B. Courcelle, H. Ehrig, G. Rozenberg,
H.J. Schneider (editors):

Graph-Transformations in Computer Science

Dagstuhl-Seminar-Report; 53
04.01.-08.01.93 (9301)

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

Report on the Dagstuhl-Seminar 9301

Graph Transformations in Computer Science

January 4 – 8, 1993

Organizers:

Bruno Courcelle (Bordeaux)
Hartmut Ehrig (Berlin)
Grzegorz Rozenberg (Leiden)
Hans Jürgen Schneider (Erlangen)

The research area of graph grammars resp. graph transformations is a relatively young discipline of computer science. Its origins date back to the early seventies. Nevertheless methods, techniques, and results from the area of graph transformation have already been studied and applied in many fields of computer science such as formal language theory, pattern recognition and generation, compiler construction, software engineering, concurrent and distributed systems modelling, database design and theory, etc.

This wide applicability is due to the fact that graphs are a very natural way to explain complex situations on an intuitive level. Hence they are used in computer science almost everywhere, e.g. as data- and control flow diagrams, entity relationship diagrams, Petri-nets, visualization of soft- and hardware architectures, evolution diagrams of non-deterministic processes, SADT-diagrams, and many more. Like the "token game" for Petri-nets, graph transformation brings dynamics to all these descriptions, since it can describe the evolution of graphical structures. Therefore graph transformation becomes attractive as a "programming paradigm" for complex structured software and graphical interfaces. In particular graph rewriting is promising as a comprehensive framework in which the transformation of all these very different structures can be modelled and studied in a uniform way.

During this Dagstuhl-Seminar 33 lectures and 3 system demonstrations where presented by the participants from 8 European countries, U.S.A. and Japan in the following areas:

- Foundations of Graph Grammars and Transformations

- Applications of Graph Transformations to
  - Concurrent Computing
  - Specification and Programming
  - Pattern Generation and Recognition

The system demonstrations in the evening showed efficient implementations of the algebraic approach to graph transformations (AGG-System), of a software specification language based on graph rewriting (PROGRESS) and of a functional programming language (Concurrent Clean) based on term graph rewriting. In each case the theoretical techniques of the underlying approach and typical applications have been demonstrated in corresponding lectures during the day. In addition, interesting new applications of graph transformations were presented in several lectures in the following areas: Concurrent constraint programming, actor systems, specification of languages for distributed systems, of hybrid database languages and of an efficient narrowing machine, and — last but not least — to pretty pattern generation and recognition ranging from graphical modelling for CAD to abstractions of modern art including Escher and Picasso.

In the lectures concerning foundations on one hand new results concerning graph languages and graph automata and their connections to decision problems were presented. On the other hand new concepts and results for the algebraic approach to graph transformations based on double and single pushouts were shown. Notions for abstraction and semantical constructions were given leading to canonical derivation sequences, true concurrency and event structures and also extensions of results from graph grammars to HLR (High Level Replacement)-systems. The HLR-approach is a categorical unification of different approaches with several interesting new applications including those to rule based modular system design and transformation and refinement of Petri-nets.

Altogether it was a very fruitful interaction between theory, applications and practical demonstrations.

This Dagstuhl workshop was coorganized by the ESPRIT Basic Research WG COMPUGRAPH and will be followed by a series of related workshops in Leiden (Fall 1993), Williamsburg, U.S.A. (1994), Pisa (1995) and hopefully again in Dagstuhl in 1996.

All participants appreciated the stimulating atmosphere in Schloß Dagstuhl and expressed their thanks to the IBFI (Internationales Begegnungs- und Forschungszentrum für Informatik) and the ESPRIT WG COMPUGRAPH for organization and support of this successful seminar.

January 1993

Hartmut Ehrig
Hans Jürgen Schneider

Contents

1 Foundations

Review and New Aspects of Algebraic Graph Transformations and High-Level-Replacement Systems

Hartmut Ehrig
Technische Universität Berlin

Algebraic Graph Transformations are based on a gluing construction in the category of graphs. In the classical case we have two gluing constructions which are pushouts in the category of graphs. This double pushout approach for graphs has been generalized to objects in quite general categories leading to the notion of high level replacement systems. Using partial instead of total graph morphisms was the main idea to develop the single pushout approach for graph transformations which was recently also generalized to a suitable categorical setting. This is now called single pushout approach for high level replacement systems.

An overview of main constructions and results in both approaches has been presented in the lecture together with some new specific results. Moreover problems of abstraction of graphs and graph derivations (up to isomorphism) can be solved using the notion of standard representation of graphs.

Canonical Derivations

Hans-Jörg Kreowski
Universität Bremen

Imagine a concurrent system. An observer would see a derivation, i.e. a sequence of actions where each action may be the parallel composition of component actions. Unfortunately, independent actions may be observed to happen one after the other or—from another point of view—in parallel. Therefore, a process may be observable through many (e.g. exponentially many) equivalent derivations. If one is interested in the processes, one faces the problem to represent them in an adequate and efficient way.

Canonical derivations are candidates to solve the problem. The idea is to look for derivations where each component action appears as early as possible. For this purpose, a shift is introduced that moves a component action to the preceding derivation step (if equivalence is preserved). Obviously, shifting-as-long-as-possible yields a canonical derivation whenever the procedure terminates. There is

a straightforward way to guarantee termination, but uniqueness is more troublesome to get. The main result states that equivalent canonical derivations are equal if the analysis relation between parallel actions and their corresponding sequential views behaves "nicely".

It is known that several graph grammar approaches and Petri nets, for example, are "nice" enough. A more recent result (by Ehrig, Taentzer & myself) reveals that high-level replacement systems meet the conditions under certain assumptions (that are fulfilled in many cases).

Canonical Derivations in Single-Pushout Graph Transformation

Michael Löwe and Jürgen Dingl
Technische Universität Berlin

The single-pushout approach to graph transformation comes equipped with an interesting parallel composition of rules and rule applications: some parallel steps cannot be decomposed into sequences with the component rules. And the possibility of one sequentialization does not imply that all sequentializations are realizable. This makes up a significant difference to the classical algebraic approach based on double pushout derivations which guarantees the existence of all sequentializations for all parallel steps.

We show that the equivalence which arises from different sequentializations of the same sequence is efficiently decidable for the single-pushout approach as well. Furthermore, there is for each sequence an unique equivalent one which realizes maximal parallelism in the sense that each atomic step in this sequence is performed "earlier" than in any other equivalent sequence. This so-called canonical sequence can be effectively computed for each given sequence using the decision algorithm mentioned above. The amount of parallelism, however, which we find in the canonical derivation sequences for the single-pushout approach considerably exceeds the parallelism obtained in the double pushout framework for the "same" input sequence.

AGG — An Algebraic Graph Grammar System and its Specification with Extended Parallel Graph Grammars

Martin Beyer, Gabriele Taentzer
Technische Universität Berlin

The AGG-system is a prototype implementation of the algebraic approach to graph transformation. It has been programmed in EIFFEL and runs on SUN4-workstations under X windows 11.5. It consists of a flexible graph editor and a

derivation component. The I/O is mouse/menue driven. A comprehensive set of operations facilitates the efficient manipulation and clear visualization of graphs. The editor allows the graphical manipulation of production rules, redices and derivation results. The derivation component performs direct transformation steps for user-selected production rules and redices using the so-called single pushout approach to graph rewriting.

First steps towards a graph specification of the AGG-system can be made by using extended parallel graph grammars. By now, an abstract version of an AGG-state is modelled by a graph and AGG-operations are modelled by graph derivations, which are performed in two steps. First a new production rule is generated from a set of elementary production rules by amalgamating them and then this new production rule is applied to the actual graph by a direct derivation. Using this kind of amalgamated two-level derivations the extended parallel graph grammars allow dynamic interfaces, partial coverings of the mother graph and operational production sets which are tables of sets describing one operation.

Transformations of Relational Structures

Yasuo Kawahara
Kyushu University, Fukuoka, Japan

This talk presented an attempt to propose a fundamental framework for graph transformations from the viewpoint of relational calculus, i.e., theory of binary relations. To this end notions of relational structures and their morphisms are introduced over a frame consisting of a pair of natural transformations between set-valued functors. The relational structures include relational algebra, simple graphs, labeled graphs, hypergraphs and so on. The category of relational structures and their morphisms are complete and co-complete under weak conditions. Given an admissible class of monomorphisms, partial morphisms of relational structures are defined. Then the categories of relational structures and their partial morphisms has pushouts if some continuity conditions hold. In this case transformations of relational structures with single pushouts can be freely achieved, and some properties on commutativity of rewritings and confluency (Church-Rosser) of derivations are shown by the similar idea due to Raoult.

Set-Theoretic Graph Rewritings

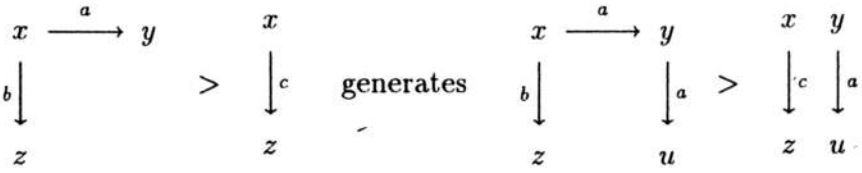
Jean-Claude Raoult, Frédéric Voisin
IRISA Rennes, LRI Orsay

We consider graphs as sets of labelled hyperarcs like $fxyz$, where f is the label of the arc, and x, y, z are the vertices to which this arc is attached, in this order. A rewrite relation is defined to be a binary relation $G \to G'$ preserved by adding contexts:

$$G \to G' \quad \text{implies} \quad G + C \to G' + C$$

Some care must be taken regarding vertices. We have three options: 1) admit all contexts, 2) vertices appearing new in g' should also appear new in $G' + C$, 3) new vertices in G' remain new in $G' + C$. The applications envisioned rule out the first alternative. The third is called a reversible rewriting, and is a particular case of the second.

For instance,



These rewritings can represent the single pushout approach in the case when the partial morphisms are injective. No gluing condition is needed.

The composition of two reversible rewritings is a reversible rewriting and an algorithm gives a systems generating the composition. The composition of two general rewritings is not a rewriting in general, but the induced rewriting is generated by a system which can be obtained by the same algorithm.

This stability under composition is false for parallel rewritings of graphs and of terms, and for single step rewritings of terms.

Graph Rewriting and Copying

Erik Barendsen, Sjaak Smetsers
University of Nijmegen

This work focuses on the theoretical aspects of (term) graph rewriting.

Graph reduction is described in terms of general operations on graphs, leading to a substitution-like mechanism called *graph replacement*. Some basic theory on the interaction of these operations is developed. Equipped with the ordering induced by *graph homomorphisms*, various sets of graphs have nice domain theoretical

properties. A Church-Rosser result for so-called interference-free orthogonal graph rewrite systems is obtained. Confluence is preserved if reduction is mixed with partial unraveling of graphs.

The notion of term graph rewrite system (TGRS) is extended with a *lazy copying* mechanism. By analyzing this mechanism, a confluence result for *copy term graph rewrite systems* (C-TGRS's) is obtained. Some ideas on the use of lazy copying combined with a node-selecting reduction strategy are presented.

C-TGRS's can be used to model parallel computations on loosely coupled machine architectures, see [1].

[1] M.C.J.D. van Eekelen, M.J. Plasmeijer and J.E.W. Smetsers, 1991: Parallel graph rewriting on loosely coupled machine architectures, in: Proceedings CTRS'90, LNCS 516, Springer-Verlag, Berlin, pp. 354–369.

Graph rewriting using a single pushout; a comparison

Pim van den Broek, Jan Kuper
Universiteit Twente

Recently two algebraic approaches to graph rewriting have been given which use single pushouts. Here we compare both approaches. The rewrite results in both approaches are similar, while the approaches differ in the applicability of a rule when an occurence of its left hand side is given.

Restricting Derivation Sequences: Single or Double Pushout

Francesco Parisi-Presicce
Università degli Studi L'Aquila

Motivated by applications to Software System Design, where productions represent interfaces and derivations using these productions can be translated into interconnections of the corresponding software components, the classical double pushout approach to the algebraic theory of graph grammars is modified by allowing "restricting" steps to precede and follow each direct derivation. Graph Grammars with restricting derivation sequences based on double pushouts are strictly more expressive than those using standard derivations based on single pushouts. Allowing restricting steps, the single pushout approach and the double pushout approach have the same expressive power. By allowing the set of productions to change, standard double pushout derivations can simulate restricting derivation sequences.

2 Concurrent Computing

On Graph Grammars, Distributed Rewriting Systems, and Event Structures

Georg Schied
Universität Erlangen-Nürnberg

Rewriting systems constitute an operational model for distributed systems that is based upon the notion of states and state transitions. Event structures, on the other side, are a more abstract description of the behaviour of distributed systems that is based on events, causality, and conflicts.

We introduce *distributed rewriting systems* as a unifying approach for different kinds of rewriting systems, like string or graph grammars. Contexts and an appropriate operator for inserting structures into the holes of a context are the foundation of this model. The double pushout approach for graph grammars nicely fits into this general framework. By the way, we obtain some simplified proofs for the Church-Rosser and parallelism theorems of graph grammars.

Derivation processes of rewriting systems intuitively can be described with event structures. We present an outline of the construction that relates the derivations of a rewriting system to its corresponding event structure.

Equivalence and Composition of ESM systems (Actor grammars)

Dirk Janssens
VUB Brussels

ESM systems are graph rewriting systems in which a step is obtained by a single pushout construction in a category of graph structures where morphisms are a variant of the usual structure morphisms, called ESM morphisms. This type of systems generalize a graph grammar model of actor systems, called actor grammars, that was introduced and investigated in the last 6 years.

Starting from the idea that a system (program) is a set of ESM morphisms (productions), a compositional semantics is developed; composition of systems is set union. The basic objects considered are concrete representations of rewriting processes, which are composed using a gluing operation. The corresponding semantics is made compositional by adding to a process some information about the

contexts in which it may occur, and then the desired semantics is obtained by abstracting from the internal structure of the processes.

As a special case one gets a simple compositional semantics for P/T nets in which nets are composed by gluing over places.

Graph Rewriting Systems for Concurrent Constraint Programming

Andrea Corradini, Ugo Montanari, Francesca Rossi
Dipartimento di Informatica, University of Pisa

The concurrent constraint programming framework extends both constraint logic programming and concurrent logic programming in that a program consists of the concurrent execution of agents which add (i.e. "tell") and check (i.e. "ask") constraints on a shared set of variables, and whose behaviour is described by a set of clauses. This formulation is very general and can be seen as a concurrent logic programming shell which is parametrized w.r.t. the underlying constraint systems.

A model called CHARM (for Concurrency and Hiding in an Abstract Rewriting Machine) is proposed as the abstract machine of concurrent constraint programming. CHARM is equipped with a clean operational semantics based on term rewriting over a suitable algebra, and it exhibits a sophisticated treatment of concurrency and modularity, which is obtained through the partition of each state into a global and a local part.

It is shown that a rewriting step of CHARM faithfully models the direct derivation of graphs described by a double pushout construction in the style of Ehrig and Schneider. A straightforward implementation of the concurrent constraint language on the CHARM is finally presented.

True Concurrency in Graph Grammars

Andrea Corradini[*], *Hartmut Ehrig*[+], *Michael Löwe*[+], *Ugo Montanari*[*], *Francesca Rossi*[*]
[*]*University of Pisa*, [+]*TU Berlin*

We first propose a partial order semantics for graph grammars, based on the notion of graph processes. Such graph processes are natural extensions of the notion of process for Petri nets, where the set of places is replaced by a graph and the flow relation is split into two to be able to represent in a correct way the interface graph in a graph production. Graph processes are shown to be in bijective correspondence with canonical derivations up to isomorphism, i.e., they represent the same class of derivations.

However, previous work of the same authors [1] showed that this notion of abstraction (up to isomorphism) is not adequate for concatenating derivations, since it allows the identification of too many derivations. To put a suitable restriction, the notion of standard isomorphism and standard graph representatives has been introduced in that work.

Therefore we examine the role of the notion of standard representation in defining a new and more adequate concept of abstraction (of graphs and graph derivations). Through various attempts and counterexamples, where abstract derivations and processes are always in bijective correspondence but the level of concurrency shown is not the desired one, we finally are able to define abstract derivations correctly: as derivations up to standard isomorphism on the initial and final graph and on the graphs of the used productions (and isomorphism on the other graphs). Then we pass from the category of such abstract derivations to an event structure semantics for graph grammars by using an extension of a technique already used for P/T Petri nets: first we add new arrows in the category, representing all automorphisms among abstract graphs; then we compute the comma category w.r.t. the initial graph of the given grammar; then we identify all objects of such comma category which are involved in cycles, and all arrows between two objects. We claim that in this way we always obtain a prime algebraic domain, which it is known to correspond to a prime event structure.

[1] Corradini et al., "Note on Standard Representation of Graphs", TR 92/25, TU Berlin, 1992.

Event Structures and Orthogonal Term Graph Rewriting

J.R. Kennaway, J.W. Klop, M.R. Sleep, and F.J. de Vries

Several authors have hinted at a connection between transition systems such as are used to describe concurrency, and the reduction sequences that arise in term rewriting and lambda calculus. We make such a connection precise for orthogonal term graph rewriting systems. We do this by associating with every normalizable term graph in such a system a conflict-free elementary event structure. The events of this structure represent the different reduction steps which must be performed to reduce the term graph to normal form. Their partial ordering represents an intuitive notion of dependency of one reduction step on another. The elements of the associated configuration domain are the Lévy-equivalence classes of reduction sequences which perform no unnecessary work, and their partial ordering is identical to the Lévy ordering. The size of the set of events is the number of reduction steps which must be performed in any reduction of the term graph to normal form. This number is independent of the order in which the reductions are performed. The bottom element of the configuration domain represents the empty sequence,

and the top element represents reduction of the term to normal form. The height of the domain places a lower bound on the time required to reduce the term graph to normal form, even when parallelism is employed. The width of the configuration domain (i.e. the size of the largest antichain) places an upper bound on the amount of parallelism that may be usefully employed in the reduction.

(To appear in "Term Graph Rewriting: Theory and Practice", eds. M.R. Sleep, M.J. Plasmeijer, and M.C.J.D. van Eekelen. John Wiley and Sons, 1993.)

Single Pushout Approach for Equationally Defined Graph Structures with Applications to Concurrent Systems

Martin Korff
Technische Universität Berlin

In the single pushout approach transformations are defined as simple pushouts within a category of graph structures and partial morphisms. Graph structures are simply algebras w.r.t. an appropriate algebraic signature. Adding conditional equations to such signatures induces a subcategory of equationally defined graph structures and partial morphisms. Again, transformations are single pushouts.

As the main result we show that each pushout in the equational category is characterized by the corresponding pushout in the supercategory without equations if and only if the pushout object already satisfies the given equations. Thus in general theoretical results become much weaker compared to the case without equations. However for a restricted class of so-called local equations some of the classical results can be reobtained. In the labelled case the situation becomes much complexer but for a suitable restriction of labelled equations we get analogous results.

From a specification point of view conditional equations appear as application conditions: the application of a rule is constrained to those graph structures where the derived structure is consistent again. A number of examples (Communication Network, Actor System, Filesystem) demonstrate that the underlying concept of consistent graph transformation is of considerable practical interest.

3 Specification and Programming

A Programming Language for Distributed Systems

Klaus Barthelmann, Georg Schied
Universität Mainz, Universität Erlangen-Nürnberg

A distributed system can be depicted as a graph in a natural way. Graph rewrite rules precisely model the local changes in such a system. Therefore, we designed and presented a small programming language with a formal semantics based on graph rewriting.

DHOP (Distributed Higher Order Processes) is an applicative language with parameterized process definitions as the basic building blocks. Processes and channels between them are created at runtime. Furthermore, processes and channels can be exchanged in messages. A process selects a synchronous communication among several possible ones.

The specification of the semantics is based on the double pushout approach to graph rewriting. We use an appropriate kind of labelled graphs defined as a comma category of graphs and algebras. This is a good trade-off between hierarchical graphs and equational term rewriting, to both of which it is equivalent in the present case. The advantage lies in a strict separation of interprocess communication from local computations.

This language, however small, is already powerful enough to specify complex abstract data types. On the other hand, a complete semantic specification comprises only a few simple rewrite rules. It is also easy to derive an efficient implementation from this description: Context-sensitive rules should be split and amalgamated, and the associated event structure gives the control component.

A Computational Model for Generic Graph Transformations

Marc Gemis, Jan Paredaens, Peter Peelman[1], *Jan Van den Bussche*[1]
University of Antwerp (UIA), Belgium

The *generic graph machine*, a Turing machine-like computational model for generic graph transformations, is introduced. A configuration of this machine consists of a

[1]*Supported by the impulsprogram "informatietechnologie" of the "Diensten voor Programmatie van het Wetenschapsbeleid", nr. IT/IF/13.*

number of machine instances that each are in a state and point to two nodes of the graph. During the execution of a step, in parallel the machine instances perform a local transformation on the graph and are replaced by zero or more other machine instances. It is proved that the generic graph machines have the same expressive power as a large class of natural object-creating database languages. Finally, we prove that a generic graph machine is polynomially reducible to a Turing machine.

Jungle Rewriting as a Specification Tool for an Abstract Narrowing Machine

Andrea Corradini, Dietmar Wolz
Università di Pisa, Technische Universität Berlin

The Narrowing Calculus is a powerful rewriting formalism which subsumes both Conditional Term Rewriting and Logic Programming: it consists essentially of term rewriting with unification. The LANAM (Lazy Abstract Narrowing Machine [Wo91]) is an efficient implementation of narrowing developed by the second author, which exploits compilation techniques borrowed from the implementations of both lazy functional and logic programming languages.

We present a top-down specification of the operational behaviour of the LANAM using a graph rewriting formalism called Jungle Rewriting. Jungles are directed hypergraphs that are suitable to represent collections of terms with possibly shared subterms (they are equivalent to DAGs), and jungle rewriting is defined as the double pushout approach in the category of jungles.

For each narrowing program $\boldsymbol{P}$ we define three jungle rewrite systems $\mathcal{J}(\boldsymbol{P})$, $\mathcal{F}(\boldsymbol{P})$, and $\mathcal{S}(\boldsymbol{P})$, which model $\boldsymbol{P}$ at different levels of abstractions. $\mathcal{J}(\boldsymbol{P})$ is a (pure) hyperedge-replacement jungle rewrite system, including one rule for each clause of $\boldsymbol{P}$: a theorem which extends a similar result for term rewriting systems and logic programming [CR93] states that the computations of $\mathcal{J}(\boldsymbol{P})$ correspond one-to-one with narrowing derivations. $\mathcal{F}(\boldsymbol{P})$ is a jungle rewriting system with applicability conditions, and consists of many rules for each clause of $\boldsymbol{P}$. These rules have to satisfy a strong injectivity requirement which, together with the applicability condition, make explicit the unification steps which were hidden in the application of rules of $\mathcal{J}(\boldsymbol{P})$. Finally, $\mathcal{S}(\boldsymbol{P})$ is obtained from $\mathcal{F}(\boldsymbol{P})$ by marking the rules in a suitable way. Markings are used, like in DACTL [GKSS88], to specify a control strategy, needed to constrain the nondeterminism of the system. In this way, we are able to specify the lazy evaluation strategy of the LANAM. As a result, the behaviour of system $\mathcal{S}(\boldsymbol{P})$ models in a faithful way the actual behaviour of the LANAM.

[CR93] A. Corradini, F. Rossi, *Hyperedge Replacement Jungle Rewriting for Term Rewriting Systems and Logic Programming*, to appear in Theoretical Computer

Science, 1993.
[GKSS88] J.R.W. Glauert, J.R. Kennaway, M.R. Sleep, G.W. Sommer, *Final Specification of DACTL*, Report SYS-C88-11, School of Information Systems, University of East Anglia, Norwich, UK.
[Wo91] D. Wolz, *Design of a Compiler for Lazy Pattern Driven Narrowing*, Proc. of the 7th Int. Workshop on the Specification of Abstract Data Types, Springer LNCS 534, 1991.

Guaranteeing destructive updatability through a type system with uniqueness information for graphs

Sjaak Smetsers, Erik Barendsen, Marko van Eekelen, Rinus Plasmeijer
Univ. of Nijmegen, NL

We have presented a type system related to linear typing: "uniqueness typing". The uniqueness type system is defined for graph rewrite systems. It employs usage information to deduce whether an object is "unique" at a certain moment, i. e. it is only locally accessible. In a type of a function it can be specified that the function requires a unique argument object. The correctness of type assignment guarantees that no external access on the object will take place in the future. This information can be used in the implementation to use the unique object destructively. The presented type system is proven to be correct. It can be used in coexistence with the standard type systems for functional languages such as the Hindley-Milner-Damas approach. The system is implemented in the functional graph rewriting language Concurrent Clean and is used for the efficient implementation of I/O and Arrays. The system can also be used to increase the efficiency of algorithms. We illustrate the power of the system by defining an elegant quicksort algorithm that performs the sorting in situ on the data structure.

1st Order Logic Based Graph Rewriting Systems with Application Conditions and Embedding Rules

Andy Schürr
RWTH Aachen

This talk presented the theoretical background of a project which is concerned with the development of an integrated set of tools for editing, analysing, and executing programmed graph rewriting systems (also presented here at Schloß Dagstuhl). Before realizing such a set of tools one has to define precisely the (concrete) syntax as well as the static and dynamic semantics of an appropriate notation for

PROgrammed Graph Rewriting Systems. This notation, termed PROGRES, is a kind of "very high level" programming language for modelling graphs and for specifying graph transformations. It contains features like complex embedding rules, application conditions, attribute equations etc. which are very useful within many application areas.

In order to be able to specify the semantics of all language constructs, it was necessary to develop a new type of graph rewriting system which

1. represents graphs and graph properties by sets of 1st order logic formulas,
2. uses relations as (sub-)graph morphisms,
3. verifies pre- and postconditions of rewrite rules by means of nonmonotonic reasoning,
4. and reduces the application of rewrite rules to the well-known construction of commuting diagrams.

Syntax and Semantics of Hybrid Database Languages

Gregor Engels, Marc Andries
Leiden University

We present the hybrid query language HQL/EER. It is a database query language for an Extended Entity-Relationship (EER) model, where a user can freely choose between graphical and textual specification of a part of a query. HQL/EER is an extension of the purely textual query language SQL/EER, the syntax and semantics of which has been defined in [1].

Syntax and semantics of this partially graphical query language HQL/EER are defined by the usage of PROGRES (see the talk of Andy Schürr at this seminar), which is a specification language based on programmed graph rewritings. We illustrate that a PROGRES specification is a well-suited means to define the syntax of such a language. In order to define the semantics of HQL/EER, the attribute evaluation mechanism of PROGRES is used to translate a HQL/EER query into a textual SQL/EER query.

[1] U. Hohenstein, G. Engels: SQL/EER—Syntax and Semantics of an Entity-Relationship-Based Query Language. Information Systems, 17(3):209–242, 1992

Semantics of Full Statecharts Based on Graph Rewriting

A. Maggiolo-Schettini and A. Peron
Dipartimento di Informatica, Università di Pisa, Italy

The formalism of Statecharts, proposed by Harel, offers interesting description facilities for reactive systems such as hardware components, communication networks, computer operating systems. Statecharts have the visual appeal of formalisms such as state diagrams and Petri nets, but, with respect to them, they offer also facilities of hierarchical structuring of states and modularity which allow high level description and stepwise development. Harel, Pnueli, Schmidt and Sherman have proposed a semantics of Statecharts in terms of steps, where a step is a set of consistent transitions from a given system configuration, which are enabled under a given external stimulus. Huizing, Gerth and de Roever have proposed a denotational semantics. In the present talk an operational semantics using graph rewriting is proposed. It is shown how to translate a statechart into a set of graph productions and how to describe the step from a configuration of the statechart to another in terms of a derivation from a graph describing the start configuration to a graph describing the reached configuration. Steps can be expressed in terms of sequences of derivations equivalently w.r.t. the semantics of Harel, Pnueli, Schmidt and Shermn. The sequential behaviour of the represented statechart is described by the graph of derivations. The non-sequential behaviour of Statecharts, described by graph rewriting, can be investigated by considering computations which are analogues of processes in the sense of Kreowski and Wilharm. Computations give a manner of mapping a partial order of transitions of the statechart onto the derivation graph. This offers a partial order semantics for Statecharts.

Uniform Modelling Using Graph Grammar Specs

Manfred Nagl
RWTH Aachen

The talk sketches the experience in conceptual modelling we got so far by building complex, integrated, interactive systems. These systems are built on top of data types for various graphs, as we regard all internal information to be a graph.

The first approach called AST graph modelling is to compose graphs from a tree (AST) with additional edges (e.g. for context sensitive relations). A big part of the structure and of the handling of these graphs can be described declaratively by a scheme. Further more, common patterns of tree and non-tree handling can be extracted to build up a basic spec.

Whereas AST graphs are flat, we regarded in recent times hierachical graph modelling. As the main idea is to think about general concepts of modelling their similarities and their relations, we called this approach metamodelling. A clean hierarchy for modelling entities was build up. The third idea is to compose a specification by layers where on each level a submodel is fixed. This has been successfully used for integration tools in the software engineering area.

In all cases reuse on specification level is our main concern. Reuse can take place on product level (using given specs) or on process level (knowledge how to build up specs). Specs are used in our group as a basis for efficient implementations.

4 Generation & Recognition of Graph Languages

Annotated Collage Grammars

Annegret Habel
Universität Bremen

Collage Grammars are a graph-grammatical device for generating classes of patterns. The key structures are collages consisting of

- a set of parts being geometric objects,
- a set of hyperedges (attached to some points) being subjects of replacement,
- pin points influencing the replacement.

Replacement of a hyperedge e by a collage R may be done if there is a transformation t from a given group of transformations which maps the pin points of the collage R to the attachment points of the hyperedge e. For avoiding the search of a suitable transformation, each hyperedge in a collage may be equipped with some transformation information. This yields to the concept of an annotated collage being the key structure of annotated collage grammars.

It is shown that given a collage grammar generating the collage language L, we can effectively construct an annotated collage grammar generating the same language.

Graphs & Designing

Ewa Grabska
Institute of Computer Science, Jagiellonian University, Cracow, Poland

This talk presented an attempt to build up a systematical theory for graphical modeling. The work should be seen as a part of a thorough study of theory for understanding design, which is necessary for building of intelligent CAD tools.

In my opinion the graph methods of processing, representation and generation of pictures and patterns applied today allow for introduction of certain innovation.

The innovation introduced by me is defined for graphs of the so called realization scheme. It is a kind of mapping which assigns a graphical model of an object to the graph representing the structure of this object. Thanks to this notion one graph

can have several different graphical models associated with it through different realization schemes.

The fact that the structure of the object is independent of its realization may be useful in the process of designing.

Path-Controlled Graph Grammars for Multiresolution Image Processing and Image Analysis

Kunio Aizawa, Akira Nakamura
Hiroshima Univ., Meiji Univ., Japan

The graph structure is a strong formalism for representing pictures in syntactic pattern recognition. In this talk, we define a subclass of nPCE graph grammars (L(1)-nPCE$_4$ graph grammars) and present a parsing algorithm of $O(n)$, where n is the number of nodes of the input graph. The algorithm is a deterministic topdown syntax analyzer applied to one-dimensional descriptions of input graphs.

Then we treat a graph construction technique on the graphs generated by the grammars. It enables multiresolutional image processing on graphs. A multiresolution graph is a maximal block representation in which the blocks have standard size (power of two). But unlike with the quadtrees, the blocks need not have standard positions. So it has more flexibility on image compression than quadtrees. Most of the basic image processing algorithm on such graphs require time proportional to the number of nodes or edges in the compressed graphs rather than that of original graphs.

Context-Free Vertex Replacement Sets of Graphs that are not Hyperedge Replacement: Characterization and Decidability

Bruno Courcelle[2]
Bordeaux – I University, France

We establish that a VR ("Vertex Replacement") set of graphs, i.e., a set of graphs generated by a C-edNCE or, equivalently, by a separated handle rewriting graph grammar is HR ("Hyperedge Replacement"), i.e., is generated by a hyperedge replacement graph grammar, iff its graphs do not contain arbitrary large complete bipartite graphs $K_{n,n}$ as subgraphs. Another equivalent condition is that its graphs

[2]*Supported by the ESPRIT Basic Research Working Group "COMPUGRAPH II" ("Computing by graph transformation") and by the "Programme de Recherches Coordonnées: Mathématiques et Informatique".*

have a number of edges that is linearly bounded in terms of the number of vertices. These properties are decidable by means of an appropriate extension of the theorem by Parikh that characterizes the commutative images of context-free languages. We extend these results to hypergraphs.

This work has been accepted for publication in *Information and Computation* (under the title: "Structural properties of context-free sets of graphs generated by vertex replacement"), and is scheduled to appear in 1994.

Decomposability helps for deciding logics of knowledge and belief

Stefan Arnborg[3]
The Royal Institute of Technology, Stockholm, Sweden

We show that decision problems in modal logics (logics of knowledge and belief) are easy for decomposable formulas. Satisfiability of a formula of size n and tree-width k can be decided in time $O(nf(k))$, where f is a double exponential function. This result holds not only for the logics S5 and KD45 with NP-complete decision problems, but also for extensions to multiple agents and arbitrary characterizations of Kripke semantics by any combination of the properties serial, symmetric, reflexive, transitive and Euclidean for the possibility relation between possible worlds (*i.e.*, in particular for the standard logics K_n, T_n, $S4_n$, $S5_n$ and $KD45_n$, whose decision problems are PSPACE complete for arbitrary formulas). Moreover, the method works for these logics extended with operators for distributed and common knowledge, which otherwise cause a complexity increase to exponential time for the satisfiability problem.

On Recognizable Sets of Graphs of Bounded Tree-Width

Jens Lagergreen
The Royal Institut of Technology, Stockholm, Sweden

We establish a set of finite graphs of tree-width at most k is recognizable (with respect to the algebra of graphs with an unbounded number of sources) *if and only if* it is recognizable with respect to the algebra of graphs with at most k sources. We obtain a somewhat stronger result for sets of simple finite graphs of tree-width at most k.

[3]*Supported by TFR and NUTEK.*

Confluent graph grammars with embeddings of depth k

Franz J. Brandenburg
Universität Passau

We investigate linear and confluent graph grammars with embeddings of depth k, LIN-edNCE$_k$ and C-edNCE$_k$. These are specializations of types of graph grammars introduced by M. Nagl and are a generalization of the class of 'context-free' graph grammars, C-edNCE, which have depth 1.

While usual C-edNCE graph grammars operate purely top down, depth k graph grammars with $k \geq 2$ permit a top down and a bottom up construction of graphs. E.g., they are capable to generate the (partial) transitive closure. Thus, C-edNCE$_k$ graph grammars introduce new features for the generation of graphs, which makes them entirely different from usual C-edNCE graph grammars.

As our first results, we can show that depth two is sufficient and that linearity and confluence are decidable. Moreover, linear depth two graph grammars can generate the sets of all trees and of all graphs, and they can even generate the set of all square grid graphs, if blocking edges are used. This sharply contrasts the generative capacity of usual C-edNCE graph grammars and demonstrates the gain in power by embeddings of depth 2. The limits are not yet clear.

Graph automata for connected L-eNCE graph grammars

Konstantin Skodinis
Universität Passau

We define marking automata (EMA), which generate the same languages as connected linear e-NCE graph grammars. The automaton consists of a finite control and of marking pairs. Each marking pair consists of a right part, a left part and a boundary set, where the edge-labels may be encoded. Given a connected input graph H the EMA reconstructs the derivation process of H. It places H's marking pairs on the edges between the already visited and the not yet visited part of the input graph.

In state p the automaton reads a node v of H, tests the compatibility between v and the local environment of H, which is defined by the marking pairs, updates its marking pairs and enters a new state q. The EMA stops, if the transition function cannot be applied by a step. It accepts, if it has reached the final stop state.

The following theorems hold:

1. If G is a connected L-eNCE graph grammar, then there exists an edge marking automaton EMA, such that $L(G) = L(EMA)$, and

2. if *EMA* is a edge marking automaton, then there exists a connected L-eNCE graph grammar G, such that $L(EMA) = L(G)$.

Combining Programmed Attributed Graph Grammars and Two-level Graph Grammars for the Design of an Object Oriented Database for Picasso Pictures

Herbert Göttler, Bernd Himmelreich
Universität Mainz

Inspired by the questions why Picasso was such a productive artist and what makes up the typical style of cubistic art, G. König, professor of art at the University of Mainz, extracted a formal picture language (which is a relatively small set of pictorial operations) for a subset of Picasso's œvre by analizing hundreds of examples. The lecture reports on the design of an object-oriented database to aid these activities and to implement the pictorial operations. A further goal is an expert system for Picasso, and a slightly utopic one is a tool for creating pictures in a 'true Picasso style'.

For the design of the structure of the database and for its operations a model based on graphs and graph grammars seems to be well-suited. Since we want to exploit the advantages of the object oriented approach—especially the concept of inheritance—we think we have to combine programmed attributed graph grammars with two-level graph grammars.

Combinatorial Generation of k-Paths

Andrzej Proskurowski
University of Oregon

A generalization of paths, k-paths are maximal graphs of proper pathwidth k. We present methods of listing all non-isomorphic k-paths as k-strings inequivalent with respect to string reversal and permutation of symbols. The resulting algorithms use time proportional to the number of generated strings. (Joint work with F. Ruskey and M. Smith)

Dagstuhl-Seminar 9301:

List of Participants (update: 28.03.93)

Kunio **Aizawa**
Hiroshima University
Department of Applied Mathematics
1-4-1 Kagamiyama
Higashi-Hiroshima 724
Japan
aizawa@huis.hiroshima-u.ac.jp
tel.: +81-824-22-71 11/32 84

Stefan **Arnborg**
Royal Inst. of Technology
NADA KTH
S-10044 Stockholm
Sweden
stefan@nada.kth.se
tel.: +46-8-790-71 94

Erik **Barendsen**
University of Nijmegen
Faculty of Mathematics and
Computer Science
Toernooiveld 1
NL-6525 ED Nijmegen
The Netherlands
erikb@cs.kun.nl
tel.: +31-80-65 26 46

Klaus **Barthelmann**
Universität Mainz
Fakultät Math./Informatik
Staudinger Weg 9
W-6500 Mainz
Germany
barthel@informatik.mathematik.uni-mainz.de
tel.: +49-6131-39-36 15

Michel **Bauderon**
Université Bordeaux I
Dépt. d'Informatique
351 Cours de la Libération
F-33405 Talence
France
bauderon@geocub.greco-prg.fr
tel.: +33.56.84.69.07

Franz-Josef **Brandenburg**
Universität Passau
Lehrstuhl für Informatik
Innstr. 33
W-8390 Passau 1
Germany
brandenb@informatik.uni-passau.de
tel.: +49-851-509-3 43

Andrea **Corradini**
Università di Pisa
Dpt. Informatica
Corso Italia 40
I-56100 Pisa
Italy
andrea@di.unipi.it
tel.: +39-50-510-2 42

Bruno **Courcelle**
Université Bordeaux I
Lab. d'Informatique
351 Cours de la Libération
F-33405 Talence
France
courcell@geocub.greco-prog.fr
tel.: +33.56.84.60.86

Jan **Cuny**
University of Massachusets
Computer & Information Sciences
Amherst MA 01003
USA
cuny@cs.umass.edv
tel.: +1-413-545 4228

Hartmut **Ehrig**
TU Berlin
Inst. f. Software u. Theoret. Informatik
FB 20 FR 6-1
Franklinstr. 28/29
W-1000 Berlin 10
Germany
ehrig@cs.tu-berlin.de
tel.: +49-30-314-7 35 10

Gregor **Engels**
University of Leiden
Department of Computer Science
Niels Bohrweg 1
NL-2300 RA Leiden
The Netherlands
engels@rulwi.leidenuniv.nl
tel.: +31-71-27 70 69

Herbert **Göttler**
Universität Mainz
Fakultät Math./Informatik
Staudinger Weg 9
W-6500 Mainz
Germany
goettler@uaimza.mathematik.uni-mainz.de
tel.: +49-6131-39 33 36

Ewa **Grabska**
Univ. Jagiellonski
Inst. Informatyki
ul. Nawojki 11
PL-30-072 Krakow
Poland
uigrabsk@plkrcyii

Annegret **Habel**
Universität Bremen
FB Math./Informatik
Postfach 33 04 40
W-2800 Bremen 33
Germany
habel@informatik.uni-bremen.de
tel.: +49-421-218-34 89

Dirk **Janssens**
Free University of Brussels
Dept. of Computer Science
Pleinlaan 2
B-1050 Brussels
Belgium
dnjans@tinf1.vub.ac.be
tel.: + -2-641-34 87

Yasuo **Kawahara**
Kyushu University 33
Research Institute of Fund.
Information Science
Fukuoka 812
Japan
kawahara@rifis.sci.kyushu-u.ac.jp
tel.: +81-92-641-11 01

J. Richard **Kennaway**
University of East Anglia
School of Information Systems
Norwich NR4 7TJ
Great Britain
jrk@sys.uea.ac.uk
tel.: +44-603-59 32 12

Martin **Korff**
TU Berlin
Fachbereich 20 Informatik
Franklinstr. 28-29
W-1000 Berlin 10
Germany
martin@es.tu-berlin.de
tel.: +49-30 314-2-7787

Hans-Jörg **Kreowski**
Universität Bremen
FB Math./Informatik
Postfach 330440
W-2800 Bremen 33
Germany
kreo@informatik.uni-bremen.de
tel.: +49-421-218-29 56

Michael **Löwe**
TU Berlin
Inst. f. Software u. Theoret. Informatik
FB 20 FR 6-1
Franklinstr. 28/29
W-1000 Berlin 10
Germany
loewe@ cs.tu-berlin.de
tel.: +49-30-314-2 58 13

Jens **Lagergreen**
Royal Inst. of Technology
NADA KTH
S-10044 Stockholm
Sweden
jensl@nada.kth.se

Andrea **Maggiolo-Schettini**
Università di Pisa
Dipartimento di Informatica
Corso Italia 40
I-56125 Pisa
Italy
maggiolo@di.unipi.it
tel.: +39-50-51 02 59

Ugo **Montanari**
Università di -Pisa
Dpt. Informatica
Corso Italia 40
I-56100 Pisa
Italy
ugo@di.unipi.it
tel.: +39-50-51 02 21

Manfred **Nagl**
RWTH Aachen
Lehrstuhl für Informatik 3
Ahornstr. 55
W-5100 Aachen
Germany
nagl@rwth3.informatik.rwth-aachen.de
tel.: +49-241-80 72 80 / 80 21 300

Jan **Paredaens**
University of Antwerpen
Dept. of Math. and Computer Science
Universiteitsplein 1
B-2610 Antwerpen
Belgium
pareda@ccu.uia.ac.be
tel.: +32-3-820-24 09

Francesco **Parisi-Presicce**
Universitá degli Studi di l'Aquila
Dipt. di Mathematica
Via Vetoio
I-67100 L'Aquila
Italy
parisi@vxscaq.aquila.infn.it
tel.: +39-862-43 31 27

M.J. **Plasmeijer**
University of Nijmegen
Faculty of Mathematics and
Computer Science
Toernooiveld 1
NL-6525 ED Nijmegen
The Netherlands
rinus@cs.kun.nl
tel.: +31-80-65 26 44

Andrzej **Proskurowski**
University of Oregon
Dept. of Computer Science
Eugene OR 97403-1202
USA
andrzej@cs.uoregon.edu
tel.: +1-503-346-44 08

Jean-Claude **Raoult**
Université de Rennes I
IRISA
Campus de Beaulieu
Avenue du Général Leclerc
F-35042 Rennes Cedex
France
raoult@irisa.fr
tel.: +33-99 84 72 78

Francesca **Rossi**
Università di Pisa
Dipartimento di Informatica
Corso Italia 40
I-56125 Pisa
Italy
rossi@di.unipl.it
tel.: +39-50-510268

Andy **Schürr**
Lehrstuhl für Informatik 3
RWTH Aachen
Ahornstr. 55
W-5100 Aachen
Germany
andy@rwthi3.informatik.rwth-aachen.de
tel.: +49-241-80 72 29

Georg **Schied**
Universität Erlangen
Lst. Programmiersprachen
Martensstr. 3
W-8520 Erlangen
Germany
schied@informatik.uni-erlangen.de
tel.: +49-9131-85 79 33

Hans-Jürgen **Schneider**
Universität Erlangen
Lst. Programmiersprachen
Martensstr. 3
W-8520 Erlangen
Germany
schneide@informatik.uni-erlangen.de
tel.: +49-9131-85-76 20

Konstantin **Skodinis**
Universität Passau
Lehrstuhl für Informatik
Innstr. 33
W-8390 Passau
Germany
skodinis@trillian.fmi.uni-passau.de
tel.: +49-851-509-7 79

Gabriele **Taentzer**
TU Berlin
Inst. f. Software u. Theoret. Informatik
FB 20 FR 6-1
Franklinstr. 28/29
W-1000 Berlin 10
Germany
gabi@cs.tu-berlin.de
tel.: +49-30-314-2 77 87

Egon **Wanke**
Universität-GHS-Paderborn
FB Informatik
Warburger Str. 100
W-4790 Paderborn
Germany
egon@uni-paderborn.de
tel.: +49-5251-60-30 74

P. M. **van den Broek**
Universiteit Twente
Faculteit der Informatica
Postbus 217
NL-7500 AE Enschede
The Netherlands
pimvdb@cs.utwente.nl
tel.: +31-53-89 37 62

Zuletzt erschienene und geplante Titel:

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, R. Kass, C. Paris, W. Wahlster (editors):
Third International Workshop on User Modeling (UM'92), Dagstuhl-Seminar-Report; 44; 10.-13.8.92 (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 for 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)

B. Courcelle, H. Ehrig, G. Rozenberg, H.J. Schneider (editors):
Graph-Transformations in Computer Science, Dagstuhl-Seminar-Report; 53; 04.01.-08.01.93 (9301)

A. Arnold, L. Priese, R. Vollmar (editors):
Automata Theory: Distributed Models, Dagstuhl-Seminar-Report; 54; 11.01.-15.01.93 (9302)

W.S. Cellary, K. Vidyasankar , G. Vossen (editors):
Versioning in Data Base Management Systems, Dagstuhl-Seminar-Report; 55; 01.02.-05.02.93 (9305)

B. Becker, R. Bryant, Ch. Meinel (editors):
Computer Aided Design and Test , Dagstuhl-Seminar-Report; 56; 15.02.-19.02.93 (9307)

M. Pinkal, R. Scha, L. Schubert (editors):
Semantic Formalisms in Natural Language Processing, Dagstuhl-Seminar-Report; 57; 23.02.-26.02.93 (9308)

H. Bibel, K. Furukawa, M. Stickel (editors):
Deduction , Dagstuhl-Seminar-Report; 58; 08.03.-12.03.93 (9310)

H. Alt, B. Chazelle, E. Welzl (editors):
Computational Geometry, Dagstuhl-Seminar-Report; 59; 22.03.-26.03.93 (9312)

J. Pustejovsky, H. Kamp (editors):
Universals in the Lexicon: At the Intersection of Lexical Semantic Theories, Dagstuhl-Seminar-Report; 60; 29.03.-02.04.93 (9313)

W. Straßer, F. Wahl (editors):
Graphics & Robotics, Dagstuhl-Seminar-Report; 61; 19.04.-22.04.93 (9316)

C. Beeri, A. Heuer, G. Saake, S.D. Urban (editors):
Formal Aspects of Object Base Dynamics , Dagstuhl-Seminar-Report; 62; 26.04.-30.04.93 (9317)