Krzysztof Apt, Klaus Indermark,
Mario Rodríguez Artalejo (editors)

Integration of Functional and Logic Programming

Dagstuhl-Seminar-Report; 10
18. - 22.3.1991 (9112)

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: office@dag.uni-sb.de

Dagstuhl-Seminar on

Integration of

Functional and Logic Programming

March 18 – 22, 1991

Organized by

Krzystof Apt (CWI-Amsterdam)

Klaus Indermark (RWTH Aachen)

Mario Rodríguez Artalejo (UC Madrid)

Preface

In recent years the integration of functional and logic programming has become a major research topic. Functional languages permit efficient (parallel) implementations whereas logic languages offer a greater flexibility for problem oriented programming. Therefore, an increasing number of researchers investigates possible combinations of both programming styles in order to achieve a proper balance between the needs of programmers and machines.

The idea of organizing a Dagstuhl seminar on this topic originated from two projects: an Esprit Basic Research Action with K. Apt coordinating the part on integration of functional and logic programming, and an Acción Integrada where research groups of M. Rodríguez Artalejo at the Universidad Complutense Madrid and of K. Indermark at the RWTH Aachen collaborate on the development and implementation of the language BABEL.

The seminar covered the main research directions within the integration field: language design, semantics, logics, type theory, evaluation strategies and implementation techniques. This report contains the abstracts of all talks, as they have been written into the seminar book by the lecturers, and a list of addresses of the invited researchers.

Schloß Dagstuhl offered an excellent environment for carrying out this seminar. Thanks are due to the management and the domestic servants of Schloß Dagstuhl for providing all participants a pleasant stay.

List of Talks

Narrowing as an Incremental Constraint Satisfaction Algorithm

María Alpuente
Universidad Politécnica de Valencia

Moreno Falaschi, Giorgio Levi
Università di Pisa

In this paper we are concerned with an instance of the CLP scheme specialized in solving equations with respect to a Horn equational theory E. The intended structure is given by the finest partition H/E induced by E on the Herbrand Universe H over a finite one sorted alphabet. This work deals with the description of an incremental constraint solver as the kernel of an operational semantics for the language CLP(H/E). The primary issues are: how to verify the solvability of constraints in the structure of H/E by using some sound and complete semantic unification procedure such as narrowing, how to simplify constraints in a computation sequence, how to simplify constraints in the computation process and how to profit from finitely failed derivations as a heuristic for optimizing the algorithms. Fixpoint and model-theoretic semantics are obtained as a straightforward way from the results about Constraint Logic Programming.

Logics with Constraints

Hélène Kirchner, Claude Kirchner, Michaël Rusinowitch
INRIA Lorraine & CRIN

A framework for first order constrained deduction is proposed in this paper. The syntax and semantics of so-called symbolic constraints and of constrained formulas are defined. Constrained deduction rules are given for equational logic, Horn clause logic and first-order logic with equality. They are applied to clausal theorem proving, unfailing completion and completion modulo a set of axioms.

The Meaning of LIFE

Hassan Aït-Kaci
DEC Paris Research Laboratories

Andreas Podelski
LITP, University of Paris-7

LIFE (Logic, Inheritance, Functions, Equations) is an experimental programming language proposing to integrate three paradigms proven useful for symbolic computation. From the programmer's standpoint, it may be perceived as a language taking after logic programming, functional programming, and object-oriented programming. From a

formal perspective, it may be seen as an instance (or rather, a composition of three instances) of a Constraint Logic Programming scheme due to Höhfeld and Smolka refining that of Jaffar and Lassez. We start with an informal overview showing the functionality of LIFE as a programming language, illustrating how the constructs of LIFE offer rather unusual and (pleasantly) startling conveniences. The second part is a formal account of LIFE's object unification seen as constraint-solving over specific domains. We build on work by Smolka and Rounds to develop type-theoretic, logical, and algebraic renditions of a calculus of order-sorted feature approximations. This approximation semantics is shown to be congruent with an operational semantics expressed as a set of complete and consistent syntax-driven non-deterministic constraint normalization rules, including functional beings.

LML: A Functional Meta-Language for Logic Programming

Dino Pedreschi
Università di Pisa

LML (Logical Meta-Language) is a functional language equipped with a data type of theories, whose objects represent logic programs. Theories come with operators for querying them, to obtain sets of answers, and combining them together, to build more complex ones. Thus, from one perspective, LML can be viewed as yet another approach to the integration of functional and logic programming, aiming at amalgamating within a single framework the expressiveness of both paradigms. From another perspective, however, LML may be viewed as a programming language for the construction of knowledge based systems, in which chunks of knowledge are represented as separated logic theories. According to this perspective, the functional layer acts as a meta-level framework for the underlying logic programming component: theories are ordinary data values, which can be manipulated by suitable operators. This accounts for describing the dynamic evolution of theories, and addressing the issue of modularity. In my talk I presented an overview of the language and its type system, together with motivating examples (modular system construction, and reconstruction of default reasoning).

The Functional Logic Language Babel

Mario Rodríguez Artalejo
Univ. Complutense, Madrid

Functional logic languages were proposed by U.S. Reddy as a way to integrate the functional and logic programming paradigms. They have functional syntax, but use narrowing as operational semantics. This execution mechanism includes the usual reduction from functional languages and also covers SLD-resolution, provided that definite Horn clauses "Head ← Body" are viewed as conditional rewrite rules "Head ⇒ true ← body". The talk presented the functional logic language BABEL, which has been worked out in cooperation by research groups in Madrid and Aachen. BABEL is a higher order language with polymorphic types, syntactically similar to SML or Miranda. The execution mechanism is narrowing. Programs allow to define types by declarations and functions by constructor based conditional equations whose boolean condition may include extra variables not occurring in the left hand side. These so called free variables must be first order and are solved by narrowing during execution. Higher order variables are also forbidden in the goals, which are expressions to be narrowed until obtaining a final result in normal form and an answer substitution.
In the talk, these features of the language were explained and illustrated by examples. Some other works in relation to BABEL's implementation and extension were announced for later talks by other participants.

S-SLD-Resolution — An Operational Semantics for Logic Programs with External Procedures

Johan Boye
Univ. of Linköping, Sweden

This paper presents a new operational semantics for logic programs with external procedures, introduced by Bonnier & Maluszyński in ISCLP '88. A new resolution procedure S-SLD-resolution is defined, in which each step of computation is characterized by a goal and a set of equational constraints, whose satisfiability cannot be decided with the information at hand. This approach improves the completeness of the resulting system, since further computation may result in the information needed to solve some earlier unsolved constraints. We also state a sufficient condition to distinguish a class of programs for which no unsolved constraints will remain at the end of computation.

Higher Order Logic Programming with Combinators

Juan Carlos González Moreno
Universidad Politécnica de Madrid

We describe an extension of the first order functional logic language BABEL which supports higher order programming, including higher order logic variables. The operational semantics is based on lazy narrowing and subsumes reduction and SLD-resolution. General higher order unification is avoided and a decidable unification is used instead. By means of a syntactical translation, we relate the operational semantics to the declarative semantics of the first order fragment of the language. This provides soundness and completeness theorems which give an exact characterization of the expected solutions on the higher order level. The class of potential solutions is program dependent. For some programs, it is as powerful as the class of λ-expressions.
In the approach that we describe, we use a restricted form of higher order narrowing, based on a decidable unification concept. General higher order unification, which is known to be undecidable, will be avoided.

BABEL Implementation Based On Lazy Narrowing

Juan José Moreno Navarro
Universidad Politécnica de Madrid

The talk discusses the implementation of lazy narrowing in the framework of a graph reduction machine. By extending an appropiate architecture for purely functional languages an abstract graph narrowing machine for the functional logic language BABEL is constructed. The machine is capable of performing unification and backtracking. The techniques used in functional programming to cope with lazy evaluation are not directly applicable, but must be modified due to the logical component of the implemented language. A prototype implementation of the new machine has been developed. A demonstration of the current available BABEL system was presented.

An Abstract BABEL-Machine Based On Innermost Narrowing

Herbert Kuchen, Rita Loogen
RWTH Aachen

Juan José Moreno Navarro
Universidad Politécnica de Madrid

Mario Rodríguez Artalejo
Universidad Complutense de Madrid

An abstract machine for the functional logic language BABEL is presented. This machine is based on innermost narrowing. It uses a graph to represent the state of computation. Hence, it is a starting point for a parallel implementation, while a stack machine is more efficient for a sequential implementation. The machine was developed by extending an abstract machine for a purely functional language by unification and backtracking mechanisms. Due to a special treatment of deterministic computations a behaviour similar to that of purely fundtional languages can be achieved for (mainly) deterministic programs. Experimental results and comparisons with other languages like PROLOG and SML are promising.

From Reduction Machines To Narrowing Machines

Rita Loogen
RWTH Aachen

Narrowing, the evaluation mechanism of functional logic languages, can be seen as a generalization of reduction, the evaluation mechanism of purely functional languages. The unidirectional pattern matching, which is used for parameter passing in functional languages is simply replaced by the bedirectional unification known from logic programming languages. The talk showed, how to extend a reduction machine, that has been designed for the evaluation of purely functional languages to a machine that performs narrowing. The necessary extensions concern the realization of unification and backtracking. The latter has to be incorporated to handle nondeterministic computations. The narrowing machine enables a space-efficient handling of nested expressions and embodies an optimized treatment of deterministic computations.

Optimizing Narrowing Strategies

Alexander Bockmayr
Max-Planck-Institut Saarbrücken

Stefan Krischer
CRIN Nancy

Rewriting and narrowing provide a nice theoretical framework for the integration of logic and functional programming. For practical applications however, narrowing in its original form is much too inefficient. Therefore many optimizations have been proposed during the last years. In this talk, we consider narrowing for arbitrary canonical term rewrite systems. We don't impose such restrictions as constructor discipline, left-linearity or non-overlapping left-hand sides. In the first part of our talk we present the most important narrowing strategies for this general case that are known today. We give also quantitative results to compare their efficiency. In the second part, we show how the efficiency of narrowing can be further improved by reducibility tests. We introduce a new narrowing strategy, LSE-SL left-to-right basic normal narrowing, sketch the proof of its completeness and illustrate it by various examples.

Type-driven Evaluation for Integrating FP and LP

Staffan Bonnier
University of Linköping, Sweden

We give a formal operational basis for integrating Horn Clause programs with procedures of any typed functional language. The functional procedures are assumed to admit normal form reduction of any term t whose variables play no significant role during reduction. A sufficient condition for the safe reduction of t, formulated in terms of the type of t, is developed. The core of the integration is a unification algorithm which employs this condition in order to unify terms modulo the rules of the functional procedures.

Introducing Sharing into OR-Parallel Implementations of L+F Programming

W. Damm, F. Liu, T. Peikenkamp
FB Informatik, Univ. of Oldenburg

The talk presents a method enhancing OR-parallel execution models by sharing common computation paths between OR-parallel processes and thus avoiding recomputations of solutions which do not depend on bindings established by left sibling processes. In the sequential case, it covers optimizations achieved by intelligent backtracking. The proposed concept of sharing computations is independent of particular aspects of OR-parallel execution model and can thus be combined with optimized implementation schemes for binding arrays or lazy process generation schemes as task stealing. Moreover, it allows a straightforward extension to resolution based implementations of L+F languages such as the language K-LEAF developed at the University of Pisa.

A WAM-Based Implementation of Narrowing and Rewriting

Michael Hanus
Universität Dortmund

We present an efficient implementation method for a language that amalgamates functional and logic programming styles. The operational semantics of the language consists of resolution to solve predicates and narrowing and rewriting to evaluate functional expressions. The implementation is based on an extension of the Warren Abstract Machine (WAM). This extension causes no overhead for pure logic programs and allows the execution of fuctinal programs by narrowing and rewriting with the same efficiency as their relational equivalents. Moreover, there are many cases where functional programs are more efficiently executed than their relational equivalents.

Constraints Unify Functional and Logic Programming — Introducing Constraint Functional Logic Programming

Yi-Ke Guo, Helen Pull
Imperial College, London, UK

To integrate the functional and logic styles of programming, a new declarative programming paradigm, constraint functional logic programming (CFLP), is presented. A CFLP system exhibits all constraint, functional and logic programming features, and unifies them systematically. CFLP extends functional logic programming (FLP), which provides an equation solving capability over a functional program to general constraint programming, and permits the use of constraints for programming as well as for posing queries. CFLP also generalises constraint logic programming (CLP) by admitting user-defined functions as a purely functional subsystem of a CLP language. The resulting system has a concise semantic foundation. Various computational models such as reduction for evaluating expressions, narrowing for solving equations and resolution for deducing logical consequences are combined via constraint solvers. This paper presents the motivation for designing CFLP, and presents its semantics. A CFLP language, Falcon, (Functional And Logic language with CONstraints) is then proposed to give a concrete representation of the framework. The expressive power of the CFLP paradigm and its programming style are illustrated by some example programs in Falcon.

Mastering Recursive Schemes by Weak Combinatory Logic

Corrado Böhm
Univ. di Roma, "La Sapienza"

The use of fixed point combinators to solve recursive equation systems, although ubiquitous, has as fundamental disadvantage the loss of strong normalization property. We show that such a disadvantage vanishes if:

1. We implement term algebras by a weak combinatory logic reduction machine, which is also useful to minimize the compiling and coding work.

2. We solve any recursive equation system by exhibiting combinators in normal form, which represent the (higher order) functions solving the systems.

Given any set of recursive scheme defining a set of functions the method consists of finding the set of minimum common recursive schemes (one for each domain algebra) such that each scheme needed for the definition of any function is an instance of an element of the above set. Then the problem of representing every function is reduced to the easier problem to find the combinators representing the constructors of each domain algebra.

Type Inference in Polymorphic Type Discipline

Simona Ronchi Della Rocha
Università di Torino (Italy)

We study the problem of the automatic inference of types in the polymorphic type discipline for lambda calculus (PTD). PTD is an extension of Classical Curry's functionality theory, in which types can be universally quantified. In the PTD, it can be shown that a principal type in the sense of Curry does not exist. Moreover, the problem of whether the set of terms having a type in this discipline is recursive is still open; this set is a paper subject of the set of strongly normalizing terms. We define a countable set of type assignment systems t_n ($n \in \omega$), which is a complete stratification of the polymorphic type assignment system. Every system t_n is decidable. Then we build a type inference algorithm for every system t_n. This algorithm generates for any untyped term M and integer n, the (finite) set of the minimal types derivable for M in t_n, if any. The set is minimal with respect to a partial order relation defined in such a way that, if M can be typed in Curry's type discipline, then for each n its (principal) Curry's type is the minimum of the relation. In fact we are looking for a conservative extension of the ML type inference algorithm (which is based on Curry's types).
A type inference problem is always connected to an unification problem. The type inference algorithm for Curry's types is based on the classical first-order unification procedure of Robinson. For solving the type inference problem in polymorphic type discipline, we are dealing with particular instances of the semi-unification problem. This problem has been proved to be undecidable by Kfoury et al. We give a parameterized solution for a restricted class of instances of semiunification problems, containing all the instances of typing problems in PTD. The technical tool is a unification procedure, which is a conservative extension of the classical one, but it never gives a failure. A situation that classically is interpreted as a failure here gives a positive information, namely the need of a quantified variable in the place in which the failure occurred. In order to do this, information about the structure of the terms to be unified must be also supplied to the algorithm. The defined stratification of PTD is not only a technical tool for the type inference, but it seems to be interesting in itself. t_0 coincides with Curry's type assignment system, and in t_1 all the normal forms can be typed.

Compiling Lazy Narrowing into Prolog

Cheong P. H.
LIENS Paris

Lazy narrowing is a complete refutation procedure for K-LEAF, a first-order logic + functional language. Lazy narrowing is simulated by first putting clauses and goals into flat form and then interpreting the flat form by outermost SLD-resolution. In this talk, we show how K-LEAF can be directly compiled into Prolog. More general the method suggests an efficient compilation of lazy narrowing into Prolog.

Uniform Narrowing Strategies

Rachid Echahed
Lifia-Imag, Grenoble, France

We focus on the computational aspects of programming languages with initial semantics, which are based on Horn clause logic with equality. We discuss first the "right" definitions of soundness and completeness that should be considered for such languages. Then, we propose SLDIE-resolution as a unique inference rule which is strongly complete. This rule is parameterized by a complete algorithm of resolution of equations in the initial model. We investigate algorithms of resolution of equations based on narrowing. We introduce the notion of strategies that ensure the completeness of narrowing algorithms. Then, we define a class of programs called "uniform programs" for which any strategy is complete, and prove the decidability of this class of programs. At last, we give a complete set of inference rules that results from the integration of SLD-resolution and narrowing with strategies.

Guarded Functional Programming, Non-Determinism, and Laziness

Roland Dietrich
GMD Karlsruhe

Guarded Term ML is a language which integrates functional programming, represented by equations and rewriting, and logic programming, represented by Horn clauses and

SLD-resolution: the selection of a guarded equation for rewriting expression is determined by pattern matching with the left hand side and solving the guard which is a Horn logic goal. Only one solution of the guard is considered ("committed choice"). Stream comprehensions evaluate to a stream of expressions which are achieved when applying all answer substitution of a goal to an expression. These streams can be consumed in functional expressions. Lazy evaluation of streams enable to deal with infinite streams and avoid unnecessary computations. With streams, non-determinism and backtracking which is inherent in the logic programming world can be exploited in the functional world. Because functional expressions are not allowed for guarded or Horn clauses, both worlds are kept strongly separate.
Besides an introduction to the basic concepts of Guarded Term ML, the talk gives examples how to use it to program different kinds of mathematical objects as pure functions, non-deterministic functions, set-valued functions and pure relations.

A General Approach to the Implementation of Functional Logic Languages

Hendrik C. R. Lock
GMD Karlsruhe

It has become apparent that SLD-resolution, narrowing and reduction are the essential operational models of an interesting large class of functional logic languages. The talk discussed a unifying approach to their implementation by means of abstract machines. In part one, the features of operational models were identified, and further, the basic features of abstract machines were introduced. Then, it was shown which machine feature supports which operational feature. Machine features can be composed freely thus obtaining new, more powerful abstract machines. By means of the relation between operational models and their features, and by the support relation between those and basic machine features, a design space is obtained. The design space specifies ways of combining machines (in terms of required features) in order to obtain an implementation of a particular operational model. Furthermore, it was discussed that the design of an abstract machine is also a compromise towards the requirements imposed by the target level and by cock generation. So, a design should take that into account. In order to be able to apply the well-understood conventional compilation techniques we pointed out that the instruction code should be block-structured, allow scoping, contain no labels nor gotos, and that values are denoted by intermediate variables.
In the second part of this talk an abstract machine has been presented which is an instance of the top mode in that design space, called the JUMP machine. In particular, it is based on a conventional stack based architecture which is extended by all required features, such that lazy narrowing, for instance, can be implemented. As a particular result, when restricting evaluation to ground term reduction, the machine just behaves

like an efficient stack machine for graph reduction, and can be seen as a variant of the "Spineless Tagless G-Machine". A compiler for the functional logic language GTML will be available in April 1991.

K-LEAF - A Logic plus Functional Language

Catuscia Palamidessi
Centre for Mathematics and Computer Science
Amsterdam, The Netherlands
and
Department of Computer Science
Utrecht University, The Netherlands

The language K-LEAF (Kernel language for Logic, Equations, And Functions) was developed in the context of the ESPRIT Project 415 (Parallel Architectures and Languages for Advanced Information Processing) as a declarative (first-order) approach to parallel and distributed programming.
The basic aims in the design of K-LEAF are

- combining in a unique language the two main declarative paradigms: the logic and the functional ones, and
- allowing the specification of interactive (possibly nonterminating) processes.

At the syntactical level, the integration of predicates and (first-order) functions is achieved in K-LEAF by expressing the functional part as a conditional equational system, so embedding it into the Horn clause formalism. The operational semantics is based on the flattening technique, which allows to deal with equality just as any other user-defined predicate symbol, and to use SLD-Resolution as the sole computational mechanism.
In order to express nonterminating processes, K-LEAF allows the underlying term rewriting system (associated to the equational part) to be noncanonical. (This was in contrast with most of the other logic plus functional proposals of the same period.) The difficulties that arise concerning effectiveness, due to the presence of infinite (and partial) functions, are dealt with by means of a distinction between two kinds of equality, and by imposing restrictions on their use in the bodies and in the queries.
The language has a model-theoretic semantics based on algebraic CPO's. The two kinds of equality are semantically characterized by their behaviour w.r.t. continuity, and the syntactical restrictions on the language are shown to be the minimal conditions necessary for the least Herbrand model to be effective, i.e. for the operational semantics (SLD-Resolution) to be complete.
K-LEAF was used as a Kernel language to implement IDEAL, a logic plus functional formalism which also includes higher-order features. The language was provided with primitives to express concurrency and it was implemented on an Extended WAM.

Participants 9112:

Hassan **Ait-Kaci**
DEC Paris Research Laboratory
85 Avenue Victor Hugo
92563 Rueil-Malmaison Cedex, France
hak@prl.dec.com

Maria **Alpuente**
Universidad Politecnica Valencia
Depart. Sistenas Informatios y Computación
Camino de Vera s/n,Apdo. 22012
46020 Valencia, Spain
maria@dsic.upv.es

Krzysztof R. **Apt**
CWI
Kruislaan 413
NL - 1098 SJ Amsterdam, The Netherlands
apt@cwi.nl

Marc **Bezem**
State University of Utrecht
Dept. of Philosophy
P.O.Box 80103
NL-3508TC Utrecht, The Netherlands
bezem@phil.ruu.nl

Alexander **Bockmayr**
Max-Planck-Institut für Informatik
Im Stadtwald
W-6600 Saarbrücken 11, Germany
bockmayr@cs.uni-sb.de

Corrado **Böhm**
Universitá degli Studi "La Sapienza"
Dipartimento di Matematica
Piazzale Aldo Moro 2
00185 Roma, Italy
bohm@astrom.infn.it /
boehm@vaxrma.infn.it

Staffan **Bonnier**
Linkoeping University
Dept. of Computer & Information Science
581 83 Linkoeping, Sweden
sbo@ida.liu.se

Johan **Boye**
Linkoeping University
Dept. of Computer & Information Science
581 83 Linkoeping, Sweden
jombo@ida.liu.se

P.H. **Cheong**
CNRS, URA 1327
L.I.E.N.S.
45 rue d'Ulm
75230 Paris Cedex 05, France
cheong@dmi.ens.fr

Werner **Damm**
Universität Oldenburg
FB Informatik
Postfach 25 03
W-2900 Oldenburg, Germany
damm@uniol.uucp

Roland **Dietrich**
GMD Forschungsstelle an der
Universität Karlsruhe
Vincenz-Prießnitz-Str. 1
W-7500 Karlsruhe 1, Germany
dietrich@karlsruhe.gmd.dbp.de

Rachid **Echahed**
LIFIA - INPG
Institut National Polytechnique de Grenoble
46, avenue Félix Viallet
38031 Grenoble, France
echahed@lifia.imag.fr

Moreno **Falaschi**
Universita di Pisa
Dipartimento di Informatica
Corso Italia 40
56100 Pisa, Italy
falaschi@di.unipi.it

Laura **Ferrari,** c/o Bosco P. Giorgio
CSELT
Via Reiss Romoli 274
10148 Torino, Italy

Juan Carlos **Gonzalez Moreno**
Universidad Politecnica de Madrid
Facultad de Informatica
Campus de Montegancedo s/n
Boadilla del Monte
28660 Madrid, Spain
jgonzale@datsi.upm.es / jcmoreno@fi.upm.es

Yike K. **Guo**
Department of Computing
Imperial College of Science and Technology
180 Queen's Gate
London SW7 2BZ, Great Britain
yg@doc.imperial.ac.uk

Michael **Hanus**
Fachbereich Informatik
Universität Dortmund

W-4600 Dortmund 50, Germany
michael@ls5.informatik.uni-dortmund.de

Klaus **Indermark**
RWTH Aachen
Lehrstuhl für Informatik II
Ahornstr. 55
W-5100 Aachen, Germany
indermark@informatik.rwth-aachen.de

Hélène **Kirchner**
Centre de Recherche en Informatique de Nancy
Boite Postale 239
54506 Vandoeuvre-lès-Nancy, France
hkirchne@loria.crin.fr

Herbert **Kuchen**
RWTH Aachen
Lehrstuhl für Informatik II
Ahornstr. 55
W-5100 Aachen, Germany
kuchen@informatik.rwth-aachen.de

Feixiong **Liu**
Universität Oldenburg
FB. Informatik
Postfach 2503
W-2900 Oldenburg, Germany
liu@uniol.uucp

Hendrik C.R. **Lock**
GMD Forschungsstelle an der
Universität Karlsruhe
Vincenz Prießnitz Str. 1
W-7500 Karlsruhe 1, Germany
lock@gmdka.uucp

Rita **Loogen**
RWTH Aachen
Lehrstuhl für Informatik II
Ahornstr. 55
W-5100 Aachen, Germany
rita@zeus.informatik.rwth-aachen.de

Francisco Javier **Lopez-Fraguas**
Universidad Politecnica de Madrid
Dep. Matematica Aplicada
Escuela Universitaria de Informatica
Carretera de Valencia Km. 7
E-28031 Madrid,Spain

Jan **Maluszynski**
Linkoeping University
Dept. of Computer & Information Science
581 83 Linkoeping, Sweden
jmz@ida.liu.se

Aart **Middeldorp**
CWI, dept. AT
Kruislaan 413
NL - 1098 SJ Amsterdam, The Netherlands
ami@cwi.nl

Juan José **Moreno Navarro**
Universidad Politécnica de Madrid
LSIIS, Facultad de Informática
Campus de Montegancedo s/n
Boadilla del Monte
28660 Madrid, Spain
jjmoreno@datsi.upm.es, jjmoreno@fi.upm.es

Catuscia **Palamidessi**
Universita di Pisa
Dipartimento di Informatica
Corso Italia 40
56125 Pisa, Italy
katuscia@dipisa.di.unipi.it

Dino **Pedreschi**
Universita di Pisa
Dipartimento di Informatica
Corso Italia 40
56125 Pisa, Italy
pedre@dipisa.di.unipi.it

Thomas **Peikenkamp**
Universität Oldenburg
FB Informatik
Postfach 25 03
W-2900 Oldenburg, Germany
Thomas.Peikenkamp@arbi.informatik.uni-oldenburg.de

Helen **Pull**
Imperial College
Dept. of Computer Science
Queens Building
London SW7 2BZ, England
hmp@doc.imperial.ac.uk

Mario **Rodríguez Artalejo**
Universidad Complutense
Dep. Informática y Automática
Facultad de Matemáticas
28040 Madrid, Spain
W450@EMDUCM11.BITNET

Simonetta **Ronchi Della Rocca**
Universita di Torino
Dipartimento di Informatica
Corso Svizzera 185
10149 Torino, Italy
ronchi@di.unito.it

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)

G. Farin, H. Hagen, H. Noltemeier (editors):
Geometric Modelling, Dagstuhl-Seminar-Report; 17, 1.-5.7.1991 (9127)

A. Karshmer, J. Nehmer (editors):
Operating Systems of the 1990s, Dagstuhl-Seminar-Report; 18, 8.-12.7.1991 (9128)

H. Hagen, H. Müller, G.M. Nielson (editors):
Scientific Visualization, Dagstuhl-Seminar-Report; 19, 26.8.-30.8.91 (9135)

T. Lengauer, R. Möhring, B. Preas (editors):
Theory and Practice of Physical Design of VLSI Systems, Dagstuhl-Seminar-Report; 20, 2.9.-6.9.91 (9136)

F. Bancilhon, P. Lockemann, D. Tsichritzis (editors):
Directions of Future Database Research, Dagstuhl-Seminar-Report; 21, 9.9.-13.9.91 (9137)

H. Alt , B. Chazelle, E. Welzl (editors):
Computational Geometry, Dagstuhl-Seminar-Report; 22, 07.10.-11.10.91 (9141)

F.J. Brandenburg , J. Berstel, D. Wotschke (editors):
Trends and Applications in Formal Language Theory, Dagstuhl-Seminar-Report; 23,14.10.-18.10.91 (9142)

H. Comon , H. Ganzinger, C. Kirchner, H. Kirchner, J.-L. Lassez , G. Smolka (editors):
Theorem Proving and Logic Programming with Constraints, Dagstuhl-Seminar-Report; 24, 21.10.-25.10.91 (9143)

H. Noltemeier, T. Ottmann, D. Wood (editors):
Data Structures, Dagstuhl-Seminar-Report; 25, 4.11.-8.11.91 (9145)

A. Borodin, A. Dress, M. Karpinski (editors):
Efficient Interpolation Algorithms, Dagstuhl-Seminar-Report; 26, 2.-6.12.91 (9149)

B. Buchberger, J. Davenport, F. Schwarz (editors):
Algorithms of Computeralgebra, Dagstuhl-Seminar-Report; 27, 16..-20.12.91 (9151)