Manfred Broy, Peter Deussen,
Ernst-Rüdiger Olderog, Willem-Paul de Roever
(editors)

Concurrent Systems: Semantics, Specification, and Synthesis

Dagstuhl-Seminar-Report; 9
11.3.1991 - 15.3.1991 (9111)

ISSN 0940-1121

Tel.: +49-6871 - 2458
Fax: +49-6871 - 5942

Das Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI) ist eine gemeinnützige GmbH. Sie veranstaltet regelmäßig wissenschaftliche Seminare, welche nach Antrag der Tagungsleiter und Begutachtung durch das wissenschaftliche Direktorium mit persönlich eingeladenen Gästen durchgeführt werden.

Bezugsadresse: Geschäftsstelle Schloß Dagstuhl
Informatik, Bau 36
Universität des Saarlandes
W - 6600 Saarbrücken
Germany
Tel.: +49 -681 - 302 - 4396
Fax: +49 -681 - 302 - 4397
e-mail: dagstuhl@dag.uni-sb.de

Report on the Dagstuhl - Workshop

"Concurrent Systems: Semantics, Specification, and Synthesis"

11 - 15 March 1991

Organizers: M. Broy, Techn. Univ. München
P. Deussen, Univ. Karlsruhe
E.-R. Olderog, Univ. Oldenburg
W. P. de Roever, Univ. Kiel

Introduction

Concurrent, interacting systems are becoming more and more widespread. Examples include distributed algorithms, operating systems, communication protocols, computer architectures, and digital circuits.

The theoretical challenges of such systems resulted in various formal approaches to their specification, analysis and verification. These approaches are based on operational models, process algebras, temporal and modal logics, and compositional calculi for design and verification. Current research problems in the theory of concurrent systems are the search for suitable notions of semantic equivalence, the analysis of system refinement, the issue of true concurrency versus interleaving semantics, and the formal description of real-time and probabilistic systems.

On the other hand, the construction of realistic distributed algorithms or the implementation of concurrent systems is mostly ignored in this theoretical work. However, the correct synthesis of such systems is often of vital importance. To cope with these tasks, pragmatic and - from a theoretical point of view - more ad hoc approaches have been developed. These approaches include iterative programs, interface specifications, automatic finite state verification, implemented tools for system specification, and separate industrial methods.

The organizers of this workshop find that too little communication is going on between the more theoretical and the more applied work on concurrent systems. The aim of this workshop was therefore to bring together representatives of both sides. The result was a very stimulating meeting with 26 talks (see the abstracts - in the order of presentation - below), lots of discussion, and one demonstration of a tool for interactive system design. All participants welcomed the plan for a successor workshop where specific case studies should be discussed.

Abstracts

Manfred Broy, Tech. Univ. München:

Functional System Specification: Higher Order Messages, Real Time and Operating System Structures

(joint work with C. Dendorfer, Tech. Univ. München)

A notation and a semantic model is introduced for specifying interactive components of concurrent distributed systems that depend on real time properties and exchange higher order messages such as processes again. This way a formal framework is obtained for the specification of operating system structures.

Jozef Hooman, Eindhoven Univ. of Technology:

Specification and Compositional Verification of Real-Time Systems using Metric Temporal Logic

To specify and verify real-time systems, we consider an Occam-like programming language with synchronous communication along unidirectional channels that connect two processes. As a starting point for a compositional axiomatization, a denotational semantics for this language is given, describing the real-time behaviour of programs. Specifications are written in a real-time version of temporal logic, called Metric Temporal Logic, in which bounds have been added to the temporal operators. To verify that a program satisfies a specification written in this logic, we formulate a compositional proof system.

Ernst-Rüdiger Olderog, Univ. Oldenburg:

Trace-based Specification and Derivation of Communicating Programs

A simple specification language SL_0 for communicating systems is presented together with some transformations for the derivation of Occam-like programs from SL_0 specifications. The main idea of SL_0 is to split the description of the desired process behaviour into a trace part and a state part. The trace part specifies the sequencing constraints on the interface channels of SL_0 whereas the values communicated are ignored. These values are specified in the state part with the help of internal state variables. The motivation for organizing an SL_0 specification in two parts is to ease its stepwise transformation into Occam. The trace part yields a synchronization skeleton and the state part completes this skeleton to a communicating program by adding purely sequential parts.

Philippe Schnoebelen, LiFia-Imag Grenoble:
Strong Bisimilarity of Nets Revisited

Following Olderog's seminal idea, we present a new definition of bisimulation between *places* of Petri nets. We compare bisimulation on places with several classical bisimulations on markings: bisimulation on places preserves more of the structure of the net. We present a way of collapsing bisimilar places, giving canonical representatives (modulo place bisimulation) of nets.

Willem-Paul de Roever, Christian-Albrechts-Univ. Kiel:
Assertional Data Reification Proofs: Survey and Perspective

(joint work with J. Coenen, Eindhoven Univ. of Technology, and J. Zwiers, Univ. of Twente)

Three wellknown methodologies for proving data refinement due to Jones, Reynolds and Back have been presented up till now separately in the literature.

We investigate how these methodologies are related:

- by developing a modest predicate transformer framework;
- by relating the 4 known varieties for proving refinement and expressing them as verification conditions within our framework;
- by analyzing Reynolds' method and VDM-style refinement proofs, stating their associated verification conditions for partial correctness, and then, through a change of interpretation to strict predicates and total relations to include non-termination, for total correctness;
- by mentioning how a restricted form of Back's general theory can also be characterized.

Reinhard Gotzhein, University of Hamburg:
Requirement Specification for Open Distributed Systems

When specifying requirements on an open distributed system, it is essential to capture both the conceptual system structure and the possible system behaviour. We use the formalism of temporal logic with operators for the future, the past, for event occurrence and for interval construction to model and specify these aspects. We show how the logic can be applied to specifying and reasoning about the Initiator-Responder service, a simplified version of the Abracadabra service from the protocol standardization literature.

Dominique Bolignano, Bull Corporate Research:

A Naive Approach to Formal Methods: an Application to Concurrency

We present the basis for an approach which is to provide an intuition based and general means of modeling. In order to achieve this, we introduce finite annotated graphs which we call semantic graphs. These graphs allow the finite structure of a software system to be captured while precisely describing potentially infinite states and behaviours. Data and basic "transitions" can be described using dedicated formal languages such as VDM. The approach is intended to be used at this first stage as a framework for comparing and unifying various approaches.

Steven Klusener, CWI Amsterdam:

Completeness in Real Time Process Algebra

Recently, J. Baeten and J. Bergstra extended the process algebra ACP with real-time, resulting in ACPρ. The aim is to do protocol verification on protocols concerning time, thus containing time outs etc. The idea is that every action "a" has now a time stamp "t" associated with it. This time stamp can be interpreted absolute (from the beginning) or relative (from the previous action). Since every term in relative time can be rewritten in a term in absolute time easily, we restrict ourselves to absolute time during the talk. ACPρ has the interesting construct of integration: $\int_{v \in <0,1>} a(v)$ denotes the process which can execute the action "a" somewhere within <0,1>. A variable binding mechanism is contained in integration, e. g. $\int_{v \in <0,1>} a(v) = \int_{w \in <0,1>} a(w)$. Hence in order to reason with all terms, we have to be able to reason with terms containing free time variables as well. Therefore we introduce conditional terms, which are simply guarded terms. Then we can express laws like

$$a(v) \,.\, b(w) = v < w \rightarrow a(v) \,.\, b(w) + v \geq w \rightarrow a(v) \,.\, \delta.$$

Furthermore a restricted notion of integration is introduced called "prefixed" integration. This allows us to write $\int_{v \in <0,1>} a(v).p$ but excludes $\int_{v \in <0,1>} a(v).b(v+10)$. Due to this restricted notion of integration and the introduction of conditionals we obtain completeness for ACPρ. Soundness and the Congruence Theorem are discussed shortly.

Joost N. Kok, Univ. of Utrecht:

On the Relation of Logic Programming and the Refinement Calculus

Back´s Refinement Calculus (BRC) is a calculus for the development of programs. It is a mixed formalism (specification and programs can be mixed) based on the preservation of total correctness.

We relate Logic Programming (including Horn Clause Logic, Pure Prolog and parts of Concurrent Logic languages) to BRC by identifying a class of programs in the command language of BRC. This enables us to

- establish a flow-of-control semantical model for LP
- extend LP with specification constructs
- obtain lines for program development:
    * from LP to distributed programs (LP = declarative = specification)
    * use LP as target of refinements (LP = implementation)

Also it provides us with some interesting correspondences:

unification	<->	angelic updates
HCL choice	<->	angelic choice
commit	<->	guarded actions

Bernhard Steffen, RWTH Aachen:

Computing Behavioural Relations, Logically

(joint work with R. Cleaveland, North Carolina State Univ.)

A model-checking algorithm for an intuitionistic fragment of the modal mu-calculus is developed, and it is shown how it may be applied to the efficient computation of behavioural relations between processes. The algorithm is linear in the size of the process times the size of the formula, and thus improves on the best known algorithms for a similar logic, where worst-case complexity is proportional to the size of the process times the square of the size of the formula. The method for computing behavioural preorders that the model-checker induces is also more efficient than existing algorithm.

Bent Thomsen, ECRC München:

Higher Order Processes

We present a calculus of communicating systems which allows one to express sending and receiving processes. Essential to this calculus is the treatment of restriction as a

static binding operator on port names. The calculus is given an operational semantics using labelled transition systems which combines ideas from applicative transition systems described by Milner. The higher order process calculus enjoys algebraic properties similar to those of CCS only needing obvious extra laws for sending and receiving processes. A calculus allowing sending and receiving processes is a powerful description tool. We show how to encode various programming paradigms including functional, imperative and object-oriented programming paradigms. Processes as first class objects enable the description of networks with changing interconnection structure. There is a close connection between the higher order process calculus and the π-calculus described by Milner, Parrow and Walker: the two calculi can simulate one another.

W. Reisig, Tech. Univ. München:

Petri Nets and UNITY: Combining their Respective Advantages

UNITY combines a temporal logic for abstract specifications with a programming notation for algorithmic design. We suggest to replace this programming notation by high level Petri nets. A symbolic version of such nets is employed, as expressive as conventional programming languages. This leads to a couple of advantages:

- fairness assumptions are introduced only if justified by the respective algorithm,
- early fixing of program variables and assignment statements are avoided,
- established Petri net proof techniques can be applied,
- data structures may be specified algebraically,
- concurrency and synchronization issues are dealt with explicitly.

The introduction of concurrency as a modality, exploiting the specific structure of Petri nets, leads to transparent proofs.

Ursula Goltz, GMD St. Augustin and Univ. Erlangen:

Towards a Modular Hierarchical Calculus for System Design

(joint work with R. van Glabbeek, Stanford Univ.)

A theoretical framework aiming at modular design of reactive systems is considered. In particular, the hierarchical structure of the design is representable by a construct for changing the level of abstraction.

We use a language based on process algebras, enriched by an operation refining actions by processes. We give a compositional semantic domain. The interplay

between refinement (substitution) and semantic refinement is investigated. The power and the limitations of the approach are discussed by applying it to a very simple communication protocol.

Alessandro Giacalone, ECRC München:
The Semantics of Facile: A Symmetric Integration of Functional and Concurrent Programming

We present the semantics of the Facile language, representative of a class of languages that combine typed λ-calculus with constructs for concurrent programming. Examples include PFL, Amber and PML. Facile supports a symmetric integration of functional and concurrent programming. Its expressions are very general and may dynamically create processes and cause communication. We develop the semantics in three steps. First, an operational semantics is developed, based on a structural approach, which characterises a notion of program execution. Next, a notion of observable behaviour of a program is introduced, based on a notion of window of observation: a set of channels through which we may decide to observe a program. The concept of window is then used to obtain a parametric notion of equivalence between programs. The notion of equivalence generalizes Milner´s bisimulation to support reasoning about higher-order processes as well as systems whose interface may change dynamically. The equivalence supports reasoning about, both, observable behaviour of processes and evaluation of expressions. We conclude with a discussion about compositional reasoning using the notion of window-parameterized equivalence.

Michael P. Fourman, Univ. of Edinburgh and Abstract Hardware Ltd.:
The LAMBDA/DIALOG System

We argue that multi-level specification and refinement requires a general-purpose formalism (at least until we have more experience that might expose new abstractions). We use higher-order predicate logic and treat time explicitly by modelling devices as predicates relating streams of values on ports. In this we follow Gordon and Hanna. As abstraction relations are often partial, we use a logic (that allows terms which do not denote) of partial functions. We model a design state by a derived rule schema and show how the design process may be supported by a tool based on computer-assisted formal reasoning with an interface presented in the familiar "schematic capture" idiom. This is the LAMBDA/DIALOG system.

Michael R. Hansen, Techn. Univ. of Denmark:
Duration Calculus

(joint work with Zhou Chaochen and A. P. Ravn, Techn. Univ. of Denmark, and C.A.R. Hoare, Oxford Univ.)

Duration calculus is an extension to interval temporal logic whose purpose is to reason about designs and requirements for time-critical systems. Its distinctive feature is the ability to reason about durations of (propositional combinations of) states. We give a formal system of durations which is shown to be relative complete wrt. interval temporal logic.

Reino Kurki-Suonio, Tampere Univ. of Technology:
The DisCo Language and Temporal Logic of Actions

DisCo (Distributed Cooperation) is a specification language for reactive systems that combines the action-oriented approach of joint actions with object-oriented ideas of classes and inheritance. It is intended for structured derivation of semi-executable models that can be reasoned about, visualized and animated. Modularity in DisCo is essentially based on superposition whereby the system state can be extended, actions can be refined, and further actions can be introduced. The talk related DisCo to TLA, Lamport's temporal logic of actions, showing how superposition and union of DisCo systems are syntactic approximations of corresponding more general notions in TLA. Restricting the language to these approximations is motivated by the preservation of both safety properties and (semi-)executability.

Hélenè Collavizza, Univ. de Provence, Marseille:
Specifying the Micro-program Parallelism in Microprocessors of the Von Neumann Style.

(joint work with D. Borrione, Univ. de Provence, Marseille)

In order to verify significant μ-processors, we believe that the proof process must be decomposed into successive steps of verification between adjacent description levels. Furthermore we recommend the use of a functional formalism. The " μ–program" level takes into account the memory/processor information exchanges, and the internal operations that can be executed in parallel. For pipelined processors we define the "implicit parallelism" of two μ-program fragments executed in two independent pipiline

stages, and the "partical parallelism" of two contiguous pipeline stages. We also define the "implicit parallelism" of several internal operations which modify different state variables and can possibly be executed during the same μ-instruction.

Shengzong Zhou, Univ. Saarbrücken:
Compositional Temporal Logic Specifications

Temporal Logic is a powerful tool for specifying and verifying reactive systems. Up to now, one specifies a program by describing the global behaviour of the program, or by describing the behaviour of the possible parallel composition of the program with an arbitrary program. The former approach results in the following disadvantages: globality, non-modularity and non-compositionalty. The latter approach suffers from the complexity of the proof rules and proof procedures. The talk presents a new approach for temporal logic specifications to overcome these deficiencies. We specify a program by describing the behaviour which every (or some) individual statement of the program has. To show our approach, we introduce a temporal logic language XYZ/U, of which one subset is an executable imperative language. We define some new temporal logical operators describing the behaviour which every (or some) individual statement of a program has. A program´s specifications expressed with the new operators imply directly the corresponding classical specifications. With the help of these new operators, a program´s specifications can be easily deduced form the specifications of its components. This facilitates the compositional specification design, synthesis and verification of large programs.

Cliff B. Jones, Univ. of Manchester:
Interference Resumed

The rely/garantee approach set out to extend operation decomposition methods for sequential programs to cover concurrent shared-variable systems. The essential step was to recognise that interference has to be specified in order to achieve a notion of compositionality. Stephen´s thesis (Univ. of Manchester) has addressed the main shortcomings of my earlier work. This talk showed how to break away form the n-tuple specifications. A semantic model (resumptions), some operators for predicates over state pairs, and some examples were sketched.

Frank de Boer, Eindhoven Univ. of Technology:

A Compositional Trace Model for Asynchronous Communication

(joint work with J. Kok, Univ. of Utrecht, J. Rutten, CWI Amsterdam, and C. Palamidessi, Univ. of Pisa)

We develop a general framework for a variety of concurrent languages all based on asynchronous communication, like data flow, concurrent logic, concurrent contraint languages and CSP with asynchornous channels. The main characteristic of these languages is that processes interact by reading and modifying the state of some common data structure. We abstract from the specific features of the various communication mechanisms by means of a uniform language where actions are interpreted as partially defined transformations on an abstract set of states. Suspension is modelled by an action being undefined in a state. The languages listed above can be seen as instances of our paradigm, and can be obtained by fixing a specific set of states and interpretation of the actions. The computational model of our paradigm is described by a transition system in the style of Plotkin´s SOS. A compositional model is presented that is based on traces (of pairs of states). This sharply contrasts with the synchronous case, where some additional braching information is needed to describe deadlock. In particular, we show that our model is more abstract than the standard failure set semantics (that is known to be fully abstract for the classical synchronous paradigms). We also investigate the problem of full abstraction, with respect to various observation criteria. To tackle this problem, we have to consider the particular features of the specific languages. We study some cases, for which we give a fully abstract semantics.

Job Zwiers, Twente University:

Program Development from Partical Order Specifications

Two different schools of thought can be observed in recent developments around specification, verification and design of distributed systems. One departs from the compositionality principle, based on algebraic structure of systems. The other school rejects this on the basis that often the algebraic structure does not match the perceived structure for analysis and design of such systems. Moreover, algebraic structure would be geared already to a particular architecture, which should not be the case for the initial design stage of systems. My claim ist that many (most?) of these disadvantages of the algebraic style can be overcome by introducing the right compositional

operations. Important here is that such operations can be explained only in models that take causality into account. Actually we use a variation of the well known Pomset model. We show for instance that the "communication closed layers principle" can be formulated as an algebraic law in this setting. Another example is serializability in databases which can be explained now by means of action refinement.

Werner Damm, Univ. Oldenburg:

AADL: A Net Based Method for Specification and Verification of Distributed Systems

(joint work with G. Döhmen, B. Josko, and R. Schlör, Univ. Oldenburg)

Using the design of an asynchronous bus-protocol as example, the talk outlined a modular approach to specification and verification of reactive systems.

The specification method is based on an assumption/commitment style temporal logic MCTL. An introductory part of the talk discussed the use of timing diagrams as a graphical representation of a certain class of temporal formulae. Such timing diagrams contain for each atomic proposition a row depicting the truth value of this proposition over time, and depict causal dependencies between changes of the truth of the atomic propositions graphically by different types of arrows corresponding to different "idioms" of the temporal logic. A formal definition of the semantics of such timing diagrams is currently under preparation.

AADL provides for implementation modules allowing synchronous or asynchronous communication between modules. The verification of the correctness of such an implementation module against its assumption/commitment style temporal specification is based on a modelchecking procedure for MCTL. The construction of the model out of the implementation module uses as intermediate step the compilation of implementation modules into a certain class of Petri-nets. To verify the correctness of a parallel composition of modules, we refrain from employing modelcheking in order to avoid the state-explosion problem but rather employ proof rules, which allow to infer the behaviour of the composite model form the temporal specification of its constituents. These rules address in particular elimination of assumptions and allow to handle circular assumptions as long as they deal only with a restricted class of safety properties.

Frank A. Stomp, Abo Akademi:

Preventing Cycles In Graphs: Formal Program Derivations

- Experiments with Reactive Refinements -

The problem of formally and completely deriving complex programs is addressed. For this purpose, derivations of a sequential, of a concurrent, and of a distributed program for preventing cycles in finite, directed, and acyclic graphs under additions of edges to that graph and deletions of edges from it are considered.

Transformations are carried out using a relation called *reactive refinement within context*. This relation is a modified version of Back's reactive refinement relation and preserves all stutter-free temporal properties (in certain environments) when a transformation is applied. It is argued that such a relation is, in general, too strong for carrying out derivations. In particular, it is shown that a (convenient) specification for the sequential case cannot be used to specify the concurrent case; and that a (convenient) specification for the concurrent case cannot be used to specify the distributed case.

Rob Gerth and Ruurd Kuiper, Eindhoven Univ. of Technology:

Action/Interface Refinement

Suppose one has a system that has a *synchronous* interface with its environment. Now, suppose that one refines this system and changes its interface to an *asynchronous* one. Whatever is meant here by refinement, it cannot be standard (process) refinement since the interface actions have changed; nor is it action refinement in the sense that a process is substituted for an action, as the intention presumably is to allow the system to proceed without having to wait until the environment is willing to synchronize. What comes closest is action refinement using Zwier's notion of conflict composition (suitably generalized).

Standard process refinement can be seen as semantic inclusion:

$$S \text{ refines } T \quad \text{iff} \quad Beh(S) \subseteq Beh(T),$$

for a suitable notion of behaviour Beh. This can be trivially rewritten as

$$S \text{ refines } T \quad \text{iff} \quad \forall s \in Beh(S)\ \exists t \in Beh(T)\ s=t.$$

In our view, refinement while changing the interface generalizes standard refinement in that instead of equality to relate behaviours one now uses a relation that defines how the interface changes:

$$S \text{ refines}_C T \quad \text{iff} \quad \forall s \in Beh(S)\ \exists t \in Beh(T)\ s\ C'\ t.$$

Here, C defines how the interface changes - so it might express that a synchronous communication action c_v now becomes an asynchronous send action s_v with a later corresponding receive action r_v which might be represented as a pair of linear temporal logic formulae (c_v , $s_v \ \Lambda \Diamond r_v$) - and C´ is the canonical extension of C to behaviors. There are some subtleties in defining this canonical extension since one would like to relate a behavoir c_1c_1 to $s_1s_1r_1r_1$ and to $s_1r_1s_1r_1$ but not to, e.g., $s_1r_1r_1$. In other words such an extension calls for a partitioning of the actions in the behaviours.
The second part of the talk concerns a verification criterion for this notion. In fact we concentrate on a subproblem of this, namely: given a set of LTL formulae and a program how can one prove that on every behavior the actions can be partitioned in such a way that every partition satisfies one of these formulae.
We develop a Manna and Pnueli style proof method for this and show how the proof obligation can be reduced to a set of *leads to* properties of some suitably choosen invariants. One of the surprising results - at least to us - is that this reduction is possible at all. Initially, we expected some form of branching time reasoning to be inevitable.

Bernard Le Goff, Bull Research Center, Versailles:
Time and Specification of Concurrency: a Synchronous Approach
In the framework of formal specifications, the complexity of models makes automatic proofs impossible, in general. The temporal indeterminism specified by models is a cause of the complexity of models. The synchronous approach introduces a referential time onto which events can be mapped. This referential time is simply an infinite well-ordered set: a time index. The map from events to time indexes allows us to specify either simultaneity of events or the fact that one occurs before another. No duration notion exists: events are instantaneous. Such a time is a logical time. Time can be represented by several time indexes. What is the structure of the space of time indexes. Does one including all the others exists?
The answer to this question determines whether a deterministic implementation exists. Moreover, the process of answering this question can detect temporal properties of models. An algorithm that answers this question is currently implemented into the SIGNAL compiler.

Participants:

Michel **Allemand**
Université de Provence, Case S, UFR MIM
3 Place Victor Hugo
13331 Marseille Cedex
France
amichel@dhend.imag.fr

Frank **de Boer**
Technical University Eindhoven
5600 MB Eindhoven
The Netherlands
frb@cwi.nl

Dominique **Bolignano**
BULL Research Center
68, route de Versailles
78430 Louveciennes
France
Dominique.Bolignano@crg.bull.fr

Manfred **Broy**
Institut für Informatik
Technische Universität München
Postfach 20 24 20
W-8000 München 2
broy@lan.informatik.tu-muenchen.dbp.de

Hélène **Collavizza**
Université de Provence, Case S, UFR MIM
3 Place Victor Hugo
13331 Marseille Cedex
France
bcl@frccup51.bitnet

Werner **Damm**
FB Informatik
Universität Oldenburg
Postfach 25 03
W-2900 Oldenburg

Peter **Deussen**
Institut für Logik, Komplexität und
Deduktionssysteme
Postfach 6980
W-7500 Karlsruhe 1
deussen@ira.uka.de

Michael P. **Fourman**
Dept. of Comp. Science
JCMB, King's Buildings,
Edinburgh EH9 3JZ
Scotland, UK
mike@lfcs.ed.ac.uk

Rob **Gerth**
Technical University Eindhoven
5600 MB Eindhoven
The Netherlands
robg@win.tue.nl

Alessandro **Giacalone**
ECRC
Arabellastr. 17
W-8000 München 81
ag@ecrc.de

Ursula **Goltz**
GMD - F1P
Schloss Birlinghoven
W-5205 St. Augustin 1
goltz@gmdzi.gmd.de

Reinhard **Gotzhein**
FB Informatik
Universität Hamburg
Bodenstedtstr. 16
W-2000 Hamburg 50
gotzhein@rz.informatik.uni-hamburg.dbp.de

Michael R. **Hansen**
Dept. of Computer Science
Technical University of Denmark
DK-2800 Lyngby
Denmark
mrh@id.dth.dk

Jozef **Hooman**
Technical University Eindhoven
5600 MB Eindhoven
The Netherlands
wsinjh@win.tue.nl

Clifford B. **Jones**
Dept. of Computer Science
University of Manchester
GB-Manchester M13 9PL
cliff@cs.man.ac.uk

Steven **Klusener**
Centrum voor Wiskunde en Informatica

Kruislaan 413
NL-1098 SJ Amsterdam
stevenk@cwi.nl

Joost N. **Kok**
Utrecht University
Dept. of Computer Science
3508 TB Utrecht
The Netherlands
joost@cs.ruu.nl

Ruurd **Kuiper**
Technical University Eindhoven
P.O. Box 513
NL-5600 MB Eindhoven
wsinruurl@win.tue.nl

Reino **Kurki-Suonio**
Tampere University of Technology
P.O. Box 527
SF-33101 Tampere
Finland
rks@cs.tut.fi

Bernard **Le Goff**
BULL Research Center PC62A13
68, route de Versailles
78430 Louveciennes
France
blegoff@crg.bull.fr

Ernst-Rüdiger **Olderog**
Institut für Informatik
Universität Oldenburg
Postfach 25 03
W-2900 Oldenburg
olderog@uniol.uucp

W. **Reisig**
Institut für Informatik
Technische Universität München
Postfach 20 24 20
W-8000 München 2
reisig@lan.informatik.tu-muenchen.dbp.de

Willem-Paul **de Roever**
Institut für Informatik II
Christian-Albrechts-Universität
Preusserstr. 1-9
W-2300 Kiel 1
wpr@informatik.uni-kiel.dbp.de

Arno **Schmitt**
Fachbereich 14 - Informatik
Universität des Saarlandes
W-6600 Saarbrücken 11
arno@cs.uni-sb.de

Philippe **Schnoebelen**
LIFIA - Institut IMAG
46, Avenue Felix Viallet
F-38031 GRENOBLE Cedex
France
phs@lifia.imag.fr

Bernhard **Steffen**
Prof. Dr.
Lehrstuhl für Informatik II
RWTH - Aachen
Ahornstraße 55
W-5100 Aachen
bus@informatik.rwth-aachen.de

Frank A. **Stomp**
Abo Akademi
Dept. of Computer Science
Lemminkäinengatan 14
20520 Turku
Finland
fstomp@ra.abo.fi

Bent **Thomsen**
ECRC
Arabellastr. 17
W-8000 München 81
PD

Shengzong **Zhou**
Fachbereich 14 - Informatik
Universität des Saarlandes
W-6600 Saarbrücken 11
zhou@cs.uni-sb.de

Armin **Zundel**
Inst. für Logik, Deduktionssysteme
und Komplexität
Universität Karlsruhe
W-7500 Karlsruhe 1
zundel@ira.uka.de

Job **Zwiers**
Dept. of Computer Science
University of Twente
7500 AE Enschede
The Netherlands
zwiers@cs.utwente.nl

Bisher erschienene und geplante Titel:

W. Gentzsch, W.J. Paul (editors):
Architecture and Performance, Dagstuhl-Seminar-Report; 1, 18.-20.6.1990; (9025)

K. Harbusch, W. Wahlster (editors):
Tree Adjoining Grammars, 1st. International Worshop on TAGs: Formal Theory and Application, Dagstuhl-Seminar-Report; 2, 15.-17.8.1990 (9033)

Ch. Hankin, R. Wilhelm (editors):
Functional Languages: Optimization for Parallelism, Dagstuhl-Seminar-Report; 3, 3.-7.9.1990 (9036)

H. Alt, E. Welzl (editors):
Algorithmic Geometry, Dagstuhl-Seminar-Report; 4, 8.-12.10.1990 (9041)

J. Berstel , J.E. Pin, W. Thomas (editors):
Automata Theory and Applications in Logic and Complexity, Dagstuhl-Seminar-Report; 5, 14.-18.1.1991 (9103)

B. Becker, Ch. Meinel (editors):
Entwerfen, Prüfen, Testen, Dagstuhl-Seminar-Report; 6, 18.-22.2.1991 (9108)

J. P. Finance, S. Jähnichen, J. Loeckx, M. Wirsing (editors):
Logical Theory for Program Construction, Dagstuhl-Seminar-Report; 7, 25.2.-1.3.1991 (9109)

E. W. Mayr, F. Meyer auf der Heide (editors):
Parallel and Distributed Algorithms, Dagstuhl-Seminar-Report; 8, 4.-8.3.1991 (9110)

M. Broy, P. Deussen, E.-R. Olderog, W.P. de Roever (editors):
Concurrent Systems: Semantics, Specification, and Synthesis, Dagstuhl-Seminar-Report; 9, 11.-15.3.1991 (9111)

K. Apt, K. Indermark, M. Rodriguez-Artalejo (editors):
Integration of Functional and Logic Programming, Dagstuhl-Seminar-Report; 10, 18.-22.3.1991 (9112)

E. Novak, J. Traub, H. Wozniakowski (editors):
Algorithms and Complexity for Continuous Problems, Dagstuhl-Seminar-Report; 11, 15-19.4.1991 (9116)

B. Nebel, C. Peltason, K. v. Luck (editors):
Terminological Logics, Dagstuhl-Seminar-Report; 12, 6.5.-18.5.1991 (9119)

R. Giegerich, S. Graham (editors):
Code Generation - Concepts, Tools, Techniques, Dagstuhl-Seminar-Report; 13, , 20.-24.5.1991 (9121)

M. Karpinski, M. Luby, U. Vazirani (editors):
Randomized Algorithms, Dagstuhl-Seminar-Report; 14, 10.-14.6.1991 (9124)

J. Ch. Freytag, D. Maier, G. Vossen (editors):
Query Processing in Object-Oriented, Complex Object, and Nested Relation Databases, Dagstuhl-Seminar-Report; 15, 17.-21.6.1991 (9125)

M. Droste, Y. Gurevich (editors):
Semantics of Programming Languages and Model Theory, Dagstuhl-Seminar-Report; 16, 24.-28.6.1991 (9126)