Hans Langmaack, Erich Neuhold,
Manfred Paul (editors):

Software Construction - Foundation and Application

Dagstuhl-Seminar-Report; 29
13.-17.1.92 (9203)

ISSN 0940-1121

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

Das Internationale 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

Software Construction – Foundation and Application

Organized by:

Hans Langmaack (Christian-Albrechts-Universität zu Kiel)
Erich Neuhold (GMD-IPSI, Darmstadt)
Manfred Paul (Technische Universität München)

Schloß Dagstuhl, January 13 - 17, 1992

Contents

1 Preface

This seminar in Dagstuhl was the continuation of a series which the organizers have held biannually for a number of years under the title *Mathematical Tools for the Construction of Software* at the Mathematical Research Institute Oberwolfach. One essential intention therewith was, among others, to provide a forum for scientists interested and active in the areas covered by **IFIP/TC 2** in order to allow interaction with the Working Groups of that International Technical Committee. A reason for this intention was that by IFIP rules the direct co-operation with its Working Groups is restricted to their members whereas the seminars in Oberwolfach just as now in Dagstuhl are certainly not.

The seminar was the first of the series mentioned above held in Dagstuhl. Its title *Software Construction – Foundation and Application* is a modification of the former intended to emphasize that applications and foundations of programming should receive equal attention in this seminar. It had 35 participants all involving themselves very actively. 32 presentations were given leading to lively and fruitful discussions. The presentations can roughly be categorized along the following streams:

- *Proving and verifying programs*
- *Semantics of programming concepts*
- *Concurrent programs and non-deterministic constructs*
- *Compiling and developing software*
- *Logic for programming*
- *Interactive programming*

The discussions both during the sessions and in the evenings have shown that the talks given were well received and have met great interest. One focus of broader interest was for instance the specification of programs and, speaking more generally, of systems. In this context the refinement of interactive systems and type theory played a particular role. Other topics which have triggered engaged discussions were the handling of non-determinism and concurrency in connection with particular language concepts including but not restricted to CSP and OCCAM. Moreover the transformation and the compilation of programs through interpreters and other means like e.g. data flow analysis have led to activities among the participants. Last not least some well known specialists in the field of the logical foundations of computing have given very stimulating talks on specific topics.

In summary the participants agreed to have visited a seminar which was worth attending with respect to both scientific communications and personal contacts, the latter having been supported not the least by the traditional hike on one beautyful afternoon and evening.

The organizers wish to thank all who have helped to make the seminar a success. First we are grateful to the participants and speakers for their presentations and contributions

during the discussions. Thanks go also to the people of the *International Meeting and Research Center Dagstuhl* and, particularly, to its secretaries who have been a big help not only in the preparatory phase but also on every day during the whole week of the seminar. Further thanks go to the director and to the council of the research center. The participants were all very impressed about what had been accomplished already during the relatively short period since the center has begun its activities. We only mention here the computing and communication facilities, and the collection of scientific books and journals which forms a good basis for a computing science library to be built upon. Finally, we are very grateful to Jens Knoop for editing this report so professionally.

H. Langmaack **E. Neuhold** **M. Paul**

2 Final Seminar Programme

Monday, January 13, 1992

8:50 Opening Remarks
M. Paul, Germany

Session 1, 9:00 – 12:00

Chair: H. Langmaack

9:00 *Optimal Interprocedural Data Flow Analysis*
J. Knoop, Germany

9:40 *Strictness Control in Term Rewriting*
O.-J. Dahl, Norway

10:20 BREAK

10:40 *SADT-Diagrams as Interface for Reuseable Software Components*
P. Forbrig, Germany

11:20 *Tools and Methods to create Simulators / C++ - Prototypes for the Specification Languages SDL/OSDL and LOTOS*
A. Prinz, Germany

Session 2, 15:30 – 17:45

Chair: M. Paul

15:30 *On the Specification of Files and Coroutines, Part I*
A. Salwicki, France

16:10 *On the Specification of Files and Coroutines, Part II*
G. Mirkowska, France

16:50 BREAK

17:05 *Relational Specification of Data Types and Programs*
R. Berghammer, Germany

Tuesday, January 14, 1992

Session 3, 9:00 – 12:00

Chair: A. Salwicki

9:00 *Bisimulation Semantics for Concurrency with Atomicity and Refinement*
J. W. de Bakker, The Netherlands

9:40 *Semantic Domains with Congruences*
G. Schmidt, Germany

10:20 BREAK

10:40 *Distinguishing Divergence from Chaos in CSP*
B. v. Karger, Germany

11:20 *Logic Based Program Development and Tactical Theorem Proving*
W. Stephan, Germany

Session 4, 15:30 – 17:45

Chair: O.-J. Dahl

15:30 *About Timed and Hybrid Systems*
A. Pnueli, Israel

16:10 *Specifying and Verifying Embedded Real-Time Systems*
A. P. Ravn, Denmark

16:50 BREAK

17:05 *Concurrent Temporal Logic*
W. Reisig, Germany

Wednesday, January 15, 1992

Session 5, 9:00 – 12:00

Chair: J. W. de Bakker

9:00 *Compiling Software directly into Hardware*
J. Bowen, UK

9:40 *Normal Form Approach to FPGA Implementation of occam*
He Jifeng, UK

10:20 BREAK

10:40 *Transformational Development of Digital Circuits*
W. Dosch, Germany

11:20 *Specifications Using the Data Type Syntactic Nodes*
V. Levin, Russia

*** AFTERNOON EXCURSION ***

Thursday, January 16, 1992

Session 6, 9:00 – 12:00

Chair: A. Pnueli

9:00 *A Note on Compositional Refinement*
W.-P. de Roever, Germany

9:40 *Composition and Refinement of Functional System Specifications*
M. Broy, Germany

10:20 BREAK

10:40 *Exploiting Concurrency for Linear Time Model Checking of Petri Nets*
H. Schlingloff, Germany

11:20 *A Generator of Formulae Manipulating Editors*
St. Jähnichen, Germany

Session 7, 15:00 – 18:00

Chair: He Jifeng

15:00 *Computational Completeness of Specification and Programming Languages over Various Data Structures*
N. Nikitchenko, Ukraine

15:40 *What is lost in Program Proving due to Clarke's Theorem?*
M. Grabowski, Poland

16:20 BREAK

16:40 *The Complexity of Verifying Functional Programs*
H. Hungar, Germany

17:20 *On-Line Garbage Collection Algorithms*
A. Kreczmar, Poland

Friday, January 17, 1992

Session 8, 9:00 – 11:30

Chair: G. Goos

9:00 *Bridging Existing Gaps between Theory Based Software Development and Practice*
P. E. Lauer, Canada

9:40 *Toward Formal Development of Programs from Algebraic Specifications: Parameterisation Revisited*
D. Sannella, UK

10:20 BREAK

10:50 *Specification based on Type Theory*
F. W. v. Henke, Germany

Session 9, 15:30 – 17:45

Chair: W.-P. de Roever

15:30 *Interactive Programming with Higher-Order Objects, Part I*
G. Schmidt, Germany

16:10 *Interactive Programming with Higher-Order Objects, Part II*
R. Berghammer, Germany

16:50 BREAK

17:05 *On Programming Languages Interpreters – Total Correctness*
H. Langmaack, Germany

3 Abstracts of Presentations

The following abstracts appear in alphabetical order of speakers.

Bisimulation Semantics for Concurrency with Atomicity and Action Refinement

Jaco W. de Bakker[1]
CWI & Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

A comparative semantic study is made of two notions in concurrency, viz. atomicity and action refinement. Parallel composition is modeled by interleaving, and refinement is taken in the version where actions are refined by atomized statements. The bisimulation domain used in the semantic definitions is obtained as solution of a system of domain equations over complete metric spaces. Both operational and denotational models are developed, and their equivalence is established using higher-order techniques and Banach's fixed point theorem. The operational semantics for refinement is based on transition rules rather than on some form of syntactic substitution.

Relational Specification of Data Types and Programs

Rudolf Berghammer
Universität der Bundeswehr München
München, Germany

In the last years the relational calculus of Tarski has widely been used by computer scientists to describe the semantics of programming languages (including the domains necessary), even in the presence of nondeterministic or higher-order constructs.

In this talk, abstract relation algebra is proposed as a practical means for specification of data types and programs. After a short introduction into abstract relation algebra, we define the concept of a relational specification by transferring some fundamental notions of the algebraic specification approach to the relational case. Then we demonstrate the usefulness of the relational approach and give an impression of relational calculations in the field of specifications by means of two examples.

In the first example we specify the natural numbers with zero and the successor operation. With the help of this specification we show a typical pattern for a relational proof of monomorphy. Furthermore, we demonstrate that in the relational case it is possible to prove totality by computational induction if the generation-principle is described as a fixed point property.

[1]This is joint work with E. P. de Vink of the Vrije Universiteit Amsterdam, Amsterdam, The Netherlands. The research of J. W. de Bakker was partially supported by ESPRIT Basic Research Action 3020: Integration.

In the second example we specify relationally the problem of computing a kernel of a Noetherian directed graph. Using a fixed point theorem for antitone mappings, then we transform this specification into another one which has the same class of models but – due to the specific form of the axioms – immediately provides an algorithmic solution of the posed problem in the form of a functional iteration.

Interactive Programming with Higher Order Objects, Part II[2]

Rudolf Berghammer
Universität der Bundeswehr München
München, Germany

In this second part of the talk, the use of abstract relation algebra to describe the semantics of the HOPS language is demonstrated.

Firstly, we give a relational description of most of the domains necessary in denotational semantics. We deal with the direct product of domains, the direct sum of domains, and the domain of partial functions. This latter domain is introduced as a specific subdomain of the domain of all relations. Its definition uses the symmetric quotient and the right residual of two relations as auxiliary constructs. Due to the relational approach we are able to distinguish between an "applicable" function (which is a functional relation) and its corresponding element in the function space (which is a point in the relational sense).

Having given a relational description of domains, we demonstrate by means of some language constructs how to define a relational semantics for the HOPS language. For instance, in the case of a function abstraction $\lambda x.t$ we have that the semantics describes exactly the transition of the semantics of t to the corresponding element in the function domain.

Compiling Software Directly into Hardware

Jonathan Bowen[3]
Oxford University Computing Laboratory
Oxford, UK

The emergence of *Field Programmable Gate Arrays* (FPGAs) has enabled the reprogramming of digital circuitry to be undertaken as easily as computer programs are changed today. A sequence of bits (typically held in a static RAM) defines the wiring of a digital

[2]See also *"Interactive Programming with Higher Order Objects, Part I"* by Gunther Schmidt.

[3]The work was undertaken as part of the European ESPRIT Basic Research Action **ProCoS** ("Provably Correct Systems") project (BRA 3104) and the UK Information Engineering Directorate **safemos** project (IED3/1/1036). Their financial support is gratefully acknowledged.

circuit in such devices in much the same way that a sequence of bits can define the instructions for a general purpose computer. In particular, highly parallel algorithms mapped onto such an architecture can be greatly speeded up using the natural concurrency of hardware.

Currently software support is limited, as it was for early computers. In the same way that most programmers have moved from assembler to high-level languages for all but the most time-critical applications, it is likely that in the future most hardware will be produced using more abstract notions than the individual low-level components available to fabricate them. It is foreseen that high-level languages will be routinely used to describe the design of the hardware and these descriptions will be compiled (often fully automatically) into a low-level description of the hardware (e.g., a *netlist* of components and their interconnections). At Oxford, a prototype has been developed in SML to compile an occam program to a netlist for a synchronous circuit.

Other work at Oxford has concentrated on proving software compilation correct. The proof is considerably simplified by defining the target machine as an interpreter in the high-level language that is being compiled. This allows the proof to be largely conducted using algebraic laws about the high-level language. The compiling scheme for each construct in the language is formulated as a theorem that must be proved correct with respect to a refinement ordering using these laws. The theorems are normally in the form of Horn clauses, and thus may be translated almost directly into a logic programming language such as Prolog. The same technique has been adapted to a hardware compiler by defining the circuit as a simulator written in the high-level language.

Specification and Refinement of Interactive Systems

Manfred Broy
Technische Universität München
München, Germany

A formal framework for the functional property-oriented specification of components of interactive distributed systems and their refinement is given. The refinement supports the change of both the syntactic interface (the number and types of channels) and the behavior (the granularity of messages).

Strictness Control in Term Rewriting

Ole-Johan Dahl
Universitetet i Oslo
Oslo, Norway

The concept of function definition by *Terminating Generator Induction* (TGI) over an identified generator basis was explained. Case constructs with respect to generators give rise to convergent sets of rewrite rules. Termination can be ensured by simple syntactic

checks which permit the definition of a large class of total functions. Partial functions are needed and may be defined using typed, ill-defined constants $\perp_T$ for each type T. Alternatively, $\perp_T$ may be seen as a generator extending the generator basis of T, giving an extended type $T_\perp$. Strictness can now be explicitly controlled, e.g. for defining symmetrical, non-strict Boolean$_\perp$ connectives. For user convenience it is better to have the treatment of ill-defined arguments behind the scenes, e.g. according to "case-semantics", where case constructs are strict in the discriminand and generators are strict, $g(..., \perp, ...) == \perp$. Problem: Generator strictness implies that $T_\perp$ is not freely generated, even if T is, in the case that T has non-constant generators; and the added rewrite rules in general cause the loss of confluence. It is shown that naive term rewriting is nevertheless strongly correct w.r.t. case semantics when applied to well-defined terms. A general construction is shown providing a convergent set of rewrite rules implementing case semantics, as well as any stronger strictness requirement.

Transformational Development of Digital Circuits

Walter Dosch
Universität Augsburg
Augsburg, Germany

A computer program is the implementation of an algorithm in terms of the constructs provided by a programming language. We view a digital circuit as the implementation of an algorithm where the primitive elements are electronic devices.

With this analogy in mind, we derive circuit descriptions for a family of binary adders (ripple-carry adder, serial adder, carry look-ahead adder, completion recognition adder) from the common specification of their functional behaviour. The design follows the methodology of transformational programming. The circuit descriptions are stepwise deduced from the specification by applying correctness preserving transformation rules. The formal derivation of trustworthy hardware gives insight into the algorithmic principles underlying digital circuits. It also disentangles the design decisions concerning the representation of numbers by digit sequences, the use of fixed word length, and the effects of binary coding.

Recursion turns out to be a fundamental concept in designing regularly structured combinational circuits by cascading some elementary circuitry. In the special case of tail recursion, the repetition can be transformed from space to time. Then the electronic devices are reused in different incarnations while keeping the entire network static.

SADT-Diagrams as Interface for Reuseable Software Components

Peter Forbrig
Universität Rostock
Rostock, Germany

The diagrams of the *Structured Analysis and Design Technique* (SADT) became accepted documents in software engineering. The development of such diagrams is up to now a very time consuming task. Often there is a gap between SADT-diagrams and their implementation.

A transformation of SADT-diagrams into attributed grammar rules is described. These rules allow the incorporation of comments and semantics into the specification. It is also shown how this internal representation can be used to check a software specification, to get a prototype, to generate code and to reuse already specified software components. Especially the reuse of already developed projects increases the productivity of software engineering very much.

What is lost in Program Proving due to Clarke's Theorem?

Michał Grabowski
University of Warsaw
Warsaw, Poland

We briefly list the achievements of program verification theory, originated by Clarke's theorem. On the other hand, this theorem has diminished interest in a deep proof-theoretical study of programming languages with the halting problem of programs undecidable in finite interpretations. This is natural, since we have no completeness concept for partial correctness logics of programs of such languages. (The relative completeness principle is not applicable in such cases, as Clarke's theorem says.) We propose another completeness concept capable of handling programming languages with the halting problem undecidable in finite interpretations. The proposal is rather unsatisfactory, but we want to point out the need for completeness concepts for such languages.

Normal Form Approach to FPGA Implementation of occam

He Jifeng[4]
Oxford University Computing Laboratory
Oxford, UK

This paper shows how to compile a program written in a subset of **occam** into *Field-Programmable Gate Arrays* (FPGA), which has recently enabled the possibility of digital circuits to be directly reprogrammable almost as easily as programs are changed routinely today. A simple state-machine model is adopted for specifying the behaviour of a

[4]This is joint work with Ian Page and Jonathan Bowen of Oxford University. The work was undertaken as part of the European ESPRIT Basic Research Action **ProCoS** ("Provably Correct Systems") project (BRA 3104) and the UK Information Engineering Directorate **safemos** project (IED3/1/1036). Their financial support is gratefully acknowledged.

synchronous circuit where it is assumed that the longest time delay in the combinatorial circuitry is shorter than the length of a clock cycle, and the observable includes the state of the control path and the data path of the circuit. We identify the behaviour of a circuit with a program consisting of a very restricted subset of occam, which is called *normal form*. The algebraic laws of occam are used to facilitate the transformation from a program into a normal form. The paper also suggests that the simulation technique used in data refinement can be used to implement the synchronised communication on the shared-state concurrency.

Specification based on Type Theory

Friedrich W. von Henke
Universität Ulm
Ulm, Germany

We consider a simple functional language with a type system providing abstract (recursive) data types, higher-order, polymorphic and dependent types, and constrained types, i.e. semantic subtypes. In a sequence of simple examples we show how typical specification constructs, including specifications of parameterized modules, can be modelled in this framework. The role of subtypes as a unifying concept is elaborated.

The Complexity of Verifying Functional Programs

Hardi Hungar
Universität Oldenburg
Oldenburg, Germany

Since long Hoare-style proof systems are known as tools to prove the correctness of programs. One property every proof system should have is *soundness*. Another one which is not that easy to achieve is *relative completeness* [Cook]. This means completeness relative to the first-order theory of the data domain provided that in the given domain, first-order formulas can define all relations resp. functions which are program definable.

If a proof system is sound and relatively complete, it gives rise to a complete proof procedure for finite domains. I.e.: Given a correctness formula f and the representation of a finite domain D, a derivation of f (a formal proof) from the theory of D can be constructed if f is valid in D.

This observation leads to another criterion measuring the quality of proof systems: How hard is it (in terms of computational complexity) to construct the derivation? Is it much harder than to prove the validity of the formula in some other way or is it of the same degree of complexity?

Here, a positive answer is given for some instances of this problem: In many cases, proof construction can be done efficiently. As a model for functional programming languages, we considered the finitely typed lambda calculus with recursion over some data

domain. We take Goerdt's assertion language and his proof system. If k is a bound on the level of the types occuring in the program, the set of finite domains where a formula about the program is valid can be decided in $(k-1)$–exponential time in the size of the data domain. In general, one can not do better. We show that formal proofs in Goerdt's system can be constructed in $(k-1)$–exponential time, too. This extends previous results on efficient proof construction (for imperative programs).

A Generator for Formulae Manipulating Editors

Stefan Jähnichen[5]
Technische Universität Berlin
Berlin, Germany

The talk describes a tool G^2F to support the use of mathematics based methods on modern computers. Such methods use two-dimensional notations frequently which are hard to edit and to manipulate on a screen if not supported by graphical means. In order to adopt the proposed style of formula manipulation to various applications, a generator was developed which takes grammars and prototype drawings of the symbols as input and produces specific editors and layout procedures as output. Besides the uniform manipulation style, the main advantage is the uniform appearance of formulas even if produced by other tools and the uniform interaction mechanism. As an extension, G^2F produces the layout descriptions as attribute grammar skeletons or logic specifications. This could be viewed as a prototype specification for graphical structures to be further refined textually.

Distinguishing Divergence from Chaos – The Semantics of Failing Computations in CSP

Burghard von Karger[6]
Christian-Albrechts-Universität zu Kiel
Kiel, Germany

A specification-oriented model for state-based CSP is presented that does not identify internal divergence with chaotic behaviour. Consistency with operational semantics is proved even in the case where unbounded nondeterminism is present. Recursion is handled by a new fixpoint construction.

---

[5]This is joint work with R. Gabriel.

[6]Work conducted within the frame of the ESPRIT-BRA-Project 3104 **ProCoS** "Provably Correct Systems".

Optimal Interprocedural Data Flow Analysis

Jens Knoop[7]
Christian-Albrechts-Universität zu Kiel
Kiel, Germany

Data flow analysis is the basis of program transformations that are performed by "optimizing" compilers in order to generate efficient object code. In this talk a new stack-based framework for interprocedural data flow analysis is introduced. The key point of this approach is the introduction of data flow analysis stacks, whose components directly correspond to the activation records representing entries of a run-time stack.

The central result of the talk is the *Interprocedural Coincidence Theorem*, which allows a characterization of *optimal* interprocedural data flow analysis algorithms. Here, optimal means the coincidence of the specifying *meet over all paths* solution of a data flow analysis problem with its algorithmical *maximal fixed point* solution. The variant of the coincidence theorem introduced here is exceptional in that it for the first time covers programs with (mutually) recursive procedures, global and local variables, and formal (value) parameters.

On-line Garbage Collection Algorithms

Antoni Kreczmar[8]
University of Warsaw
Warsaw, Poland

We investigate the problem of garbage collection for object-oriented programming languages. The aim is to solve that problem in an incremental way, i.e. allowing the execution of user actions interleaved with the process of garbage collection & space compaction. The main idea of our approach comes from Baker's algorithm which uses two semi-spaces and copies incrementally from old semi-space to a new semi-space all accessible objects, hence the memory is simultaneously compactified. The most important variant of that method called generation scavenging algorithm divides the world of objects into two generations. The world of new objects is permanently copied by Baker's algorithm while the world of old objects is compactified in emergency by a traditional mark-sweep method. We propose a new uniform method which eliminates the necessity of dividing the space. Objects are marked and rotated according to some strategy such that when a phase of that algorithm is finished all garbage is collected at one end of the space while all accessible objects are compacted at the other end of the space.

---

[7]Joint work with Bernhard Steffen, RWTH Aachen, supported by the DFG-project La 426/9-2 "Vollständige interprozedurale Programmlaufzeitoptimierung".

[8]Joint work with Marek Warpechowski, University of Warsaw.

On Programming Languages' Interpreters – Total Correctness

Hans Langmaack[9]
Christian-Albrechts-Universität zu Kiel
Kiel, Germany

Programming languages interpreters correctness proofs in the literature seem to show mainly partial correctness

$$[\![Int]\!](\pi, d) \sqsubseteq [\![\pi]\!] d,$$

in words: if an interpreter for language L is applied to an L-program π and data d and terminates then L-semantics of π applied to d is defined and both results coincide. Total correctness proofs for

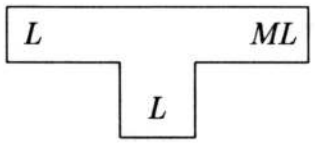
$$[\![Int]\!](\pi, d) = [\![\pi]\!] d,$$

seem that they have seldom been achieved up to now, maybe because partial correctness is sufficient in most practical applications. But not only interpreters Int formulated in a high level language $\tilde{L}$ must be partially correct, realizations Int' of Int in machine language ML should be so too.

C. A. R. Hoare's, He Jifeng's and J. Bowen's L to ML translation correctness proof (the underlying algebraic laws technique has been articulated in He Jifeng's lecture of this seminar) requires that any source L-program π applied to data d has to terminate, only then the translated program π' in ML applied to d will show a result which coincides with $[\![\pi]\!] d$; if $[\![\pi]\!] d$ is undefined it might happen that π' applied d terminates regularly with a good looking result r; Oxford's proof gives no guarantee about the nature of r in case $[\![\pi]\!] d$ is undefined.

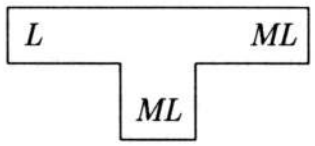
A severe situation arises when we use Int' in order to generate compilers. Let Com be an L to ML compiler written in L

Com =



In order to be sure that Int' applied to L-program Com and datum Com yields really a correct compiler

Com' =



we must know that $[\![Int]\!](Com, Com)$ is defined. Only total correctness of Int can conclude this from definedness of $[\![Com]\!] Com$ which is clear because decent compilers terminate always.

[9]Joint work with Markus Müller-Olm, Christian-Albrechts-Universität, Kiel, in the frame of the ESPRIT-BRA-Project 3104 **ProCoS** "Provably Correct Systems".

We have succeeded to prove total correctness of *Int* by using fixpoint induction in both directions. L and $\tilde{L}$ are both languages of so called recursive functions, i.e. systems of recursive function definitions.

$$[\![\pi]\!] d \sqsubseteq [\![Int]\!](\pi, d)$$

is more difficult to show than partial correctness of *Int* because nested induction is required. It is claimed that "true" wellfounded set methods seem not to be well applicable or implicitly force us to introduce equivalent operational semantics for L and $\tilde{L}$ and to do tedious bisimulations.

Bridging Existing Gaps between Theory Based Software Development and Practice

Peter E. Lauer
Mc Master University
Ontario, Canada

We propose that we must systematize our approaches to closing a number of existing gaps between theory based systems, actual systems and real world problems, to achieve ultimate success. Such systematization must precisely identify the existing gaps and discover the specific nature of each gap, so that one can initiate well-reasoned and cooperative participation of users, theoreticians and programmers aimed at closing the gap as a result of the evolution of the system and our knowledge about it.

It is suggested that success in this endeavour depends on the development of adequate theory-based abstraction tools which allow one to start from conventional embedded and heterogeneous systems, and to turn them into evolutionary systems which are described and controlled by means of systematized knowledge about the system, cooperatively by the user community, on line, and over the lifetime of the system.

We also suggest that our notion of Requirements for an Evolutionary Computer-based Information Processing Environment (RECIPE, see [1]); our notion of On the Fly Operational Semantics based on Environment Enquiry (OFOSEE, see [2]); and our notion of Computer-based Multi Media Learning (CMML); as well as our partial implementation of these notions can be seen as modest successes to construct tools for systematically closing some of the gaps indicated.

[1] Lauer, P. E., and Campbell, R. H. *RECIPE: Requirements for an Evolutionary Computer-based Information Processing Environment.* Software Process Workshop, Egham, UK, IEE 1984.

[2] Lauer, P. E. *Computer System Dossier.* In: Distributed Computing Systems, Academic Press, 1983.

Specifications Using the Data Type Syntactic Nodes

Vladimir Levin
Keldysh Institute of Applied Mathematics
Moscow, Russia

Syntactic nodes are introduced as some formalization of occurrences. They may be used to describe text interfaces, in particular, well-formed programs and text transformations, in particular, compiler functions. The complete compiler function includes two mappings: A Code-generating one and a Diagnostics one.

Syntactic nodes is a pair of a label and a number. They appear as nodes of a tree-like graph. The last is called a tree. Labels of non-terminal nodes may be used in syntax rules as syntactic sorts. Labels of terminal nodes are quoted words.

Calculations over nodes are localized within an embracing tree. So, the last is used as the Environment. Basic node operations are: $label(x)$, $text(x)$, $x \sim y$ ("x is textually congruous to y"), $x.selector$, $x.index$, $x \leq y$ ("x is subordinated to y"), $x < y$ ("x is subordinated to y in the proper way"), L/x ("the node with label L above x"), x without A ("the set of nodes which are subordinated to x and are not subordinated to any nodes from the set A; the reduction rule: nodes, which are not subordinated to x, are deleted from A in advance").

By using these node operations, one can define, for example, the scope function:

$$scope : \#DefI \rightarrow Set(\#Comp)$$

$$scope(x) = (Block/x) \text{ without } \{b : \#Block;\ x \text{ isRedeclaredIn } y\}$$

So, formal specification of well-formed programs may be like an intuitive description.

On the Specification of Files and Coroutines

Grazyna Mirkowska & Andrzej Salwicki
Universite de Pau and Warsaw University
Pau, France

We present the examples of algorithmic specifications of two different elements of programming languages: files and coroutines. We consider them as algebraic structures with corresponding operations and relations. E.g. for the system of files we consider the operations *open*, *create*, *close* etc., and for the system of coroutines - *attach*, *new*, *kill* etc. More precisely, for files we consider the hierarchy of types - sequential files, direct access files, text files - and we construct the corresponding tower of structures.

In both cases, the specification is a list of algorithmic formulas (axioms) which describes the properties of the operations and relations. Both sets of axioms are proved to be consistent. The main result is a theorem on representation, which states that any model of the system of files (the system of coroutines respectively) is isomorphic to some standard model. This proves that the axioms capture all the properties valid in any reasonable implementation. Moreover, the class of implementations which is correct with respect to the given axiomatization does not contain any pathological examples.

Computational Completeness of Specification and Programming Languages over Various Data Structures

Nikolai S. Nikitchenko
Kiev State University
Kiev, Ukraine

It is usually believed that specification and programming languages are universal, which means that any computable function may be defined in these languages. This common opinion may not be valid if computability over structures other than just natural numbers is considered. To justify this statement we present a more general notion of computability – abstract computability – applicable to an arbitrary data structure.

We start by presenting a general concept of a finite data structure defined via 'natural' data structures built over a basic set B. This computability may be considered as generalized enumeration computability relative to B. This concept is used then to generalize the standard notion of computability to arbitrary finite structures. We instantiate the presented notions to specific finite data structures, such as sequences, lists, bags, sets, maps, trees, etc.

We construct simple algebraic representations of complete classes of multi-valued and single-valued computable functions over these structures as closures of sets of basic functions under specific operators over functions. For example, the complete class of multi-valued computable functions over sets and lists (hierarchically built over B) precisely coincides with the class of functions obtained by algebraic closure of the set of basic functions $\{$ *first, tail, apndl, is-atom, choice, is-set, is-emptyset,* $\cup$, $\setminus$, $\times$, *singleton, element* $\}$ under operators of the set $\{$ *functional composition, iteration, construction, apply to all* $\}$.

In the case of infinite data structures we represent their elements as special functions and obtain computability, which is relative to such functions.

The obtained results may be used to show computational completeness of specification and programming languages.

About Timed and Hybrid Systems

Amir Pnueli[10]
Weizmann Institute of Science
Rehovot, Israel

We consider a framework for the formal specification and verification of *timed* and *hybrid* systems. This framework extends the temporal framework recommended for reactive untimed systems.

For timed systems we propose the computational model of *timed transition system* which extends conventional transition systems by the assignment of lower and upper

[10]Joint work with T. Henzinger, Y. Kesten, O. Maler, and Z. Manna.

bounds to each transition. In a computation of a timed transition system, each transition must be continuously enabled for at least its lower bound before it can be taken, and cannot be continuously enabled more than its upper bound without being taken. As specification language for timed system we consider two extended versions of temporal logic. One uses bounded versions of the operators in the style of *metric temporal logic*. The other allows reference to time through *tenure* functions which measure the length of the most recent time interval in which a given formula has been continuously true.

We also review timed statecharts for the detailed description of timed systems.

We then consider *hybrid systems*, which are systems consisting of a non-trivial mixture of discrete and continuous components, such as a digital controller that controls a continuous environment. The proposed framework extends the temporal logic approach which has proven useful for the formal analysis of discrete systems such as reactive programs. The new framework consists of a semantic model for hybrid time, the notion of *phase transition systems*, which extends the formalism of discrete transition systems, an extended version of Statecharts for the specification of hybrid behaviors, and an extended version of temporal logic that enables reasoning about continuous change.

Tools and Methods to create Simulators / C++-Prototypes for the Specification Languages SDL/OSDL and LOTOS

Andreas Prinz
Humboldt-Universität Berlin
Berlin, Germany

In the talk the approach of Humboldt-University to (protocol) specification is shown. Furthermore the use of tools for several parts of the specification process is recommended. There are three parts of our work on specifications:

a) Graphics

   1) Work on graphic versions of specification languages like GLOTOS and SDL/gr.
   2) Work on a yacc-like tool for construction of graphic editors for the languages mentioned in (a1).
   3) Work on animation of simulation results for these languages.

b) Transformation and compilation

   1) Building tools for transforming graphic versions into programming language versions of specification languages and vice versa.
   2) Constructing a kernel library for processes using C++.
   3) Compiling OSDL by means of yacc into C++-Prototypes (using (b2)).
   4) Providing means for modularization of OSDL.

c) Simulation and analysing

1) Transforming specifications (e.g. OSDL) into appropriate (guarded) Petri nets in order to use Petri net analysing tools.

2) Parallelizing of the simulation.

3) Usage of data bases for simulation results.

Specifying and Verifying Requirements of Real-Time Systems

Anders P. Ravn[11]
Technical University of Denmark
Lyngby, Denmark

An approach to specification of requirements and verification of designs for real-time, embedded computer systems is presented. A system is modelled by a conventional mathematical model for a dynamic system where application specific state variables denote total functions of real time. Specifications are formulas in the Duration Calculus, which is a real-time, interval temporal logic; atomic predicates are defined as relations between durations of states within an interval. Requirements are specified by formulas on the continuous or discrete global state of the system, and they reflect safety and functionality constraints on the system. A top level design for such a hybrid system is given by a conjunction of formulas that specify sensor, actuator and program components. Programs are specified by timed, finite state machines, which ultimately are defined by a state variable recording a trace of transition events. Actuator and sensor specifications are hybrid, relating the event trace to global system states. Verification is a proof that a design implies a more abstract design or ultimately the requirements. The approach is illustrated by the case of a Gas Burner control unit.

Keywords: Requirements engineering, formal methods, specification, verification, real-time systems, hybrid systems.

Concurrent Temporal Logic

Wolfgang Reisig
Technische Universität München
München, Germany

Concurrent Temporal Logic (CoTL) is a temporal logic, designed as a technique for proving the correctness of concurrent systems, as well as for specifying such systems. CoTL

[11]Joint work with K.M. Hansen, M.R. Hansen, H. Rischel, J.U. Skakkebaek, and Zhou Chaochen, within the **ProCoS** project, ESPRIT BRA 3104.

exploits the nature of concurrent systems by employing causality based runs as models, and by arguing on local states. This leads to distinguished rules and to an expressive power that differs from conventional temporal logics.

A particularly important rule concerns parallel composition of liveness:

$$\frac{p_1 \rightsquigarrow q_1,\ p_2 \rightsquigarrow q_2}{p_1 \wedge p_1 \rightsquigarrow q_1 \wedge q_2}$$

A rule of this form is not valid in any other known temporal logic.

We apply CoTL to elementary net systems, i.e. a standard modelling technique of concurrent systems. Elementary CoTL-formulae can directly be picked of such systems, and can be used as axioms in correctness proofs.

A Note on Compositional Refinement

Willem-Paul de Roever[12]
Christian-Albrechts-Universität zu Kiel
Kiel, Germany

Implementing a (concurrent) program P often requires changing the syntactic structure of P at various levels. We argue and illustrate that in such a situation a natural framework for implementation correctness requires a more general notion of refinement than that of [Hoare, He Jifeng & Saunders '87], a notion which involves the introduction of separate refinement relations for P's various abstract components. An outline is given of a formal framework for proving implementation correctness which involves these notions inside a unified mixed term framework in which relations and predicate transformers are combined. The resulting formalism is applied to deriving the correctness of (1) an implementation of a synchronous message passing algorithm involving action refinement of messages by sequences of packages using a sliding window protocol, (2) a self-stabilizing algorithm due to Katz & Peled for computing a global snapshot of a distributed computation.

Toward Formal Development of Programs from Algebraic Specifications: Parameterisation Revisited

Don Sannella[13]
University of Edinburgh
Edinburgh, UK

Modular structure is an important tool for managing large and complex systems of interacting units, and its use in organizing algebraic specifications is well-known. An important structuring mechanism is parameterisation, which allows program and specification

[12]The paper appears in the Proceedings of the 5^{th} Refinement Workshop, Springer's "Workshops in Computing" Series, and is co-authored by Job Zwiers, Twente University, and Jos Coenen, University of Eindhoven.

[13]This is joint work with Andrzej Tarlecki and Stefan Sokołowski of the Polish Academy of Sciences.

modules to be defined in a generic fashion and applied in a variety of contexts. Appropriate parameterisation mechanisms give rise to parameterised programs and parameterised specifications. It is possible to specify parameterised programs, just as it is possible to specify non-parameterised programs. The result is *not* a parameterised specification; it is a non-parameterised specification of a parameterised program. In work on algebraic specification there has been a tendency to ignore this distinction and use one flavour of parameterisation for both purposes. This has sometimes led to misunderstanding and confusion. The distinction is important both for semantical reasons and because the two kinds of specifications belong to different phases of program development: parameterised specifications are used to structure requirements specifications, while specifications of parameterised programs are used to describe the modules which arise in the design of an implementation. This part of the talk may be summarized by the following slogan:

parameterised (program specification) $\neq$ (parameterised program) specification

It is natural to consider what happens when parameterisation mechanisms are extended to the higher-order case in which parameterised objects are permitted as arguments and as results. This makes sense both for parameterised specifications and for specifications of parameterised programs, and has interesting methodological consequences. We have designed a calculus in which it is possible to construct arbitrarily higher-order parameterised programs, parameterised specifications, and mixtures of these such as parameterised specifications of parameterised programs.

Exploiting Concurrency for Linear Time Model Checking of Petri Nets

Bernd-Holger Schlingloff[14]
Technische Universität München
München, Germany

In this talk a model checking algorithm for linear time temporal logic and one safe Petri nets was developed, which traverses the state space online: That is, atoms —consisting of a state (marking) of the net and a guess for "until" subformulas— are constructed depth first, and upon backtrack from the recursion the strongly connected components of these atoms are checked for unsatisfied eventualities.

Then an improvement of this algorithm was derived, which avoids the usual state explosion problem by generating only some out of all possible interleaving execution sequences. This is done by looking at the dependency of an arbitrary enabled transition t: t' is in the dependency of t if the firing of t' could lead to the firing of a transition conflicting with t. Special attention has to be paid to those transitions which may alter the truth value of the given formula as well as to continuously enabled transitions.

Experimental results show that for nets with little or no conflict it is possible to reduce the exponential complexity of the problem to a linear complexity.

[14]Joint work with Tomohiro Yoneda, Tokyo Institute of Technology, and Edmund M. Clarke, Carnegie-Mellon-University.

Semantic Domains with Congruences

Gunther Schmidt[15]
Universität der Bundeswehr München
München, Germany

Rigorous software development requires among other concepts higher order functions and algebraic specifications. For the semantical foundation of higher order functions researchers use constructs like countably algebraic cpo's. Algebraic specifications are interpreted considering the term algebra in which congruences are introduced by laws. If these aspects are studied together, the orderings and congruences obtained should be compatible with one another. We define the concept of compatibility of an inductive ordering with a congruence. As our main result we show how to close a class of algebraic cpo's against division by a congruence. To obtain the result some additional requirements are necessary which, however, are satisfied when the congruence originates from the introduction of algebraic laws.

Interactive Programming with Higher Order Objects, Part I[16]

Gunther Schmidt
Universität der Bundeswehr München
München, Germany

A report on the system HOPS (Higher Object Programming System) is given. Basically spoken, the HOPS-System gives a set of LEGO-like bricks or building-blocks that may be visualized on the screen and manipulated by the mouse under strong system control. The bricks offered resemble generic constructs to be obtained in connection with the universal characterization of domain constructions such as explicit domains, direct sum, direct product, function space, inverse limit construction, lifting, introduction of new sorts and function symbols as well as the introduction of new algebraic laws.

The approach taken gives the opportunity of using higher order and algebraic constructs together. Furthermore, it is possible to parameterize the constructs obtained on all levels.

The system is planned to enable type-controlled constructions of directed acyclic graphs (DAGs) that can afterwards be transformed using self-defined rules by some sophisticated rule-automaton. At the end of a development the emission of program text in the usual programming languages is possible. In prototypes the generation of texts in ADA, PASCAL, Modula-2 and C has been studied.

The report focussed on several aspects of the system: DAG-manipulation, layered DAG-language, rule-formalization, relational semantics of the DAG-language and issues raised in programming methodology.

[15]This is joint work with Peter Kempf, Universität der Bundeswehr München.

[16]See also *"Interactive Programming with Higher Order Objects, Part II"* by Rudolf Berghammer.

Logic Based Program Development and Tactical Theorem Proving

Werner Stephan
Universität Karlsruhe
Karlsruhe, Germany

Tactical Theorem Proving is proposed as a promising architecture for deduction systems to be used in formal program development. By an example, the derivation of an algorithm that computes equivalence classes, it is shown how entirely different deduction problems occur as proof obligations. The presentation is based on Dynamic Logic that forms the logical basis of the Karlsruhe Interactive Verifier. Problems mentioned are the synthesis and a posteriori verification of program fragments satisfying certain correctness assertions, the proof of verification conditions, and the logical treatment of program transformations and data-refinements. It is argued that Tactical Theorem Proving can be used to integrate the various derived calculi and special purpose systems appropriate for these classes of deduction problems into a common framework.

Dagstuhl-Seminar 9203

List of Participants (update: 4. 8. 92)

Jaco W. **de Bakker**
CWI - Mathematisch Centrum
Kruislaan 413
NL-1098 SJ Amsterdam
The Netherlands
mieke@cwi.nl
tel.: +31 20 592 4136 / 4058

Rudolf **Berghammer**
Universität der Bundeswehr München
Fakultät für Informatik
Institut fuer Programmiersprachen
Werner-Heisenberg-Weg 39
W-8014 Neubiberg
Germany
rudolf@informatik.unibw-muenchen.de
tel.: +49-89 6004 2261

Jonathan **Bowen**
Oxford University
Computing Laboratory
Programming Research Group
11 Keble Road
Oxford OX1 3QD
Great Britain
Jonathan.Bowen@comlab.ox.ac.uk
tel.: +44 865 27 25 74 / 27 38 40 (Secr.)

Manfred **Broy**
TU München
Institut für Informatik
Arcisstraße 21
W-8000 München 2
Germany
broy@lan.informatik.tu-muenchen.dbp.de
tel.: +49-89 2105 8161

Ole-Johan **Dahl**
Universitetet i Oslo
Institutt for Informatikk
Postboks 1080 Blindern
0316 Oslo
Norway
olejohan@ifi.uio.no
tel.: +47 2 45 34 48

Walter **Dosch**
Universität Augsburg
Institut für Mathematik
Universitätsstraße
W- 8900 Augsburg
Germany
dosch@uni-augsburg.de
tel.: +49-821 598 2170

Peter **Forbrig**
Universität Rostock
Fachbereich Informatik
Albert-Einstein-Str. 21
O-2500 Rostock
Germany
forbrig@informatik.uni-rostock.dbp.de
tel.: +37 81 44424 162

Gerhard **Goos**
Universität Karlsruhe
Fakultät für Informatik
Vincenz-Prießnitz-Str. 1
W-7500 Karlsruhe
Germany
ggoos@ira.uka.da
tel.: +49-721 66 22 66

Michal **Grabowski**
University of Warsaw
Institute of Informatics
Ul. Banacha 2
02-097 Warszawa
Poland
mich@mimuw.edu.pl
tel.: +48 22 658 31 65

Jifeng **He**
Oxford University
Computing Laboratory
Programming Research Group
11 Keble Road
Oxford OX1 3QD
Great Britain
jifeng@prg.oxford.ac.uk
tel.: +44 865 27 25 75

Friedrich Wilhelm **v. Henke**
Universität Ulm
Fakultät für Informatik
James-Franck-Ring
W-7900 Ulm
Germany
vhenke@informatik.uni-ulm.de
tel.: +49-731 502 4120

Hardi **Hungar**
Universität Oldenburg
FB 10 - Informatik
Ammerländer Herrstr. 114
W-2900 Oldenburg
Germany
hardi.hungar@arbi.informatik.uni-oldenburg.de
tel.: +49-441 798 3046

Stefan **Jähnichen**
TU Berlin - GMD-FIRST
Forschungszentrum Innovative
Rechnersysteme und Softwaretechnik
Hardenbergplatz 2
W-1000 Berlin 12
Germany
jaehn@cs.tu-berlin.de
tel.: +49-30 254 99 104

Burghard **von Karger**
Universität Kiel
Inst. für Informatik und Prakt. Mathematik
Haus II
Preusserstrasse 1-9
W-2300 Kiel 1
Germany
bvk@causun.uucp
tel.: +49-431 56 04 37

Jens **Knoop**
Universität Kiel
Inst. für Informatik und Prakt. Mathematik
Haus II
Preusserstrasse 1-9
W-2300 Kiel 1
Germany
jk@informatik.uni-kiel.dbp.de
tel.: +49-431 56 04 34

Antoni **Kreczmar**
University of Warsaw
Institute of Informatics
Ul. Banacha 2
02-097 Warszawa
Poland
antek@plearn.bitnet
tel.: +48 22 658 31 64

Hans **Langmaack**
Universität Kiel
Inst. für Informatik und Prakt. Mathematik
Haus II
Preusserstrasse 1-9
W-2300 Kiel 1
Germany
hl@causun.uucp
tel.: +49-431 56 04 28 / 27 (Sekr.)

Peter **Lauer**
McMaster University
Dept. of Comp. Science & Systems
1280 Main Street West
Hamilton Ontario L8S 4K1
Canada
lauer@mcmaster.ca
tel.: +1 416 648 1525

Vladimir **Levin**
Keldysh Institute of Applied Mathematics
Miusskaya sg.4
Moskau 125047
Russia
VLGAL@KELDYSH.MSK.SU
tel.: +7 095 333 80 45

Claus **Lewerentz**
FZI Karlsruhe
Haid-und-Neu-Straße 10-14
W-7500 Karlsruhe 1
Germany
lewerentz@fzi.de
tel.: +49-721 9654 602

Grazyna **Mirkowska**
Université de Pau
Departement d'Informatique
Avenue de l'Université
F-64000 Pau
France
mirkowska@fruppa51.bitnet
tel.: +33 59 92 31 54

Nikolai **Nikitchenko**
Kiew State University
Departement of Cybernetics
Vladimizskaja Street 64
Kiew-17
Ukraine
tel.: +7 044 513 34 04 (home)

Manfred **Paul**
TU München
Institut für Informatik
Arcisstraße 21
W-8000 München 2
Germany
paul@informatik.tu-muenchen.de
tel.: +49-89 48095 161

Amir **Pnueli**
Weizmann Institute
Departement of Appl. Mathematics
P.O. Box 26
76100 Rehovot
Israel
amir@wisdom.weizmann.ac.il
tel.: +972-8-343434

Andreas **Prinz**
Humboldt Universität
Fachbereich Informatik
Unter den Linden 6
O-1086 Berlin
Germany
prinz@hubinf.uucp
tel.: +37 2 2093 2348

Anders P. **Ravn**
Danmarks Teknikske Hojskole
Instituttet for Datateknik
Build. 344
DK-2800 Lyngby
Danmark
apr@id.dth.dk
tel.: +45-45 93 12 22-37 47

Wolfgang **Reisig**
TU München
Institut für Informatik
Arcisstraße 21
W-8000 München 2
Germany
reisig@lan.informatik.tu-muenchen.dbp.de
tel.: +49-89 2105 2405

Günter **Riedewald**
Universität Rostock
Fachbereich Informatik
Albert-Einstein-Str. 21
O-2500 Rostock
Germany
riedewald@uni-rostock.informatik.dbp.de
tel.: +37 81 44 4 24 App. 117

Willem P. **de Roever**
Universität Kiel
Inst. für Informatik und Prakt. Mathematik
Haus II
Preusserstrasse 1-9
W-2300 Kiel 1
Germany
wpr@informatik.uni-kiel.dbp.de
tel.: +49-431 56 04 71 / 74

Andrzej **Salwicki**
Université de Pau
Departement d'Informatique
Avenue de l'Université
64000 Pau
France
salwicki@fruppa51.bitnet
tel.: +33 59 92 31 54

Donald **Sannella**
University of Edinburgh
Dept. of Computer Science
Mayfield Road
Edinburgh EH9 3JZ
Great Britain
dts@dcs.ed.ac.uk
tel.: +44 31 650 51 84

Holger **Schlingloff**
TU München
Institut für Informatik
Arcisstraße 21
W-8000 München 2
Germany
schlingl@informatik.tu-muenchen.de
tel.: +49-89 48095 140

Gunther **Schmidt**
Universität der Bundeswehr München
Fakultät für Informatik
Werner-Heisenberg-Weg 39
W-8014 Neubiberg
Germany
schmidt@informatik.unibw-muenchen.de
tel.: +49-89 6004 24 49 / 22 63

Werner **Stephan**
Universität Karlsruhe
Fakultät für Informatik
Postfach 6980
W-7500 Karlsruhe
Germany
stephan@ira.uka.de
tel.: +49-721 608 39 77

Zuletzt erschienene und geplante Titel:

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 90s and Beyond, 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.-12.9.91 (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. Dress, M. Karpinski, M. Singer(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)

K. Compton, J.E. Pin , W. Thomas (editors):
Automata Theory: Infinite Computations, Dagstuhl-Seminar-Report; 28, 6.-10.1.92 (9202)

H. Langmaack, E. Neuhold, M. Paul (editors):
Software Construction - Foundation and Application, Dagstuhl-Seminar-Report; 29, 13..-17.1.92 (9203)

K. Ambos-Spies, S. Homer, U. Schöning (editors):
Structure and Complexity Theory, Dagstuhl-Seminar-Report; 30, 3.-7.02.92 (9206)

B. Booß, W. Coy, J.-M. Pflüger (editors):
Limits of Modelling with Programmed Machines, Dagstuhl-Seminar-Report; 31, 10.-14.2.92 (9207)

K. Compton, J.E. Pin , W. Thomas (editors):
Automata Theory: Infinite Computations, Dagstuhl-Seminar-Report; 28, 6.-10.1.92 (9202)

H. Langmaack, E. Neuhold, M. Paul (editors):
Software Construction - Foundation and Application, Dagstuhl-Seminar-Report; 29, 13.-17.1.92 (9203)

K. Ambos-Spies, S. Homer, U. Schöning (editors):
Structure and Complexity Theory, Dagstuhl-Seminar-Report; 30, 3.-7.2.92 (9206)

B. Booß, W. Coy, J.-M. Pflüger (editors):
Limits of Information-technological Models, Dagstuhl-Seminar-Report; 31, 10.-14.2.92 (9207)

N. Habermann, W.F. Tichy (editors):
Future Directions in Software Engineering, Dagstuhl-Seminar-Report; 32; 17.2.-21.2.92 (9208)

R. Cole, E.W. Mayr, F. Meyer auf der Heide (editors):
Parallel and Distributed Algorithms; Dagstuhl-Seminar-Report; 33; 2.3.-6.3.92 (9210)

P. Klint, T. Reps, G. Snelting (editors):
Programming Environments; Dagstuhl-Seminar-Report; 34; 9.3.-13.3.92 (9211)

H.-D. Ehrich, J.A. Goguen, A. Sernadas (editors):
Foundations of Information Systems Specification and Design; Dagstuhl-Seminar-Report; 35; 16.3.-19.3.9 (9212)

W. Damm, Ch. Hankin, J. Hughes (editors):
Functional Languages:
Compiler Technology and Parallelism; Dagstuhl-Seminar-Report; 36; 23.3.-27.3.92 (9213)

Th. Beth, W. Diffie, G.J. Simmons (editors):
System Security; Dagstuhl-Seminar-Report; 37; 30.3.-3.4.92 (9214)

C.A. Ellis, M. Jarke (editors):
Distributed Cooperation in Integrated Information Systems; Dagstuhl-Seminar-Report; 38; 5.4.-9.4.92 (9215)

J. Buchmann, H. Niederreiter, A.M. Odlyzko, H.G. Zimmer (editors):
Algorithms and Number Theory, Dagstuhl-Seminar-Report; 39; 22.06.-26.06.92 (9226)

E. Börger, Y. Gurevich, H. Kleine-Büning, M.M. Richter (editors):
Computer Science Logic, Dagstuhl-Seminar-Report; 40; 13.07.-17.07.92 (9229)

J. von zur Gathen, M. Karpinski, D. Kozen (editors):
Algebraic Complexity and Parallelism, Dagstuhl-Seminar-Report; 41; 20.07.-24.07.92 (9230)

F. Baader, J. Siekmann, W. Snyder (editors):
6th International Workshop on Unification, Dagstuhl-Seminar-Report; 42; 29.07.-31.07.92 (9231)

J.W. Davenport, F. Krückeberg, R.E. Moore, S. Rump (editors):
Symbolic, algebraic and validated numerical Computation, Dagstuhl-Seminar-Report; 43; 03.08.-07.08.92 (9232)

R. Cohen, W. Wahlster (editors):
3rd International Workshop on User Modeling, Dagstuhl-Seminar-Report; 44; 10.-14.8.92 (9233)

R. Reischuk, D. Uhlig (editors):
Complexity and Realization of Boolean Functions, Dagstuhl-Seminar-Report; 45; 24.08.-28.08.92 (9235)

Th. Lengauer, D. Schomburg, M.S. Waterman (editors):
Molecular Bioinformatics, Dagstuhl-Seminar-Report; 46; 07.09.-11.09.92 (9237)

V.R. Basili, H.D. Rombach, R.W. Selby (editors):
Experimental Software Engineering Issues, Dagstuhl-Seminar-Report; 47; 14.-18.09.92 (9238)

Y. Dittrich, H. Hastedt, P. Schefe (editors):
Computer Science and Philosophy, Dagstuhl-Seminar-Report; 48; 21.09.-25.09.92 (9239)

R.P. Daley, U. Furbach, K.P. Jantke (editors):
Analogical and Inductive Inference 1992 , Dagstuhl-Seminar-Report; 49; 05.10.-09.10.92 (9241)

E. Novak, St. Smale, J.F. Traub (editors):
Algorithms and Complexity of Continuous Problems, Dagstuhl-Seminar-Report; 50; 12.10.-16.10.92 (9242)

J. Encarnação, J. Foley (editors):
Multimedia - System Architectures and Applications, Dagstuhl-Seminar-Report; 51; 02.11.-06.11.92 (9245)

F.J. Rammig, J. Staunstrup, G. Zimmermann (editors):
Self-Timed Design, Dagstuhl-Seminar-Report; 52; 30.11.-04.12.92 (9249)