Abstracts of the Talks at the Fourth International Workshop on the Semantics of Programming Languages in Bad Honnef

March 14 - 18, 1983

Edited by

K. Indermark (*) and J. Loeckx

March 1983

A 83/07

Fachbereich 10

Universität des Saarlandes

D - 6600 Saarbrücken

(*) Rheinisch-Westfälische Technische Hochschule, Aachen

The present report contains the abstracts of the talks - in the order of their presentation - at the Fourth International Workshop on the Semantics of Programming Languages which took place in Bad Honnef on March 14 - 18, 1983, in the buildings of the Physikzentrum. The Workshop was sponsored by the European Association for Theoretical Computer Science and the Gesellschaft für Informatik and financially supported by the Deutsche Forschungsgemeinschaft.
We are indebted to all those who assisted us in preparing the Workshop and, in particular, to Ursula Goltz and Herbert Klaeren for their assistance in organizational matters and to Claus-Werner Lermen for his assistance in editing the report.

The editors

CONTENTS

ALGEBRAIC SOFTWARE SPECIFICATION - A TOY FOR THEORETICIANS OR A TOOL FOR PRACTITIONERS?

H.A. Klaeren (Aachen)

Algebraic software specification has become widely accepted in the scientific community. The question is , however, whether it is really of any practical value, which would mean that it can be used by system and program designers in a software development process. Experience shows that it is not too difficult to communicate an intuitive concept of algebra to practitioners, but it turns out that the semantics of equations by means of a generated congruence in general is not perspicuous at all. We propose a constructive method for algebraic software specification, where the operations are not implicitly specified by equations but by an explicit recursion on the generating operations of an algebra characterizing the underlying data structure. (This algebra itself may be equationally specified since we cannot assume that all data structures will correspond to free algebras.) This implies that we distinguish between generating and defined operations and that the underlying algebra has a mechanism of well-founded decomposition w.r.t. the generating operations. We show that the explicit specification of operations using so-called structural recursive schemata offers advantages over purely equational specifications, especially concerning the safeness of enrichments, the ease of semantics description and the separation between the underlying data structure and the operations defined on it.

For reference, see

H.A. Klaeren: A constructive method for abstract algebraic software specification, RWTH Aachen, Schriften zur Informatik und Angewandten Mathematik Nr. 78 (To appear in TCS)

MODULAR COMPILER DEFINITIONS

Harald Ganzinger (München)

This work aims at developing a method for defining compiler definitions that (i) exhibit a semantic processing based on

fundamental concepts of the source language; (ii) provide for increased independency of the concrete source language; (iii) are combinations of language-independent modules and (iv) are subject to automatic compiler generation. The basic idea is to view languages as combinations of projections to the sublanguages reflecting specific language facets. Relation symbols are used to define the syntax of the sub-languages. This allows for also characterizing the possible context of a syntactic construct. A compiler module is now viewed as representing these relations in terms of relations between semantic objects associated with the syntactic constructs. Algebraically this is captured by a specific notion of signature morphism. Known results about structuring the definition of abstract data type definitions can be extended to this case. Moreover, attribute grammars form a subclass of such morphisms. This gives on one hand a new algebraic view of attribute semantics and provides, on the other hand, for an efficient implementation of the concept.

DATA TYPES BY EXAMPLE OR DTs FOR PDLs

Eric G. Wagner (IBM Yorktown Heights)

In recent months we (ADJ & IBM) have been trying to improve and put firm foundations under a programming design language (a PDL) and its associated design methodology. In the past we have been concerned with abstract data types as isomorphism classes of total algebras, with specifications given by equational or conditional axioms and semantics given by initiality. However the requirements of the PDL, as to what must be specified and as to who must be able to write and read the specifications, have lead us to take a quite different approach. First, in order to cope with exceptions (error conditions) we view data types as partial, rather than total, algebras. Second, in order to produce specifications that are easier to read and write we present data types by means of models (concrete algebras described using naive set theory and assertions) rather than axiomatically. Furthermore to

gain the desired degree of abstraction we go beyond isomorphism classes to a notion of behavior equivalence. The result is a notion of an abstract module as a category of behaviorally equivalent algebras (compare Goguen-Meseguer). We further introduce a subclass of abstract modules, the "stately modules", which correspond (more closely) to state machines and provide the necessary framework in which to refine a high level design to a lower level design involving memory models. Lastly we point out that the strong typing required by the design methodology forces the PDL to have a fairly complex collection of base types, but, at the same time, if we allow the use of assertions in the definition of functions, we can start from a rather limited collection of operations on the base types.

FUNCTIONAL AND IMPERATIVE ABSTRACT SEMANTIC ALGEBRAS

Peter Mosses (Edinburgh)

Abstract Semantic Algebras (ASAs) are abstract data types whose values correspond to "actions" (or computations) that may be executed, and whose operators correspond to fundamental combinators for actions (e.g. sequential composition). ASAs are used as the basic ingredients of a special form of Initial Algebra (or Denotational) Semantics called "A-Semantics", described in [1].

Recent work has attempted to simplify the ASAs given in [1], the main concern being the elimination of the need for (infinite) indexed families of sorts and operators in signatures. New versions of the Fun and Imp ASAs of [1] were presented, and it was indicated how to obtain systematically many of the laws that should hold in their combination. Non-determinism plays a seemingly essential part in these new ASAs, which were partly inspired by the work of Main and Benson [2]. The use of a commutative "a_1, a_2" operator in Fun helps the relationship between Fun and Imp to be shown. Models (denotational and operational) for these new ASAs are currently being investigated to indicate the soundness and completeness of the equational specifications given.

[1] Mosses, P.D.: "Abstract semantic algebras!", in Proc. IFIP TC2 Work. Conf. on Formal Description of Programming Concepts II, Garmisch, June 1982, North-Holland (to appear).

[2] Main, M.G. and Benson, D.B.: "An algebra for non-deterministic distributed processes", Tech. Rep. CS-82-087, Washington State Univ.

SPECIFICATION-ORIENTED SEMANTICS FOR COMMUNICATING PROCESSES

E.-R. Olderog (Kiel & Oxford)
(joint work with C.A.R. Hoare)

Specification-oriented semantics attempts to unify the various semantical models proposed for Communicating Processes. The main guideline for its development is the following simple concept of process correctness: a process P is called correct w.r.t. a given specification S, abbreviated by P sat S, if every observation we can make about the behaviour of P satisfies S.

To realize this idea, we start from a set Obs of observations and define the space Spec of specifications S as a certain family of subsets of Obs. A specification-oriented semantics $[[\cdot]]$ assigns then denotationally to every process P a specification $[[P]]$ such that P sat S holds if $[[P]] \subseteq S$ is true. Technically, Spec is a complete partial order, and $[[\cdot]]$ maps every syntactic constructor of Communicating Processes onto a continuous operator on specifications. We derive general theorems for constructing and representing such continuous operators.

We then apply these methods to discuss three semantical models for Communicating Processes. These models differ in the structure of their observations which influences both the number of representable operators and the notion of correctness. In particular, both safety properties and liveness properties of processes can be described by P sat S.

SEMANTICS OF COMMUNICATING PROCESSES

Manfred Broy (München)

The semantics of a simple language for describing tightly coupled "synchronous" systems is defined. An operational semantics is given by term rewriting rules and a consistent fully abstract denotational semantics is defined based on the concept of observable behavior and advanced fixed point theory. Particular properties of the language are analysed and especially algebraic laws of the language are discussed. Finally particular aspects and problems of the formal definition of the semantics of such a language are treated also looking at other approaches found in the literature.

TESTING EQUIVALENCES FOR PROCESSES

M. Hennessy (Edinburgh)

We define a language for processes in which there are two methods of synchronising sub-processes called loose synchronisation and tight synchronisation. A notion of experimenting on processes is introduced which leads to definitions of when a process may pass an experiment and when it must pass an experiment. By connecting the experimenter to the process using the tight synchronisation we then define three pre-orders on processes called the synchronous pre-orders. By using the loose synchronisation primitive the asynchronous pre-orders are defined. For each of these pre-orders we prove that there exists a fully-abstract model in the sense of Scott. These models are defined using sets of equations and lead automatically to complete proof systems.

NAIVE DENOTATIONAL SEMANTICS

A. Blikle (Warsaw)
(joined work with A. Tarlecki)

The sophisticated mathematical framework of denotational semantics (Scott's reflexive domains and continuations) is not only discouraging for many practitioners but also leads to several

technical problems in applications. In this paper we investigate the possibility of developing denotational semantics where semantic domains are just sets and where states rather than continuations are transformed by the program's components. We show that a full mechanism of goto's may be described without continuations and that some procedural mechanisms (e.g. static binding with a hierarchy of procedural parameters - like in PASCAL) do not require reflexive domains. We show further, that if we relax the denotational principle and adopt a copy-rule semantics of procedures, then any procedural mechanism can be described in our framework.

DENOTATIONAL SEMANTICS FOR RECURSION WITH MERGE

J.-J. Chr. Meyer (Amsterdam)
(joined work with J.W. de Bakker, J.A. Bergstra, J.W. Klop)

First two ways were considered of assigning meaning to a class of statements built from a set of atomic actions (the 'alphabet'), by means of sequential composition, nondeterministic choice, recursion (μ-constructs) and merge (arbitrary interleaving). The first was linear time semantics (LT), in which trace sets were considered as the meaning of a statement. The second was branching time semantics (BT), as introduced by de Bakker and Zucker, in which the semantic domain is the metric completion of the collection of finite processes. For LT we have continuity of the operators composition and merge, which in fact can be proved in a direct, combinational way.
Next we saw a connection between LT and BT by means of an operation *trace* which assigns to a process its trace set, of which it can be proved that it is closed in case we have a finite alphabet. Furthermore, trace appears to be continuous.
Finally an algebraic approach as taken by Bergstra & Klop, was presented : first the basic process algebra for composition, union and merge, and then an extension of this to deal with communication.

ON "CONCRETE" SEMANTIC ALGEBRAS

A. Poigné (Dortmund)

Because of the complexity of denotational definitions of programming languages there is a need to develop specification techniques, which allow to modularize the semantic definitions. To achieve modularization the basic idea is to use the methods of abstract data type theory. But in contrast to the work of Mosses our approach is on classical denotational semantics.
We first extend a specification of basic data types by products, disjoint sums, function spaces, fixpoint operators and domain equations. Technically we give an algebraic specification of a cartesian closed category with additional properties. We show that an extended typed λ-calculus provides an initial model with respect to this specification. It is discussed how "concrete" semantic algebras may be obtained by restriction at the model, and how "concrete" semantic algebras can be manipulated.
Especially we investigate implementation as a specific modularization technique. Our notion of implementation is a restricted one in the sense of abstract data types, as we only allow to implement sorts by recursive domains and operations by recursive procedures. We prove that composition of correct implementations is correct if the composed implementation preserves termination, i.e. each term is implemented by a "finite" (fixpoint-operator-free) term. We give sufficient syntactical criteria to ensure correctness at composed implementations.

AN AUTOMATIC VERIFIER FOR TEMPORAL PROPERTIES OF COMMUNICATING FINITE STATE MACHINES

E.M. Clarke (Pittsburgh)

We give an efficient procedure for verifying that a finite state concurrent system meets a specification expressed in a (propositional) branching-time temporal logic. Our algorithm has complexity linear in both the size of the specification and

the size of the global transition graph for the concurrent system. We also show how the logic and our algorithm can be modified to handle *fairness*. We argue that this technique can provide a practical alternative to manual proof construction or use of a mechanical theorem prover for verifying many finite state concurrent systems.

INITIALITY AND COMPUTABILITY

J. Goguen and J. Meseguer (SRI)

A survey of results on initiality and computability was given, including several applications in computer science. In particular, the following were discussed: a software engineer motivation for an initial algebra approach to abstract data types (ADTs); many-sorted equational logic; generalized Peano axioms for ADTs; abstract syntax; denotational and fully abstract semantics; initial and final realizations of software modules; computable algebras; the relation of Gödel numberings to initiality; rewrite rules; and criteria for computable realizations of software modules to exist.

INSTITUTIONS: ABSTRACT MODEL THEORY FOR PROGRAM SPECIFICATION

J.A. Goguen and R.M. Burstall
(SRI International and University of Edinburgh)

There is a population explosion among the logical systems used in computer science. Examples include first order logic (with and without equality), equational logic, Horn clause logic, second order logic, higher order logic, infinitary logic, dynamic logic, process logic, temporal logic, and modal logic; moreover, there is a tendency for each theorem prover to have its own idiosyncratic logical systems. Yet it is usual to give many of the same results and applications for each logical system: of course, this is natural in so far as there are basic results in computer science that are independent of the logical system in which they happen to be expressed. But we *should not have to do*

the same things over and over again; instead, we should generalize, and do the essential things once and for all! Also, we should ask what are the relationships among all these different logical systems. This paper shows how some parts of computer science can be done in any suitable logical system by introducing the notion of an institution as a precise generalization of the informal notion of logical system. A first main result shows that if an institution is such that interface declations expressed in it can be glued together, then theories (which are just sets of sentences) in that institution can also be glued together. A second result gives conditions under which a theorem prover for one institution may be validly used on theories from another; this uses the notion of an institution morphism. Another result is that institution admitting free models can be extended to institutions whose theories may include, in addition to the original sentences, various kinds of constraints on interpretations; such constraints are useful for defining abstract data types, and include so-called 'data', 'hierarchy', and 'generating' constraints. Further results show how to define institutions that mix sentences from one institution with constraints from another, and even mix sentences and (various kinds of) constraints from several different institutions. It is noted that general results about institutions apply to such 'polyplex' institutions, including that mentioned above on gluing together theories. This paper also discusses applications of these results to specification languages, showing that much of this subject is independent of the institution used.

PROGRAMMING WITH PARAMETERIZED ABSTRACT OBJECTS IN OBJ

J. Goguen, J. Meseguer (SRI)
(joined work with D. Plaisted, Illinois)

OBJ is a logic based ultra-high level programming language that has been strongly influenced by modern programming methodology.

In particular, it provides facilities for user definable abstract data types, parameterized abstract objects, interactive programming (when the system detects errors, it provides suitable error messages and puts the user into an edit mode such that execution resumes when editing is completed), libraries, and other features. OBJ is based on equational logic, rather than on first order logic; because equations can be interpreted directly as rewrite rules, it is easy to see their computational significance as well as their logical significance. OBJ is so high level that we originally thought of it as a specification language. However, it now seems clear that data flow and other innovative architectures will support very efficient implementations of rewrite rules. This paper describes some experiences with an experimental OBJ implementation in LISP; in general, this experience courages our belief that equational logic based languages are a promising research direction that could have significant practical impact.

STRUCTURING THE DESIGN OF DATA ABSTRACTIONS

Udo Lipeck (Braunschweig)

This talk presents results of the thesis [1]: An algebraic calculus is introduced to structure a software design into a hierarchy of modules and implementations. Thus, a semantic basis for algebraic specification and design languages is provided.
Modules are assumed to be parametric abstract data types (padts) describing functorial type constructors between classes of algebras. Basic operations to construct padts from padts are the "parametric application" and the "reduction"; abstraction between padts is expressed by the "realization" relation. Then, the "implementation" of a "target" padt by a set of "base" padts is defined to be a term of construction operations such that the target is realized by the construction on the base. Related approaches may be

classified into special cases of this definition.
The analysis centres upon mutual compatibilities of the calculus operations and relations. Interchanging parametric applications and reductions leads to a normal form of construction terms. In the main theorem, the compatibilities of realizations are classified, especially those with parametric applications. The results permit of composing implementations constructively, i.e. by term substitution, if specific restrictions hold or typical situations are considered.

[1] U. Lipeck: Ein algebraischer Kalkül für einen strukturierten Entwurf von Datenabstraktionen. Dissertation/Bericht Nr. 148, Abteilung Informatik, Universität Dortmund, 1983

ELIMINATING RECURSION OVER ACYCLIC DATA STRUCTURES IN FUNCTIONAL PROGRAMS

C. Böhm (Rome)

(joined work with D. Kozen, IBM -Yorktown Heights)

Acyclic data structures mean here naturals, lists or strings (sequences), trees and dags. All these structures are defined by struct. induction. All total functions over these data based on struct. inductive definitions, like primitive recursion for naturals, tail-recursion for lists and tree-iteration for trees may be described by straight line programs without conditionals and/or loops.
The key idea is to represent each data structure as a functional which can be applied to other objects. In order to compute a function f recursively on the data structure d, d is applied as an operator to some "simpler" function f'. All the necessary recursion is contained in the structural definition of d, which is done once and for all at the beginning, independent of f'. The method becomes specially perspicuous by using a function programming style founded on combinators.

FAIRNESS IN ADA - A PROOF THEORETICAL VIEW

W.P. de Roever (Univ. of Utrecht / Nijmegen Univ.)
(joined work with Amir Pnueli (Weizmann Inst. / Harvard Univ.))

A fragment of ADA abstracting the communication and synchronization part is studied. An operational semantics for this fragment is given, emphasizing the justice and fairness aspects of the selection mechanisms.
An approximate notion of fairness is shown to be equivalent to the explicit entry-queues proposed in the reference manual. Proof rules for invariance and liveness properties are given and illustrated on an example. The proof rules are based on temporal logic.

REMARKS ON FAIR ITERATION CONSTRUCTS

David Park (Warwick University)

1. A predicate transformer for weak fair iteration is specifiable by

$$wp(WDO,R) = \mu X.((\bigwedge_i \neg B_i \wedge R) \wedge \bigvee_i G_i(X))$$

where $G_i(X) = \nu Y.(B_i \wedge wp(C_i,X) \wedge \bigwedge_j (B_j \Rightarrow wp(C_j, X \vee Y)))$

and WDO: wdo $B_1 \to C_1 \,[]\, \ldots \,[]\, B_n \to C_n$ od

is the construct to be interpreted fairly. [μ, ν are minimal, maximal fixpoint operators, respectively].

2. Unbounded nondeterminism, strong and weak fairness are interderivable, and require specifications with fixpoints of monotone, non-continuous functions.

3. A result of Paterson and M. Fischer has established that "queuing" is essential to implementing strong fairness. Automata corresponding to schedulers for strong/weak fairness require at least n!, n states respectively.

MODELLING THE DISTIRIBUTED TERMINATION CONVERTION OF CSP

Krzysztof R. Apt (Paris)

It is shown how the distributed termination convention of CSP repetitive commands can be modelled using other CSP constructs. The presented transformation suggests a simple implementation of this convention. We argue that this convention should be used as a compiler option.

AXIOMATIZATION OF IF-THEN-ELSE - REVISITED

Irène Guessarian (Paris)

We introduce classes of interpretations. We characterize the free and Herbrand interpretations for a class. We define the equational and relational classes of interpretations, study their properties and relate them to the literature. We apply this study to derive complete proof systems for deducing (in some (in)equational logic) all (in)equalitions valid in a class. We also apply this study to derive a complete equational specification of the tests, i.e. "IF...THEN...ELSE..." in various classes of algebras.

CHURCH-ROSSER PROPERTIES OF EQUATIONAL TERM REWRITING SYSTEMS

Jean-Pierre Jouannaud (Nancy)

The well known Knuth and Bendix completion procedure computes a convergent (both confluent and terminating) Term Rewriting System from a given set of equational axioms. This procedure was extended [L&B,77] [HUE,80] [R&S,81] [PAD,82] to handle Equational Term Rewriting Systems (ETRS in short), that is mixed sets of rules R and equations E in order to deal with the case where some axioms cannot be used as rewrite rules without loosing termination.
The first technique we develope[JOU,83] both unifies and extends Huet's, Lankford and Ballantyne's and Peterson and Stickel's results by describing a model of computation for ETRS at a more abstract level. As in the previous works,

a complete unification algorithm is required together with the so called E-termination property ε (termination of R modulo E). Moreover, we show that complete sets of E-critical paths are required only when non linear rules are involved, providing a new powerful and efficient E-completion algorithm for such theories [J&K,83].
The second technique we develop [JKR,83] deals with ETRS that do not satisfy E-termination, which arises for example when E contains an idempotent axiom. As previously, an abstract model of computation for such ETRS is described which only requires termination of R, yet provides a simplified and generalized version of Padawitz's results.

[HUE,80] HUET, G.: "Confluent reductions: Abstract properties and application to Term Rewriting Systems", JACM 27-4.

[JOU,83] JOUANNAUD, J.P.:"Confluent and Coherent Equational Term Rewriting Systems. Application to proofs in Data Types", CAAP-83, to appear in LNCS

[J&K,83] JOUANNAUD, J.P., KIRCHNER, H.: "An efficient completion procedure for Equational Term Rewriting Systems", draft

[JKR,83] JOUANNAUD, J.P., KIRCHNER, H., REMY, J.L.: "Church-Rosser Properties of weakly terminating Equational Term Rewriting Systems", submitted

[L&B,77] LANKFORD, D.S., BALLANTYNE, A.M.: "Decision procedures for simple equational theories with permutative axioms: complete sets of permutative reductions", Report, Univ. of Texas at Austin

[R&S,81] PETERSON, G.E., STICKEL, M.E.: "Complete sets of reductions for equational theories with complete unification algorithms", JACM 28-2

[PAD,82] PADAWITZ, P.: "Equational data type specification and recursive program scheme", in "Formal Description of Programming Concepts 2", D. BJØRNER Ed., North-Holland

RATIONAL COMPLEXITY OF INFINITE TREEES

K. Indermark (Aachen)

Rational schemes interpreted over rank-free derived algebras permit a simple algebraic analysis of higher type recursion. Their equivalence is characterized by infinite trees. Measuring their complexity by the size of finite subtrees we obtain a direct proof of Damm's recursion hierarchy theorem.

VERIFICATION WITH ABSTRACT DATA TYPES, OR: CORRECTNESS PROOFS WITH FEWER TEARS

Jacques Loeckx (Saarbrücken)

It is claimed that a top-down programming system based on algorithmic specifications (see e.g. [1]) leads to easy correctness proofs. More precisely, structural induction on the data together with unfolding allows to prove "most" properties.

The claim is first illustrated with the help of the iterative program counting the tips of a binary tree which is classically used to illustrate the intermittent assertions method [2]. A formal proof of total correctness of this program takes about half a page and may be performed by an AFFIRM-like system (cf. [3]).
The proof methodology applied is shortly discussed. It is shown to be applicable whenever the property to be proved contains assertions about defined values only.

[1] J. Loeckx, *Algorithmic specifications of abstract data types*, Proc. ICALP 81 (Acre), LNCS 115, pp. 129-147, 1981

[2] R.M. Burstall, *Program proving as hand simulation with a little induction*, Proc. IFIP Congress 74, pp. 309-312, 1974

[3] J. Loeckx, *Proving properties of algorithmic specifications of abstract data types in AFFIRM*, AFFIRM-MEMO-29-JL, USC - ISI, Marina del Rey, 1980

PARTICIPANTS

K.R. Apt / LITP / Université Paris 7 / 2, place Jussieu / F - 75251 Paris Cedex 05

A. Blikle / Institute of Computer Science / Polish Academy of Sciences / P.O. Box 22 / 00-901 Warsaw PKiN / Poland

C. Böhm / Via S. Crescenziano 20 / I-00199 Roma / Italy

M. Broy / Institut für Informatik der Technischen Universität München / Postfach 20 24 20 / 8000 München 2

R.M. Burstall / Department of Computer Science / James Clerk Maxwell Building / The King's Buildings / Mayfield Road / GB - Edinburgh EH9 3JZ

E.M. Clarke / Department of Computer Science / Schenley Park / Carnegie-Mellon University / Pittsburgh, Pennsylvania 15213 / USA

G. Cousineau / LITP / Université Paris 7 / 2, place Jussieu / F - 75230 Paris Cedex 05

H. Ganzinger / Institut für Informatik der Technischen Universität München / Postfach 20 24 20 / 8000 München 2

J.A. Goguen / SRI International / Menlo Park, CA 94025 / USA

I. Guessarian / LITP / Université Paris 7 / 2, place Jussieu / F - 75251 Paris Cedex 05

M. Hennessy / Department of Computer Science / James Clerk Maxwell Building / The King's Buildings, Mayfield Road / GB - Edinburgh EH9 3JZ

G. Huet / INRIA / Domaine de Voluceau / Rocquencourt / B.P. 105 / F - 78150 Le Chesnay

K. Indermark / Lehrstuhl Informatik II / Rheinisch Westfälische Technische Hochschule / Büchel 29 - 31 / 5100 Aachen

J.P. Jouannaud / CRIN / Université de Nancy I / C.O. 140 / F - 54037 Nancy Cedex

H. Klaeren / Lehrstuhl für Informatik II / Rheinisch Westfälische Technische Hochschule / Büchel 29 - 31 / 5100 Aachen

H. Langmaack / Institut für Informatik / Christian-Albrecht-Universität / Olshausenstr. 40 - 60 / 2300 Kiel 1

U. Lipeck / Institut für Theoretische und Praktische Informatik / Technische Universität / Postfach 3329 / 3300 Braunschweig

J. Loeckx / Fachbereich 10 - Angewandte Mathematik und Informatik / Universität des Saarlandes / 6600 Saarbrücken

J. Meseguer / SRI International / 333 Ravenswood Avenue / Menlo Park, CA 94025 / USA

J.-J.Ch. Meyer / Free University Amsterdam / Math. Department / De Boelelaan 1081 / 1081 HV Amsterdam / The Netherlands

P. Mosses / University of Edinburgh / The King's Buildings, JCMB / GB - Edinburgh EH9 3JZ

E.-R. Olderog / Oxford University Computing Laboratory / Programming Research Group / 8 - 11 Keble Road / Oxford OX1 3QD /England

D. Park / University of Warwick-Coventry / Warwickshire CV4 7AL / Great Britain

A. Poigné / Lehrstuhl für Informatik / Universität Dortmund / Postfach 500 500 / 4600 Dortmund 50

W. de Roever / Department of Computer Science / University of Utrecht / Budapestlaan 8 / P.B. 80012 / 3508 TA Utrecht / The Netherlands

W. Thomas / Lehrstuhl Informatik II / Rheinisch Westfälische Technische Hochschule / Büchel 29 - 31 / 5100 Aachen

E.G. Wagner / Mathematical Sciences Department / IBM / Thomas J. Watson Research Center / Box 218 / Yorktown Heights, N.Y. 10598 / USA

Observers

W. Damm / Lehrstuhl Informatik II / Rheinisch Westfälische Technische Hochschule / Büchel 29 - 31 / 5100 Aachen

E. Fehr / Lehrstuhl Informatik II / Rheinisch Westfälische Technische Hochschule / Büchel 29 - 31 / 5100 Aachen

R. Gerth / Rijksuniversiteit Utrecht / Vakgroep informatica / Princetonplein 5 / Postbus 80.002 / 3508 TA Utrecht / The Netherlands

U. Goltz / Lehrstuhl Informatik II / Rheinisch Westfälische Technische Hochschule / Büchel 29 - 31 / 5100 Aachen

B. Josko / Lehrstuhl Informatik II / Rheinisch Westfälische Technische Hochschule / Büchel 29 - 31 / 5100 Aachen

C.-W. Lermen / Fachbereich 10 - Angewandte Mathematik und Informatik / Universität des Saarlandes / 6600 Saarbrücken

W. Reisig / Lehrstuhl Informatik II / Rheinisch Westfälische Technische Hochschule / Büchel 29 - 31 / 5100 Aachen