complexity inherent in developing concurrent systems (see the paper by Michael Barnett and Christian Lengauer in this volume).

10. **Graphical representation of knowledge is important.** Graphical representation of knowledge is increasingly recognized as an important technique for visualizing complex relationships. Thus, in mathematics, category theory generalizes the conventional arrow representation of functional relationships, to obtain powerful and general ways of conceptualizing complex (functional) domains, and replacing specific combinatorial arguments by graph manipulation (arrow chasing).
11. **Level of performance of students rises when courses stress theory based approach and use of theory based systems.** Limited experiments with students indicates that use of rigorous specification techniques, particularly following the algebraic approach, enhances the student's ability for independent, verified , and complete program development, and allows for the ready transferral of knowledge from high school elementary algebra to the business of specifying and designing sofware. Using the algebraic approach also reinforces their knowledge of the algebraic techniques they learned in high school.
    This seems to indicate that this approach may well be the best for requirement specification, because it is to be assumed that any potential customer requesting a software system will have completed high school algebra.
    Theory based systems which are based on logic require more training and sophistication than can be expected from high school graduates. But as more programming takes place in languages like Prolog even at the high school level, this may change soon.
12. **Theory based systems should be human and problem oriented.**
    (a) In computer science, graphical representations have extended application in software engineering environments and particularly in the representation of concurrent systems. The IDEF/CPN system from MetaSoftware Corporation (see the paper by Robert Shapiro, Valerio Pinci and Roberto Mameli in this volume), is one of the most developed, integrated , and theoretically sound systems elegantly supporting graphical interaction.
    In IDEF/CPN it is possible to input an (inscribed) graph from which the system automatically generates a correct program. The ExSpect system is a similar system which at present has more system analysis support than IDEF/CPN (see the paper by K.van Hee, P. Rambags and P. Verkoulen in this volume).
    (b) In Nuprl (see the paper by Douglas Howe in this volume) it is possible to input a proof (a reasoned logical specification) from which the system automatically extracts a correct program.
    (c) Pattern matching is a natural human activity and the use of pattern match-

ing in explaining the application of functions to their arguments in SML, OBJ, and W/AGE enhances readability greatly (see the papers by David MacQueen and John Ophel on SML, by Tim Winkler on OBJ3, and by R. Frost and S. Karamatos on W/AGE, in this volume).

(d) The dictum that system code be as high-level as possible and the same throughout, which is one of the aspects of parametric programming as introduced by J. Goguen, leads to ease of comprehension of the whole system in the case of all of the systems described in this volume.

Intended readers of this volume

This volume is meant as a modest contribution to narrowing the learning gap facing conventional computer users when they wish to use advanced theory based systems. The papers in this volume are meant for a wide audience and should not require great mathematical sophistication for their comprehension, in fact a high school knowledge of algebra, and perhaps a little set theory and formal logic should suffice. The papers contain numerous references for those wishing to pursue any of these topics to greater depth. These references may require more mathematical accumen from the reader, but the appropriate utilization of the available computer implementations of the mathematical theories, during the learning stages, should enhance the process of self-instruction required to acquire the necessary mathematical knowledge and skills for an informed use of these systems.

The collection of papers could also be used in advanced courses by students and researchers as an introduction and guide to advanced theory based systems, all of which are operational at McMaster and are readily available to other educational and research institutions.

Acknowledgements

Financial support for the series was given by the Department of Computer Science and Systems at McMaster University, supplemented with some support from Dr. H. A. Elmaraghy at the Flexible Manufacturing Research and Development Centre and the Department of Mechanical Engineering at McMaster, and Dr. W. Elmaraghy at the Design Automation and Manufacturing Research Laboratory and the Faculty of Engineering at the University of Western Ontario.

It is due to Ryszard Janicki's prompting that Peter Lauer undertook to produce this volume of papers which gives these lectures a more permanent form of use to a much wider audience.

Thanks are due to Alfred Hofmann and Hans Wössner, both of Springer-Verlag, for their continuing support and excellent advice during the production of this volume.

Last, but most important, we would like to thank the authors of the papers for taking the time in their busy schedules to produce such excellent papers in such a short time.

March 1993

Peter E. Lauer (Editor)
McMaster University

Contents