MATHEMATISCHES FORSCHUNGSINSTITUT OBERWOLFACH

Tagungsbericht 1/1982

Formale Methoden und Mathematische Hilfsmittel für die Softwarekonstruktion

4.1. bis 8.1.1982

Die diesjährige Tagung über "Formale Methoden und Mathematische Hilfsmittel für die Softwarekonstruktion" stand unter der Leitung von H. Langmaack (Kiel), E. J. Neuhold (Stuttgart) und M. Paul (München).

Neben Fragen der Sprachübersetzung, Datenbanksystemen und abstrakten Datentypen standen vor allem Probleme der Programmverifikation und Definition kommunizierender Systeme im Mittelpunkt des Interesses. Im Bereich der axiomatischen Semantik dominierten Fragen der (relativen) Vollständigkeit und der Erweiterung der Verifikationsregeln auf Prozeduren und kommunizierende Prozesse. Bei den konkurrierenden Programmsystemen konzentrierten sich die Arbeiten außerdem auf die formale Definition und Beschreibung. Insgesamt läßt sich eine Aussicht auf eine Konvergenz gewisser verschiedenartiger Begriffsbildungen und Auffassungen erkennen. Weitere Tagungen dieser Art können im Sinne dieses wünschenswerten Prozesses nur begrüßt werden.

Gefördert durch
DFG Deutsche Forschungsgemeinschaft

Teilnehmer

K. R. Apt, Paris
D. Björner, Lyngby (Dänemark)
C. Böhm, Rom
W. Brauer, Hamburg
M. Broy, München
E. M. Clarke, Cambridge, MA.
V. Claus, Dortmund
A. B. Cremers, Dortmund
O. J. Dahl, Oslo
W. Damm, Aachen
J. B. Dennis, Cambridge, MA.
P. Deussen, Karlsruhe
J. Eickel, München
G. Goos, Karlsruhe
D. Harel, Rehovat (Israel)
C. A. R. Hoare, Oxford
G. Hotz, Saarbrücken
K. Indermark, Aachen
N. D. Jones, Aarhus
P. Kandzia, Kiel
M. Karpinski, Edinburgh
U. Kastens, Karlsruhe
I. O. Kerner, Dresden
F. Kröger, München
H. Langmaack, Kiel
P. Lauer, Newcastle-upon-Tyne
J. Loeckx, Saarbrücken
O. Mayer, Kaiserslautern
A. Meyer, Cambridge, MA.
B. Möller, München
E. J. Neuhold, Stuttgart
M. Nivat, Paris
E.-R. Olderog, Kiel & Oxford
M. Paul, München
G. Plotkin, Edinburgh
J. E. Stoy, Cambridge, MA.
S. Takasu, Kyoto
B. Trakhtenbrot, Tel-Aviv
J. V. Tucker, Leeds
H. K. G. Walter, Darmstadt
M. Wirsing, München

Vortragsauszüge

K. R. Apt: Fair termination revisited

(Work done jointly with A. Pnueli and J. Stavi)

A proof method for establishing the total correctness of nondeterministic programs under the assumption of fairness is presented. The method makes use of auxiliary delay variables which count down to the instant in which a certain action will be scheduled. It can be applied to programs allowing nested do-loops in contrast to the previously suggested methods which only allowed non-nested loops.

D. Bjørner: Software architectures & programming systems design on the didactics of a methodology

The structure of a Software Development Methodology was outlined and argued in terms of the corresponding composition of a larger (set of) textbook(s) which the speaker is currently writing: following introduction parts on denotational semantics based on "software abstraction principles" are parts on the "formal description of programming concepts", "implementation techniques" and the application of the above implied techniques to "programming language linguistics and the development of the interpreter- and compiler processors", "data base models and their data base management systems", "parallel-, or process-oriented systems and their monitors", and "applications software". The main thrusts are: (1) that mathematical semantics definitions be the way in which software architectures be specified and the basis from which implementations be "derived", and w.r.t which correctness be argued; and (2) that such techniques are uniformly applicable across systems- and application programming areas, thereby pre-empting much of the material hitherto classified as exclusively or particularly belonging to specific such areas.

C. Böhm: The purpose of unit-lists in functional programming

A new set of primitives for Combinatory Logic and Algebras is suggested, which enables us to construct an algebraic notion of finite linear sequences of combinators. It turns out that the notion of unit-lists is powerful enough, together with the primitives, to express (with almost no change) all the basic concepts of Functional Programming. In particular, this approach enables us to eliminate the "condition" and the "apply-to-all" constructs.

W. Brauer: <u>On modular specification and implementation of abstract data types</u>

(Joint work with O. Schoett, Hamburg/Edinburgh)

Starting from considerations of practical programming needs and habits some proposals are made concerning the construction of programs by decomposition into modules and their separate implementation. A formalization of the notion of module is given, the idea is to distinguish between the surrounding of a module (the given sorts and functions) and the (new) sorts and functions it defines such that semantically a module is a function associating to any interpretation of the surrounding, an interpretation of the new sorts and functions. A corresponding notion of specification using first order logic is given. A rather general notion of implementation is defined: an implementation must only have the same observable behaviour as a model of the specification. All these notions are formulated within the framework of partial algebras; the interpretation of term equations with terms possibly having no value is defined according to Russels theory of descriptions - it coincides with P. Burmeister's treatment of E-equations (FCT' 81, LNCS 117). Two homomorphism criteria are given to show that a (separate) implementation is correct and that a system of separate implementations of the components of a decomposition of a specification is an implementation of the whole specification. Moreover, it can be shown that a system of concrete, programmed modules satisfies the homomorphism criterion if each module accesses data values of new sorts of other modules only by access functions declared in these modules.

The diploma thesis by O. Schoett on which this fact is based appeared as a technical report of the Department of Informatics, Hamburg University.

M. Broy: Variations on fixed point theory for nondeterminism and concurrency

For an applicative programming language that includes recursive definitions of nondeterministic functions and systems of expressions communicating by streams a fixpoint-based mathematical semantics is given. The language comprises straightforward nondeterminstic choice and $\perp$-avoiding ambiguity operator. It allows to represent networks of stream processing, nondeterministic functions. To overcome the problems of the powerdomain approach for giving a correct meaning for this language several partial orderings are used in combination for characterizing the intended fixed point leading to a fully abstract mathematical semantics.

E. M. Clarke: Effective axiomatizations of Hoare logics

For a wide class of programming languages P and expressive interpretations I, we show that there exist sound and relatively complete Hoare logics for both partial correctness and termination assertions. In fact, under mild assumptions on P and I we show that the assertions true in I are uniformly decidable in the theory of I(TH(I)) iff the halting problem for P is decidable for finite interpretations. Moreover, the set of true termination assertions is uniformly r.e. in TH(I) even if the halting problem for P is not decidable for finite interpretations. Since total correctness assertions coincide with termination assertions for deterministic programming languages, this last result unexpectedly suggests that good axiom systems for total correctness may exist for a wider spectrum of languages than is the case for partial correctness.

V. Claus: Mathematical methods for decompilation

The decompilation deals with the translation from lower level to higher level programming languages, e.g. decompilation from Assembler to PASCAL. There are straightforward methods to embed lower level languages into higher ones, however the main problems are the detection of high level structures in primitive programs and the handling of conflicts.

The decompilation of control structures will be described by a set of transformations acting on the underlying graphs. In the case that the destination language has PASCAL-like control structures the kernel set of these transformations are shown to have the Church-Rosser-property. The problem, how to structure unstructured structures, leads to several results which are important for practical implementations.

Reference: Hecht & Ullman, SIAM JfC, 1972 (Flow Graph Reducibility), B. Baker, J ACM 1977, 98-120 (An algorithm for structuring flowgraphs), several master thesis at the University of Dortmund.

A. B. Cremers: Program prototyping in the data space machine

(Joint work with T. N. Hibbard, Pasadena, CA.)

A data space is a formal model of an abstract machine, and consists of a transition system on which an information structure is imposed. The full class of data spaces, as defined in our previous publications, is clearly beyond any computable syntax. We have identified a subclass of the data spaces, the "context-free" ones, which include all the abstract machines we know about and which are within the reach of a syntax. The "data space machine" is a syntactic embodiment of context-free data spaces. It can be used for program prototyping in the sense of constructing highly conceptual, problem oriented executable models of data spaces. Two realizations of the data space machine have been built which are now being used in nontrivial applications.

O. J. Dahl: Partial corectness semantics of communicating sequential processes

A simple extension of conventinal Hoare logic is defined for CSP programs. The system is based on the idea of introducing communication histories as additional program variables. A slightly more complicated version of the system is shown to be complete as well as consistent.

W. Damm: On veryfying higher type procedures
(Joint work with B. Josko)

Since Clarke's work on incompleteness results for PASCAL it has been conjectured, that the sublanguages of ALGOL 68 with only finite modes and without global variables has a sound and relatively complete Hoare-style proof-system. However, since program-trees of such programs are highly non-regular, it follows from the work of Langmaack and Olderog that new proof-rules are needed to deal with such languages.

The crucial observation to obtain a complete proof systems is that the proof of a procedure call $P(\Gamma_1, \ldots, \Gamma_m)$ can be obtained by proving first correctness formula for the actual parameters and proving the procedure P separately. The new proof rule then guarantees, that a correctness formula for $P(\Gamma_1, \ldots, \Gamma_m)$ can be obtained by substituting the pre-and post-conditions of the actual parameters into the pre- resp. post-condition of P. This demands an extension of the assertion language by allowing higher order predicate variables. The net effect of this proof-rule is, that a proof of a complex call $\{p\}\ P(Q_m)\ldots(Q_0)\ \{q\}$ - with Q_j of functional level j - is eventually reduced to "simple" proofs of correctness formula $\{f\}\ P'\ \{f'\}$, where P' is a procedure identifier and f, f' are such higher-type formula, hence to a regular proof tree. Applying the classical recursion rule gives a finite proof-tree.

P. Deussen: An exercise in verifying programs: partial correctness of recognising and of parsing algorithms

An intuitively given algorithm for recognising (and parsing) languages is verified by using Dijkstra's weakest preconditions and by using Hoare's derivation rules.

In both cases a finite choice operator had to be considered, first by deriving its weakest preconditons equal to that of Dijkstra's if-construct, secondly by presenting a Hoare-rule for this operator.

As a by-product of the verification a necessary and sufficient condition for the partial correctness of the algorithm is obtained, which fact is of importance because from that algorithm all types of parsers can be obtained by suitable refinements.

J. Eickel: Data structured programming

Certain classes of programs (e.g. compilers or editors) allow the global control structure to be generated from a formal description of the data structure by using standard routines. This approach is based on a special case of Jackson's principle and is used in compiler generating systems. It allows to introduce very high level language constructs in special purpose languages and is reflected in Henke's method of function extension in algebraic semantics.

These connections having been pointed out the problem of improving efficiency and removing nondeterminism in programs by merely transforming the data type definitions is discussed. This transformation is independent of particular problems and consists essentially of an efficient Greibach-normal-form-transformation. Finally the relation with a unified approach to parser generating algorithms is shown.

G. Goos: Systematic Code Generation

Code generation is the transformation of a source program expressed as a structure tree into a sequence of abstract machine instructions. The transformation consists of the steps storage mapping, tree transformation and target attribution, code selection. It is shown how storage mapping can be achieved by a straightforward set of algorithms independent of how this step is integrated into the remaining steps. The necessary tree transformations and target attribution depend on the desired degree of optimazition and on the code selection method. It is shown how algebraic laws may be used for optimizations. A special code selection based on the work of Glanville and its embedding into the generation process is discussed. The method has been successfully used in a number of code generators for PASCAL. Results from these experiments are reported.

D. Harel: Decision problems in propositional dynamic logic

The validity problem for formulas of regular propostional dynamic logic (PDL) is decidable. Some recent results extending this fundamental result of Fischer & Ladner are surveyed. In particular, the subject of extensive recent research is the status of the theorem upon enriching the class of regular programs. A new approach to proving decidability is suggested, which might provide new decidable cases, and help explain the seemingly unbehaved borderline between those classes of context-free programs whose addition to PDL ruins its decidability, and those whose addition does not.

C. A. R. Hoare: Specifications, programs, proofs

A specification of a mechanism is a predicate describing all possible observations of its permitted behaviour. A computer program defines a mechanism, of which we enquire "what is the strongest specification that it meets?" We define a simple programming language of communicating sequential processes by identifying each program with its strongest specifiation. Many important properties of programs follow directly from a definition in this style.

N. D. Jones: A simple compiler generator based on algebraic semantics

A simple algebra-based algorithm for compiler generation is described. Its input is a semantic definiton of a programming language, and its output is a "compiling semantics" which maps each program into a sequence of compile-time actions whose net effect on execution is the production of a semantically equivalent target program. The method does not require individual compiler correctness proofs or the construction of specialized target algebras.

Source program execution is assumed to proceed by performing a series of elementary actions on a runtime state. A semantic algebra is introduced to represent and manipulate possible execution sequences. A source semantic definition has two parts: A set of semantic equations mapping source programs into terms of the algebra, and an interpretation which gives concrete definitions of the state and the elementary actions on it.

P. Kandzia: Structural properties of relational data base schemes with functional dependencies

Investigating normalization processes and the use of null values in relational data base schemes with functional dependencies one needs to know properties of the set of all functional dependencies holding in the considered scheme. Usually this set is called the closure CL(α, F), where α and F are the given attribute set and the given set of functional dependencies, respectively. It is shown in the lecture that the closure has some kind of coset structure. The cosets themselves can be characterized very simply by one maximal and some minimal elements. The results can be employed as a common framework for algorithms essential in logical data base design. Especially for the problem of Boyce-Codd normal form (BCNF) classes of data base schemes are given for which BCNF can be found in acceptable time.

U. Kastens: <u>Code generation based on operator identification</u>

The code generation phase of a compiler maps operations of the source language into instruction sequences of the target language. In general the desired effect can be achieved by several instruction sequences which differ in costs (e.g. code length) and used recources (e.g. registers). The code selection depends on the context of the operation.

We present a method for code selection for tree structured source programs. It is based on the principle of overloading resolution for operators applied in the analysis of high level languages (e.g. ALGOL68, Ada). The target machine instructions are considered as typed operations: Properties of operands and results are described by types in terms of the target machine (e.g. "register", "address"). By that means code selection and resource allocation can be formally specified. It will be shown that completeness and optimizing properties of such a specification are decidable.

I. O. Kerner: Some questions concerning problem-specifications and their transformation into programs

Most of the errors in software construction arise from incomplete or even wrong problem specifications. There are examples for the transformation of specifications into programs. But what is the impact of different (correct) specifications to program efficiency? But what happens if those transformations are applied on unsolvable (undecidable) problems?

F. Kröger: Process abstractions

The talk describes an example: Starting from two parallel programs π_1 and π_2, both working in a producer-consumer-fashion, we try to formulate an "abstract" version of this process scheme by first introducing abstract data types instead of the concrete one, and secondly abstracting the processes and their different synchronization techniques themselves. We apply the result to verification. It is shown that several properties (e.g. deadlock-freedom) of π_1 and π_2 can be proved on the abstract level, and other properties (e.g. partial correctness) can be proved by only "implementing" some few of the abstract notions.

P. E. Lauer: Modelling concurrent systems without globality

The problem is the modelling of the behaviour of collections of concurrent and interacting systems without any globality assumptions, such as a single clock, a global state or even a single observer. We discussed this problem within the COSY formalism. The semantics of a COSY specification has been defined (i) by an interleaving semantics using totally ordered histories and projections, (ii) translation to Petri-nets, and

(iii) vectors of histories representing partially ordered histories. Although the models (i) and (iii) are isomorphic they are not equivalent since two specifications S and S' which are equal in semantics (i) are not necessarily equal in (iii). In fact, semantics (iii) distinguish between specifications which (a) are subdivided into a different number of subsystems and (b) have events differently distributed to subsystems. Semantics (i) does not distinguish systems with regard to their degree of concurrency and/or distribution as semantics (iii) does.

J. Loeckx: A new specification method for abstract data types

The algorithmic specification method for abstract data types has been introduced in (Lo81). The present talk presents an overview of the specification method while commenting the main ideas, however, particular attention is given to the error treatment, the proof techniques, the specification of parameterized data types and the notion of implementation. The advantages over the algebraic specification method are indicated and the price paid for these advantages is discussed. Finally, plans for future work are discussed.

(Lo81) J. Loeckx: "Algorithmic specifications for abstract data types", Proc, 8th ICALP (Acre, Israel), LNCS 115 (July 1981) 129-147

A. Meyer: Termination assertions about recursive programs: Completeness and axiomatic definability

(Joint work with J. Mitchell)

The termination assertion p<S>q means that whenever the formula p is true, there is an execution of the possibly nondeterministic program S which terminates in a state in which q is true. Program S may contain local variable and recursive procedure declarations with call-by-value and call-by-address parameters. Formulas p and q are first order extended with a construct for expressing hypotheses about calls to undeclared procedures in S. There is a natural, effective, complete axiomatization of the termination assertions valid in all interpretations. Termination assertions also suffice to define the semantics of recursive programs in the following sense: two programs have the same termination theory iff they are semantically equivalent.

B. Möller: Transformational semantics for pointers and updatable storage

A small applicative language is introduced which allows to define objects with sharing and circularities. Essentially, recursion equations for objects are used as objects themselves. In order to get meaningful solutions for these equations, non-strict constructor functions have to be used. Therefore the language requires lazy evaluation; an operational semantics using this technique is given. Several examples illustrate how familiar structures with pointers can be modelled and reasoned about in this language. Secondly, a language level more oriented towards the von Neumann machine is added which allows variables and selective updating. This level is connected to the applicative one by definitional transformation rules. They are given in such a way that the notions of assignment and selective updating are disentangled and the problems of the use of references in ALGOL 68 are avoided. The applications of the techniques to the derivation of correct procedural implementations of abstract data types are briefly outlined.

E. J. Neuhold: Building data base management systems through formal specification

The Vienna Development Method has been used to develop the formal specification of a relational data base system. We illustrate how this specification can form the basis for systematically constructing an implementation of the system. The approach follows the three-level architecture concept proposal by the ANSE-SPARC committee and specifies the necessary scheme definition, data manipulation and mapping languages. Through stepwise refinement of the abstract data type oriented specification an implementation is produced. The formal techniques applied allow both the verification of the original specification but also the proof that the derived implementation behaves correctly.

References: E. J. Neuhold, Th. Olnhoff: The Vienna Development Method and its use for the specification of relational data base systems.
E. J. Neuhold, Th. Olnhoff: Building data base management systems through formal specification.

M. Nivat: Behaviour of processes

A process is a tuple of languages over an alphabet of actions A $p = \langle L^{init}, L^{fin}, L^{inf} \rangle$ where $L^{init} \subseteq A^*$ is the set of initial behaviour, L^{fin} the set of terminated finite behaviour and $L^{inf} \subseteq A^{\omega}$ is the set of infinite behaviour. These 3 sets always satisfy $L^{init} = FG(L^{init})$, $L^{fin} \subseteq L^{init}$ and $L^{inf} \subseteq Adh(L^{init})$.

One defines several natural notions: deadlock-freeness, safety, ideality, closedness, normality, and centrality. They are properties of triples of words which express properties of actual processes.

One tries then to realize a process p given by L^{init}, L^{fin}, L^{inf} using a transition system or nondetermistic automaton: such as transition system $S = \langle C, A, T, D, C_{fin}, C_{inf} \rangle$ naturally defines 3 languages $L^{init}(S)$, $L^{fin}(S)$, and $L^{inf}(S)$ and p is said to be realized by S iff $p = p(S)$.

These notions are extended to vectors of processes such as $\vec{p} = \langle p_1, \ldots, p_k \rangle$ which run simultaneously in accordance with some synchronization condition Syn, a general form of which is $\mathrm{Syn} = \{\vec{\alpha} \in S^{\infty} \mid \varphi(\vec{\alpha}) \in p_0\}$ where $S \subseteq A_1 \times \ldots \times A_k$. $\varphi : S \to A_0$ and p_0 is a process called the "synchronization mechanism". The general problem of building a multitransition system S which realizes the system $P = \langle \vec{p}, \mathrm{Syn} \rangle$ of synchronized processes is raised, unfortunately not solved: a practical question is to choose the realizations of $p_0, \ldots, p_k$ so that S be the simplest possible. Even in the rational case where all the p_i's are realizable by finite transition systems we are far from a satisfactory solution (though there exists an effective construction of S).

E.-R. Olderog: Correctness of PASCAL-like Programs without global variables

PASCAL-like programs are defined to be blockstructured programs with procedures of mode depth ≤ 2. Due to Clarke 79 there cannot be any sound and relatively complete Hoare-like system proving partial correctness for the full set of these programs. However, in Langmaack & Olderog '80 it has been conjectured that such a system exists once global variables are disallowed.

In this talk a slightly weaker version of this conjecture is proved by presenting a Hoare-like system which is sound and g-complete for PASCAL-like programs without global variables. g-completeness means completeness modulo a special second order theory and an appropriate notion of expressibility. The proof system provides new methods of dealing with procedures, namely the use of relation variables and the so-called separation principle. The completeness proof for the system is carried out in a transparent way with help of modified computation trees.

G. PLotkin: Fairness and countable nondeterminism

The problem faced is how to extend denotational semantics, as developed by Scott, Strachey and others, to deal with concurrent programming languages when the fairness (= finite delay) property is assumed. The difficulty is that natural attempts lead to the failure of the continuity that lies at the heart of the usual theory. Apt and I considered previously the case of random assignment and showed how a weak continuity property (preservation of ω_1-lubs) provides a substitute for the normal continuity provided the domain involved had lubs of both ω_0 and ω_1 sequences. The point here is that fairness leads to countable nondeterminism and random assignment is an easy way to introduce the latter.

S. Takasu: An interactive program synthesizer

An interactive program synthesizer is described. The system is an interactive proof-checker which constructs a Pascal program as its background job if we use the system to prove a quantified specification of the program.

B. Trakhtenbrot: On denotational semantics and partial correctness for languages with procedures as parameters and with aliasing

Semantics and partial-correctness theory for programming languages above were developed up to now mainly in frame of the operational approach to semantics. As an alternative a pure denotational approach is suggested, that is based on suitable compilation of program texts into terms of a λ-language with the construct "let p = body in T". Equivalence transformations of terms in this λ-language use conventional λ-arguments and fixpoint techniques. They induce equivalence transformations of programs, that are included in the proof system in addition to more traditional Hoare-like rules and axioms. In such a way a sound and relatively complete axiomatization is possible, after suitable denesting for programs under conditions formulated by E. Clarke. (This investigation is related to works of H. Langmaack and E. R. Olderog, but was performed independently.)

J. V. Tucker: <u>The axiomatic semantics of programs based on Hoare's logic</u>

Floyd's Principle says that the semantics of a program language can be usefully defined by the axioms and rules of inference used to prove properties of programs in the language. Together with J. A. Bergstra, I have studied the semantics of while-programs based on the assumption that all what is known about while-program computation is what can be proved in Hoare's logic for partial correctness. The semantics AS one obtains is not the conventional operational semantics OS - the axiomatic semantics is non-deterministic, for example. Noteworthy is the fact that Hoare's logic is always complete w.r.t. AS whereas this is not the case for OS. OS is embedded in AS and on interpretations I for which the assertion language is expressive we find that OS = AS. The talk will present this material as the conclusion of a survey of my work with J.A. Bergstra on Hoare's logic.

M. Wirsing: Implementation of algebraic specifications

(Joint work with D. Sanella)

A notion for the implementation of one specification by others is given to handle parameterized specifications, hierarchical specifications (used for the modularization of large specifications) as well as loose specifications (having an assortment of non-isomorphic models).

Implementations are proved to compose "vertically" (two successive implementation steps compose to give one large step) and "horizontally". That is, under some restrictions two separate implementations of the parameterized specification and the actual parameter compose to an implementation of the application. Moreover, the implementation of a hierarchical system of specificarions can be done "locally" : Replacing a subtype of the system by its implementation leads (again under certain conditions) to an implementation of the overall system.

Berichterstatter: Manfred Broy

Adressen der Tagungsteilnehmer

Herrn
Dr. K. R. Apt
Erasmus Universiteit
Kamer H5-12
Burg. Outlaan 50
Rotterdam
Niederlande

Herrn
Prof. Dr. D. Björner
Dept. of Computer Science
Technische Universität of Denmark
Building 343,344

DK-2800 Lyngby

Herrn
Prof. Dr. C. Böhm
Universite di Roma
Instituto Mathematico
Gino Catelnuovo, P. Delle Science

I-00100 Roma
Italy

Herrn
Prof. Dr. W. Brauer
Institut für Informatik
Schlüterstraße 70

2000 Hamburg 13

Herrn
Dr. M. Broy
Institut für Informatik
TU München
Arcisstraße 21

8000 München 2

Herrn
Prof. Dr. V. Claus
Abteilung Informatik
Universität Dortmund
Postfach 500 500

4600 Dortmund

Herrn
Prof. Dr. E. M. Clarke
Aiken Comp. Lab.
Harvard University
Cambridge

Massachusetts 02138
USA

Herrn
Prof. Dr. A. Cremers
Abteilung Informatik
Universität Dortmund
Postfach 50 05 00

4600 Dortmund

Herrn
Prof. Dr. O. J. Dahl
Mathematisches Institut Avd. D.
Universität Oslo
Blindern

O s l o 3
Norwegen

Herrn
Dr. W. Damm
Institut für Angewandte Mathematik
und Informatik
Technische Hochschule Aachen
Büchel 29 - 31

5100 Aachen

Herrn
Prof. Dr. J. B. Dennis
MIT Laboratory for Computer Science
545 Main Street

Cambridge, Mass. 02139

USA

Herrn
Prof. Dr. P. Deussen
Institut für Informatik
Universität Karlsruhe
Postfach 63 80

7500 Karlsruhe

Herrn
Prof. Dr. J. Eickel
Institut für Informatik
Technische Universität München
Arcisstraße 21

8000 München 2

Herrn
Prof. Dr. G. Goos
Institut für Informatik
Universität Karlsruhe
Zirkel 2

7500 Karlsruhe

Herrn
Dr. D. Harel
The Weizmann Institute of Science
Department of Applied Mathematics

Rehovot, Israel

Herrn
Prof. Dr. C.A.R. Hoare
Merton College
Oxford University

Oxford
England

Herrn
Prof. Dr. G. Hotz
Institut für Angewandte Mathematik
und Informatik
Universität des Saarlandes
Im Stadtwald

6600 Saarbrücken

Herrn
Prof. Dr. K. Indermark
Institut für Informatik
RWTH Aachen
Büchel 29 - 31

5100 Aachen

Herrn
Prof. Dr. Neil Jones
Computer Science Department
Aarhus University
Ny Munkegade

DK-8000 Aarhus C
Dänemark

Herrn
Prof. Dr. P. Kandzia
Institut für Informatik
und Praktische Mathematik
Christian-Albrechts-Universität Kiel
Olshausenstraße 40-60

2300 Kiel 1

Herrn
Prof. Dr. M. Karpinski
Computer Science Dept.
Univ. of Edinburgh
Mayfield Road

Edinburgh EH 9322
Great Britain

Herrn
Dr. U. Kastens
Institut für Informatik
Universität Karlsruhe
Zirkel 2

7500 Karlsruhe

Herrn
Prof. Dr. I. O. Kerner
Pädagogische Hochschule
Abteilung Geographie
Wedardstraße

8060 Dresden
DDR

Herrn
Prof. Dr. F. Kröger
Institut für Informatik
Technische Universität München
Arcisstraße 21

8000 München 2

Herrn
Prof. Dr. H. Langmaack
Institut für Informatik
und Praktische Mathematik
der Christian-Albrechts-Universität
Olhausenstraße 40 - 60

2300 Kiel 1

Herrn
Dr. P. Lauer
Computing Laboratory
Univ. of Newcastle upon Type

Newcastle upon Type
UK

Herrn
Prof. Dr. J. Loeckx
Institut für Angewandte Mathematik
und Informatik
der Universität des Saarlandes
Im Stadtwald

6600 Saarbrücken

Herrn
Prof. Dr. O. Mayer
Institut für Informatik
Universität Kaiserslautern
Pfaffenbergstraße

6750 Kaiserslautern

Herrn
Dr. G. Plotkin
Computer Science Dept.
Univ. of Edinburgh
Mayfield Road

Edinburgh EH 9322
Great Britain

Herrn
Prof. Dr. A. R. Meyer
MIT, 545 Main Street
Room 806

Cambridge, Mass. 02139

Herrn
Prof. Dr. J. Stoy
Merton College
Oxford University

Oxford
England

Herrn
Dr. Möller
Institut für Informatik
Technische Universität München
Arcisstraße 21

8000 München 2

Herrn
Prof. Dr. Satoru Takasu
Research Institute for Mathematical
Sciences
Kyoto University
Sakyoku, Kyoto 606

JAPAN

Herrn
Prof. Dr. E. Neuhold
Institut für Informatik
Universität Stuttgart
Azenbergstraße 12

7000 Stuttgart 1

Herrn
Prof. Dr. B. Trakhtenbrot
Tel Aviv University
Department of Mathematical Sciences

Ramat-Aviv, Israel

Herrn
Prof. Dr. M. Nivat
Université Paris 7
Tour 45 - 55
2, Place Jussieu

F-75230 Paris

Herrn
Dr. J. V. Tucker
School of Mathematics and Computer Science
University of Bristol
University Walk
Bristol BS8 1 TW
England

Herrn
Dipl.-Inform. E.-R. Olderog
Institut für Informatik
und Praktische Mathematik
Christian-Albrechts-Universität Kiel
Olshausenstraße 40 - 60
2300 Kiel 1

Herrn
Prof. Dr. H. Walter
Technische Hochschule Darmstadt
Informatik
Hochschulstrasse 1

6100 Darmstadt

Herrn
Prof. Dr. M. Paul
Institut für Informatik
Technische Universität München
Arcisstraße 21

8000 München 2

Herrn
Dr. M. Wirsing
Institut für Informatik
TU München
Arcisstraße 21

8000 München 2