Report on the Dagstuhl-Seminar 9637

Graph Transformations in Computer Science

September 9 - 13, 1996

Organizers:

Hartmut Ehrig (Berlin)
Ugo Montanari (Pisa)
Grzegorz Rozenberg (Leiden)
Hans Jürgen Schneider (Erlangen)

The research area of graph transformations dates back to the early seventies. Its methods, techniques, and results have already been 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, and so on. This wide applicability is due to the fact that graphs are a very natural way to explain complex situations on an intuitive level. Graph transformation brings dynamics to all these descriptions and can describe the evolution of structures. Therefore graph transformation has become attractive as a programming and specification 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. The operational approach immediately formalizes a specific type of graph transformation and may be implemented efficiently. On the other hand, the categorical approach yields attractive formal properties. In recent years this approach was extended to high-level replacement systems and was modified by replacing the double-pushout construction both by a single-pushout construction and by a pullback construction. The evolution of the field and the state-of-the-art are documented in six volumes of Lecture Notes in Computer Science (73, 153, 291, 532, 776, and 1073).

During this Dagstuhl-Seminar, 35 lectures were presented by the participants from seven European countries, Brazil, Israel, and Japan on foundations as well as on applications. Comparing the abstracts in this booklet with those of the previous Dagstuhl-Seminar on this topic (Report No. 53), we observe a clear shift to the new aspects of foundations mentioned above and to specification of concurrent and distributed systems. System demonstrations showed the improvement of efficient implementations and let us expect that the gap between formal methods in system development and graphical methods can be bridged over by graph-transformation based methods.

Altogether the seminar led to fruitful discussions between theory and application and between advocates of different approaches. The participants appreciated the stimulating atmosphere in Schloß Dagstuhl.

Finally, I would like to express my warmest thanks to the co-organizers; unfortunately two of them, Hartmut Ehrig and Grzegorz Rozenberg, could not attend the seminar because of sickness; nevertheless, they also have contributed to the success of the seminar by their co-operation in preparing it.

October 1996 Hans J. Schneider

Contents

A Simple Graph Grammar Enabling the Representation of Layered Graphs as "Pages" over Words Generated by the Grammar

Azaria Paz

Technion Haifa, Israel

Interconnection networks are useful structures used in many applications such as packet routing schemes, telephone switching networks, parallel computation architectures etc. In the past several years Even and Litman introduced a new technique, the "layered cross product" technique. They showed that several well known interconnection networks can be constructed by this technique from simple layered graphs. We present a simple graph grammar over a 3 letters alphabet having the property that when the words generated by the grammar are combined into "pages" according to a simple given rule, the resulting pages can represent layered graphs, including all layered graphs which can be generated by the layered cross product technique of Even and Litman. Moreover the graph grammar representation enables a decomposition technique for layered graph which can be split, by this technique, into primitive cross product components.

Aspects of Term Graph Homomorphisms

Wolfram Kahl

Universität der Bundeswehr München, Germany

The main topic of this talk were second-order term graphs, i.e., term graphs with explicit variable binding, explicit variable identity and with metavariables of arbitrary arities. Rewriting on these term graphs emulates Klop's Combinatory Reduction Systems (CRSs), a term formalism generalizing λ-calculus.

The aim then was to procure some familiarity with and insight into the intrinsic structure of these second-order term graphs. To this purpose, the talk concentrated on an appropriate notion of homomorphisms that

1) avoids "capture of variables" in analogy to the valuation mechanism in CRSs,

2) ensures compositionality of at least certain interesting subsets of homomorphisms.

These homomorphisms can then be used to form a foundation for an appropriate **algebraic approach to second-order term graph rewriting** that also encompasses graph reduction, but we did not delve into the details of the whole rewriting approach.

Just a few of the more involved homomorphism conditions were presented, since these most directly give a feeling for the differences between terms and term graphs on one hand, and between first-order and second-order term graphs on the other hand. These conditions are:

Variable control: No node of a variable occurring free within the image of a metavariable is the image of a bindable variable.

Encapsulation unity: If any node not encapsulated in G_1 has an image that is encapsulated by another node r, then the images of all nodes in G_1 are reachable from r.

Encapsulation consistency: For any two unencapsulated nodes x and y in G_1, no variable in G_2 occurs free under the image of x, but only bound under the image of y (while being reachable over a non-selfencapsulating path).

In terms (or term trees, as opposed to general term graphs) these conditions are ensured to always hold either by virtue of the way binding is coded by name and scope and from the definition of substitutions, or by virtue of the absence of sharing, of rootedness or of acyclicity. Indeed, the last condition is only relevant in the presence of cycles, where special care has to be taken with the definition of the images of metavariables, which have to stop not only at the images of their successors, but also at "self-encapsulation borders".

For a proper treatment of second-order term graphs, furthermore extensions of the conventional term graph homomorphism concept as total node mappings are necessary: certain partial mappings have to be considered, and edges leaving metavariables may have to be mapped to sets of edges.

Although quite involved in the technical details, the second-order term graph homomorphisms discussed in this talk still have a direct intuitive appeal and the complicated homomorphism conditions serve very well to give us a much more direct grip on the structural properties of variable binding.

The Higher Object Programming System HOPS

(System Demonstration)

Wolfram Kahl

Universität der Bundeswehr München, Germany

The **H**igher **O**bject **P**rogramming **S**ystem HOPS, which has been developed by a group led by Gunther Schmidt since the mid-eighties, is a graphically interactive term graph editing and transformation system designed for transformational program development.

HOPS manipulates arbitrary second-order term graphs with nameless variables, explicit variable binding, explicit variable identity and metavariables with arbitrary arity.

Although usually the user will start from a functional programming paradigm, in principle the current implementation is language independent.

The most notable features of HOPS are:

Term DAGs at the user interface, and not only as an internal representation — this makes program structure more immediately accessible to the user and offers sharing in term graphs as a separate abstraction principle,

Literate programming with embedded term graphs, where programs are considered as documents where the code (in the shape of term graphs serving the purposes of declarations, transformation rules and example graphs) is interspersed between documentation prose, and both can be edited from the same interface,

Strong online term graph typing that not only serves to make typing mistakes impossible, but also guides program development,

Interactive transformation facilities that can be used at any stage of program development and bring about a great flexibility,

Automatic transformation support for speeding up mechanical transformation tasks.

Rewriting Partial Graphs

Gabriel Valiente

Technical University of Catalonia, Spain
joint work with Francesc Rosselló
University of the Balearic Islands, Spain

Rewriting of partial graphs using total and partial morphisms is put in the unified framework of algebraic transformation of unary partial algebras. Explicit constructions for several variants of the double-pushout and the single-pushout transformation of partial graphs are given in detail.

A relationship between rewriting of total graphs and rewriting of partial graphs is established by means of free completions, which preserve both double-pushout and single-pushout derivations. Given a double-pushout or single-pushout direct derivation of total graphs, a corresponding direct derivation of partial graphs can be found such that the resulting total graph is the maximal free completion of the resulting partial graph. Moreover, any double-pushout or single-pushout direct derivation of partial graphs can be uniquely extended to a corresponding direct derivation of total graphs.

Maximal free completions allow a more efficient rewriting of total graphs by rewriting of corresponding partial graphs and completion of the resulting partial graph. Sufficient conditions are also given for a class of completions of partial graphs to preserve double-pushout and single-pushout derivations.

Efficient Graph Rewriting and Its Implementation

Heiko Dörr

Daimler-Benz AG, Germany

At first glance, efficient graph rewriting is a contradiction in terms. Before a graph rewriting rule can applied, an appropriate subgraph of the graph to be transformed must be found. By definition of rewriting, this subgraph must be

isomorphic to the rule's left-hand side. Hence each rewriting step requires the solution of the isomorphic subgraph problem, which is NP-complete.

Our work faces this shortcoming on a theoretical and practical level. First of all, we develop a sufficient condition under which the isomorphism problem for graphs and rules of a given rewriting system con be solved in constant time. The condition can be evaluated by static analysis of the rewriting system. The evaluation firstly performs an abstract interpretation of the system. It approximates the sets of unique vertex labels and so-called strong V-structures of the language defined by the rewriting system. Secondly, the evaluation of the condition tries to find for each rule a search strategy appropriate for the isomorphism problem. That strategy must initiate the search in a uniquely labeled vertex and bypass any potential strong V-structure. If there is such a strategy for each rule of a rewriting system, the system belongs to the class of so-called UBS graph rewriting systems and the isomorphic subgraph problem is solvable in constant time.

On the practical level we develop an environment for programmed and attributed graph rewriting systems. It consists of a compiler and an abstract machine for graph rewriting. On the basis of a denotational semantics for control structures we design an optimization for rule sets. It speeds up the subgraph isomorphism test for a set of applied rules by reuse of previous results. The abstract machine supports the optimized application test for rule sets. It can furthermore execute whole programmed attributed graph rewriting systems.

Parallel Composition of Graph Grammars

Leila Ribeiro

Universidade Federal do Rio Grande do Sul, Brazil

The specification of complex systems is usually done by the "divide and conquer" idea: the system is divided into smaller, less complex components that are developed separately and then merged in some way to form the specification of the whole system. A suitable formalism that supports such a development shall assure that the composition operators used to merge the component specifications are compatible with the semantics of the system. Compatibility here means that the behaviour of the whole system can be derived from the behaviours of its components, that is, the composed system does not shows a behaviour that is not specified in any of its components.

The main aim of this paper is to provide an approach to the parallel composition of graph grammars. The parallel composition presented here formalizes the intuitive idea of divide and conquer described above: an abstract description of a system is divided into components that are further specialized and then merged together to form the specification of the whole system. The important requirement is that each component is a kind of conservative extension of the abstract description of the system, in the sense that the specialization of the abstract view defined in the component does not imply that the behaviour of the abstract level would change. Such specializations are formalized by special graph grammar morphisms. These morphisms are not only interesting to describe specializations (refinements) of grammars, but also to express structural and behavioural compatibilities between graph grammars.

Summarizing, the new concept of *parallel composition* of graph grammars presented here has the following characteristics:

- The *initial (start) graph* is taken into account;
- The composition of two grammars can be based on a shared part (*cooperative parallel composition*), or be a composition without any shared parts (*pure parallel composition*);
- The composition is based on *specialization morphisms*. These morphisms express the fact that both components to be composed are specializations of the shared parts;
- The result of the composition is suitably *syntactically and semantically related* to the component grammars;
- The parallel composition is *compositional* with respect to a true concurrent semantics of graph grammars, namely the unfolding semantics.

A View-Based Approach to System Modelling

Gregor Engels

Leiden University, The Netherlands
joint work with H. Ehrig, R. Heckel, G. Taentzer (TU Berlin),
A. Corradini (Univ. of Pisa)

In order to manage the complexity of large system specifications, they have to be decomposed into subspecifications. Each subspecification describes a certain part of the system. This might be a certain aspect, like the data, dynamic, or functional aspect, as it is known from object-oriented modelling techniques. Or it might be a certain view onto the system, as it is known from database modelling techniques. The talk motivates the usage of views in graph grammar-based specifications. First, the usage of typed graph grammars inherently ensures an integration of the data and the functional aspect within a view. Second, it is explained that it is not appropriate in case of views to have a fixed semantics. The standard fixed semantics, i.e. a graph transformation system, has to be relaxed to a loose semantics, i.e. a graph transition system. This reflects the idea that a view models only a part of the complete system. Other views may overlap a view with respect to data or functionality. A complete system specification is yielded by exploiting the approach of cooperative parallel composition of graph grammars (see talk by Leila Ribeiro).

Synchronization of Views and Loose Semantics of Typed Graph Productions

Hartmut Ehrig

Technische Universität Berlin, Germany
joint work with Reiko Heckel, Julia Padberg, Gabriele Taentzer,
Uwe Wolter (TU Berlin), Andrea Corradini (Pisa), Gregor Engels (Leiden)

The concept of views is used on two levels. First, so-called *design views* are developed for structuring specifications, that is, a system is modeled according to different views (e.g., representing the needs of different kinds of users) which

have to be synchronized afterwards in order to build the whole system. Views can be specified by means of typed graph transformation systems, where the type graph determines the visible types and the productions describe the known operations of that view. The *synchronization of views* is done by the construction of cooperative parallel composition of graph transformation systems, developed by Leila Ribeiro and presented at the same seminar.

If the specification is complete, a view may describe an observation of the system *in operation*. In this case we speak of a *user view*. It turns out that the semantics of such a view cannot be described by computations (i.e., graph transformations), but just by observations of computations of the global system. Such observations of computations cannot be represented by graph transformations in the usual sense because a local view may lack operations (productions) of the global system, so that state changes may be observed that do not have a cause in the local view.

Therefore, the notion of *graph transition* is introduced as loose semantics for productions, where the production specifies only a lower bound to the activities that are to happen during application. Contrastingly, in the classical double-pushout approach to graph rewriting, productions are interpreted as complete descriptions of the transformations to be performed.

For typed graph transformation systems a *transition sequence semantics* is developed, comprising all finite and infinite sequences of transitions in a system. Moreover, this semantics is shown to be compositional w.r.t. the synchronization of views.

Behavioral Constraints for Loose Graph Transformation Systems

Reiko Heckel

Technische Universität Berlin, Germany

In this lecture, the concept of synchronization of views presented at the same seminar in the framework of typed graph transformation systems with loose semantics is extended by *behavioral constraints*. Such constraints can be used to control the transformation process, to express properties of systems for their verification, or (what provided the initial motivation of this talk) to restrict the loose

semantics of productions. Examples of behavioral constraints include starting and ending graphs, application conditions for productions, static and dynamic integrity constraints, programmed graph transformations, etc.

In order to support a variety of behavioral constraints we develop a generic framework for behavioral constraints for typed graph transformation systems in the double-pushout approach. The framework, called *logic of behavioral constraints*, provides the main notions and results presented in the talk on synchronization of views and loose semantics of productions on an axiomatic basis. The techniques are motivated by the concepts of *logic of constraints* and *institutions* in the field of algebraic specification of abstract data types.

Known instances of logics of behavioral constraints include (so far) delete/create permissions for graph transitions, negative application conditions for productions, as well as static and dynamic integrity constraints expressed by temporal logic.

For any given logic of behavioral constraints, the synchronization by parallel composition of graph transformation systems as well as the transition sequence semantics extend to graph transformation systems with constraints. Moreover, the compositionality of the semantics w.r.t. the synchronization has been transfered to the extended setting.

The framework can be made approach independent if we assume a category of graph transformation systems (of whatever approach) such that a morphism of that category corresponds to a translation of the transformation steps in the source system to transformation steps in the target system. Then, a flat (unstructured) graph transformation system becomes comparable to a flat GRACE transformation unit, which could provide a new way of structuring transformation units, featuring refinement and synchronization in addition to the currently available use relation.

Graph Rewriting with Active Integrity Constraints

Andy Schürr

RWTH Aachen, Germany
joint work with Manfred Münch, Andreas Winter

Integrity constraints are a well-known means to define consistency conditions for data stored in any kind of information system. Knowledge bases or deductive

database systems usually offer predicate logic formulas for defining static integrity constraints which restrict the set of all legal data(base) states. But there are also many publications on Information System modeling which advocate the usage of temporal logic formulas or life cycle expressions. They support the definition of dynamic integrity constraints which restrict the set of all legal database state transition sequences. Last but not least there are so-called active database systems which associate conditions (constraints) with actions. These actions may update derived data (views) or recover a database from an inconsistent state.

Rather recently the graph grammar community developed a growing interest in combining the operational paradigm of programming with graph rewrite rules with the more declarative paradigm of constraint based programming. Both paradigms are now combined in the programmed graph rewriting system language and environment PROGRES. It offers predicate logic formulas and conditional graph patterns for the definition of static integrity constraints as well as pre- and postconditions for restricting the applicability of graph rewrite rules. Furthermore, it supports the definition of future directed dynamic integrity constraints based on testing the applicability of complex graph rewriting programs (which are similar to life cycle expressions of OO-modeling notations). All these types of constraints may have associated repair actions which transform an inconsistent graph into one which respects all defined integrity constraints.

The PROGRES system and its documentation is available as free software on the worldwide web:

http://www-i3.informatik.rwth-aachen.de/research/progres/index.html

Reengineering of Databases using Triple Graph Grammars

Albert Zündorf

Universität Paderborn, Germany

Object-oriented technology has become mature enough to satisfy many new requirements coming from areas like computer-aided design (CAD), computer-integrated manufacturing (CIM), or software engineering (SE). However, a competitive information management infrastructure often demands to merge data from CAD-, CIM-, or SE-systems with business data stored in a relational system. In addition, complex dependencies between those data stored in the different

systems might exist and should be maintained. One approach for seamless integration of object-oriented and relational systems is to migrate the data (and the corresponding schema) from a relational to an object-oriented system.

In this talk I describe an integrated design environment that supports the migration process and overcomes major drawbacks of comparable approaches. Our approach is based on the application of Triple Graph Grammars for the specification of the translation of (elements of) relational schemata to "equivalent" OO schemata (elements). From this specification we automatically derive an interactive schema design and translation environment.

Reference:

J. Jahnke, W. Schäfer, A. Zündorf: A Design Environment for Migrating Relational to Object Oriented Database Systems. To appear in Proc. 1996 International Conference on Software Maintenance (ICSM '96)

Distributed Graph Transformation With Rule-Based Split and Join Operations

Gabriele Taentzer

Technische Universität Berlin, Germany

Distributed Graph Transformation as presented in this talk combines structured graph transformation on two abstraction levels, the network and the local level, with the concept of synchronization by interface graphs. In this new approach, the main distribution concepts of categorical graph grammars developed by Schneider are combined with the algebraic approach to distributed graph grammars introduced by Ehrig et.al. Modeling distributed systems by this new kind of distributed graph transformation offers a clear description of dynamic networks, local and distributed actions such as synchronization based on graph transformation.

In this talk, we emphasize the split and the join operations which are special network changing actions. In contrast to previous definitions, the split and join operations are described in a rule-based way within the framework of distributed graph transformation. This leads to a constructive and deterministic formulation of these operations.

Distributed graph transformation is based on the double-pushout approach. For the formulation of split operations the original double-pushout approach is used which allows non-injectiveness in all parts of a production. A distributed transformation step is characterized by a double-pushout on distributed graphs and graph morphisms. Satisfying the so-called distributed gluing condition the applicability of distributed graph productions to distributed graphs and the uniqueness of such production applications is stated.

Distributed graph transformation has a close relationship to synchronization of views. The main differences are the global view which is not necessarily reflected in a distributed graph. Moreover, views are defined by distributed types whereas distributed graphs model distributed instances (of types).

How to Construct a Hyperedge Replacement System for a Context-free Set of Hypergraphs

Klaus Barthelmann

Universität Mainz, Germany

We give a new proof for the fact that a context-free set of simple hypergraphs can be generated by hyperedge replacement if the connectivity of its elements is bounded in some way. Measures for connectivity are the maximum degree, the tree-width, or the number of hyperedges relative to the number of vertices. It all boils down to the question whether or not arbitrarily large bipartite graphs are subgraphs of the elements. We show how to decide these preconditions without extra work.

Sets of hypergraphs are described as least solutions of polynomial systems of equations with certain operations. The collections of operations cause the difference in expressive power. The general one, our starting point, consists (mainly) of the quantifier-free definable operations and was introduced by Courcelle (1992). The second collection is suitable for hyperedge replacement; it traces back to Bauderon and Courcelle (1987). We translate a system of equations using the first into a larger one using the second under the conditions stated above. Our transformation is "direct" in the sense that the structure of the resulting system of equations reflects the structure of the original one. This distinguishes it from the approach of Courcelle (1995), which uses methods from automata theory, logic and graph theory, and employs several encodings in the transformation process.

Our proof is more general than its other precursors: Engelfriet and Heyker (1994) on the one hand and Engelfriet and Rozenberg (1990) followed by Brandenburg (1991) on the other, which transform sets of rewriting rules.

Concurrent Semantics for the ss-Calculus via Graph Rewritings

Marco Pistore

Università di Pisa, Italy
joint work with Ugo Montanari

In this lecture we equip the π-calculus with an operational semantics based on graphs and double-pushout rewritings. The π-calculus is a process algebra with the ability of handling channels as messages, thus modeling agents able to change their neighborhood. Graph rewriting systems happen to be a very convenient metalanguage for the π-calculus. They allow for instance for expressing the behavior of any particular agent with a finite number of productions — production schemata are required instead for the operational semantics of the whole language.

Concurrent semantics for the π-calculus can be extracted from the operational semantics on graphs. A (truly) concurrent semantics of a concurrent language aims at expressing, in addition to input-output behavior and temporal dependencies, other informations like causal dependencies of actions and spatial distribution of systems.

An *operational* concurrent semantics is obtained by applying the shift construction, due to Ehrig and Kreowski: derivations which differ just in the execution order of concurrent rewritings are identified, and in this way the amount of parallelism in the derivations is recognized.

Abstract concurrent semantics are also defined. Three classical observations for concurrent semantics — interleaving, partial ordering and mixed ordering — are defined on the graph derivations. Bisimulation is exploited to obtain three equivalences of π-calculus agents corresponding to the observations. As expected, the ordinary observational equivalence for the π-calculus is re-obtained in this context as the equivalence induced by the interleaving observation.

Genuine Pullback Rewriting

Michel Bauderon

LaBRI, Université de Bordeaux, France

Graphs can roughly be considered from two points of view, either as sets of vertices linked by edges or as sets of edges glued by vertices, each point of view leading to a different kind of graph rewriting systems, basically *node rewriting* and *edge rewriting*.

In both cases the basic ingredients are given by specifying what is to be replaced, how it is linked to the rest of the graph, by what it will be replaced and how the replacing part will be connected to the remaining part of the original graph, the main difference between both types of rewriting being probably that node rewriting may create new edges in an unpredictable way, while edge rewriting does not create anything, but simply unites into a single object already existing items.

This has made a big difference when trying to develop a more abstract setting for graph rewriting. Indeed, considering the graph to be rewritten as "embedded" in the big graph (in a sense which we shall not make more precise), it quickly appeared that the categorical generalizations of union and equivalence relation, namely coproduct and coequalizer where enough to give a good description of edge-oriented rewriting. This gave rise to the well known double-pushout approach extensively developed by the Berlin school. Unfortunately, this approach was absolutely unable to describe the creation of edges and thus was not applicable to node rewriting.

	Hyperedge Replacement	Vertex replacement
Substitution	replaces an edge	replaces a node
Interface	a family of nodes	a set of edges
Connection	glues interface nodes	creates interface edges
Algebraic framework	Available	Available
Categorical approach	Available	*None*
Logical Theory	Available	Available

This talk presents the main principles and results obtained over the last two years using pullback rather than pushout as a basic rewriting mechanism. The basic objects are *structured graphs* i.e. graph morphisms of the form $G :\!\!\rightarrow S$ where S is a fixed graph. They form a (comma) category with pullbacks and S is a neutral element for the categorical product.

A fixed structured graph of a very specific shape $\mathcal{A}$, called the alphabet is defined such that unknowns and rules are morphisms of structured graphs with codomain $\mathcal{A}$, whose pullback provides the basic rewriting mechanism.
We then show that with an appropriate choice of S, single pullback and double pullback rewriting can describe both vertex replacement (of the NLC and NCE type) in graphs and hypergraphs, hyperedge replacement and most cases of double pushout rewriting.

References :

M. Bauderon, A uniform approach to graph rewriting : the pullback approach, in Proceedings WG'95, *Lect. Notes in Comp. Sci 1017.*, 101-115

M. Bauderon, H. Jacquet, Node rewriting in graphs and hypergraphs : A categorical approach, *submitted*, short version to appear in Proceedings WG'96, *Lect. Notes in Comp. Sci*

Double Pullback and Pattern Rewriting in Hypergraphs.

Hélène Jacquet

LaBRI, Université de Bordeaux, France

This talk was a follow-up on a framework introduced by Michel Bauderon in 1994. Michel Bauderon has shown how the pullback (and double pullback) can be used to provide a unifying description of various graph rewriting mechanisms such that NLC or NCE.

This work extends the use of the pullback in order to rewrite a node, a handle or more generally a pattern in a hypergraph.

A double-pullback rule is a couple of morphisms (l, r) - in the category we have defined for hypergraphs - from two hypergraphs L and R to a third structured hypergraph $\mathcal{A}$ called the alphabet.

An occurrence of this rule in a given hypergraph G is a morphism $occ : G \to L$ such that the pattern of G which will be rewritten exactly occurs in L.

The application of the rule to this occurrence is done in two steps :

- We first built the pullback complement D. This pullback complement is a hypergraph such that there exists a morphism $a : D \rightarrow \mathcal{A}$ and G is the result of the pullback of the pair (a, l).
- Second, we build $\overline{G}$ as the pullback of the pair (a, r) : $\overline{G}$ is the result of the rewriting of G by the double pullback rule (l, r).

Finally, we have presented *HH-grammars* introduced by Courcelle, Engelfriet and Rozenberg in "Handle Rewriting Hypergraph grammars" [1993], and we have shown how we can construct for each rule of an *identification-free HH-grammar* a corresponding double pullback rule.

Persistent Sequential Entities in ESM Systems

Dirk Janssens

University of Antwerp (U.I.A.), Belgium

In the analysis of systems consisting of a set of interacting sequential agents, such as concurrent object-oriented systems, actor systems or protocols, it is desirable to have the possibility of proving properties of the global system by combining proofs of properties of the separate sequential components. In this contribution it is shown that the process semantics for ESM graph rewriting (a generalization of Actor Grammars) allows such a compositional analysis.

The basic idea is that the semantics consists of processes (in the sense of Petri Net processes, i.e., based on an explicit representation of causality) equipped with a condition on the contexts in which they may occur. These conditions, which are closely related to the embedding mechanism of ESM systems, provide a means to relate information from the semantics of one sequential agent to information from the semantics of another sequential agent, thus providing an interface between the parts of a compositional proof.

The method is demonstrated by a proof of a global property of a simplified sender-receiver protocol. The results presented illustrate the potential of a process-based semantics for graph rewriting as well as the expressive power of a well-chosen embedding mechanism.

A Partial Order Representation of Processes of Transforming Graphs

Józef Winkowski

Polish Academy of Sciences, Poland
joint work with Andrea Maggiolo-Schettini, University of Pisa, Italy

Usually processes of rewriting graphs are represented by sequences of applications of productions, called derivations, and it is assumed that applications which are independent in the sense that they do not intersect except in the contexts can be performed concurrently. The potential concurrency of applications of productions in a process of rewriting graphs can be reflected in the so called shift-equivalence of representing derivations, that is in the possibility of obtaining such derivations one from another by repeatedly exchanging contiguous independent steps. Consequently, a process of rewriting graphs can be defined as an equivalence class of derivations with respect to shift-equivalence, called a derivation trace.

In a paper by Corradini, Ehrig, Löwe, Montanari and Rossi it is shown that derivation traces of so called safe graph grammars can be ordered such that they form prime event structures. In a paper by Corradini, Montanari and Rossi it is shown that derivation traces of safe graph grammars correspond exactly to isomorphism classes of partially ordered structures called graph processes.

Both derivation traces and graph processes are defined for concrete graph grammars with the idea of associating with each grammar a semantical meaning in the form of the set of its derivation traces or graph processes.

In our presentation we introduce models of processes of rewriting graphs independently of graph grammars, and we define operations allowing us to combine such models. In particular, this allows us to describe sets of processes of concrete graph grammars.

Similarly to graph processes, our models, called dynamic graphs, are enriched variants of contextual occurrence nets in the sense of Montanari and Rossi. A dynamic graph consists of data elements which represent nodes and edges of graphs taking part in a process of rewriting or in a collection of such processes, and of events which represent rewriting steps. When representing a single process a dynamic graph may be equipped with some canonical representations of the initial graph and of the resulting graph, and then, by analogy with concatenable processes of Degano, Meseguer and Montanari, called a concatenable dynamic graph.

Dynamic graphs may be considered to be either concrete or abstract, that is up to isomorphism. An important feature which distinguishes dynamic graphs from derivation traces and graph processes is that abstract concatenable dynamic graphs admit natural universal operations with the aid of which one can represent derivation traces of arbitrary graph grammars.

Another Approach to Model Actor Systems with Graph Transformations

Ingrid Fischer

Universität Erlangen, Germany

Actors are independent concurrent objects that interact by sending asynchronous messages. Each actor has a unique mail address which may be used to send it messages. Moreover, a mail address may be included in messages sent to other actors so that the actor topology may be changed. Each actor has a behaviour which determines how the actor responds to a message. This response can consist of different actions: the actor can send new messages, create new actors and change its behaviour.

Visual specification of both the structure and the dynamic process flow in this environment can help either to write actor programs in a corresponding language or to debug already written programs at the same level the specification is given.

Graph transformations are an appropriate formalism to handle this task. In a configuration graph, actors as well as messages are given as nodes marked with a behaviour/message label and an element of a Σ-algebra for their contents. Acquaintances of actors/messages and the target of messages is modelled with the help of edges. Transformation rules serve to change the configuration graph. Each rule simulates the acceptance of a message by an actor and inserts new messages/actors in the graph together with the new behaviours and acquaintances.

The application of graph transformation rules is modelled with the classical double pushout approach. Several possible transitions in an actor system can be described with the help of parallel application of rules.

Graph Rewriting for Coordination

Ugo Montanari

Università di Pisa, Italy
joint work with Francesca Rossi

In the talk we describe our approach to modelling the dynamics of coordination systems. For distributed systems we mean systems consisting of concurrent processes communicating via shared ports and posing certain synchronization requirements, via the ports, to the adjacent processes. We use graphs to represent states of such systems, and graph rewriting to represent their evolution. The kind of graph rewriting we use is based on simple context-free productions which are however combined by means of the synchronization mechanism. This allows for a good level of expressivity in the system without sacrifying full distribution. Moreover, to approach the problem of combining productions together, we suggest to exploit existing techniques for constraint solving. This is based on the observation that the combination problem can be modelled as a (finite domain) constraint problem. In this respect, we propose to use both local consistency techniques, to remove the possible redundancies in a system state, and a distributed backtracking search algorithm, as used in distributed constraint solving. Our method has two main advantages: first, it is completely formal and thus provides a precise description of the way a distributed system evolves; second, it also seems very promising from the performance point of view, since the techniques we propose to combine productions together have been proven very convenient in several cases.

A (2-)Categorical Presentation of Term Graph (Rewriting)

Fabio Gadducci

Università di Pisa, Dipartimento di Informatica, Italy

The classical theory of *Term Graph Rewriting* studies the issue of representing finite terms with directed, acyclic graphs, and of modeling term rewriting via

graph rewriting. The main advantage of using graphs is that the sharing of common subterms can be represented explicitly in a graph. Therefore the rewriting process is speeded up, because the rewriting steps do not have to be repeated for each copy of an identical subterm. Thus a single graph reduction may correspond to n term reductions, where n is the "degree of sharing" of the reduced subterm.

The rich theory of Term Rewriting also includes a nice categorical presentation of term rewriting systems in terms of 2-categories. Terms over a given signature Σ can be represented faithfully as arrows of a suitable cartesian category, called the *Lawvere theory* of Σ. This category has natural numbers as objects, and an arrow from n to m is an m-tuple of terms with at most n variables; arrow composition is substitution, and the product of two arrows is the concatenation of the corresponding tuples of terms.

Given such a category, a rewrite rule $R = \langle l, r \rangle$ can be regarded as a *cell*, i.e., a vertical arrow between the two arrows corresponding to the left-hand side l and to the right-hand side r. This situation can be denoted as $R : l \Rightarrow r : n \to 1$, which also says that l and r are arrows from n to 1, i.e., they have at most n variables. Given a term rewriting system, the structure obtained in this way, i.e., the Lawvere theory of Σ enriched with one cell for each rule, is called a *c-computed*. The interesting fact is that from such a c-computed, a free construction can generate a (cartesian) 2-category by adding all identity cells, and closing cells with respect to horizontal and vertical composition. The resulting 2-category faithfully represent all the possible rewriting sequences of the original system. In fact, horizontal composition of cells generates all the possible instantiations of the rules and, at the same time, places rules in all possible contexts. Vertical composition acts instead as sequential composition. Furthermore, the generated rewriting sequences are subject to an equivalence that subsumes the Lévy equivalence, due to the axioms of 2-categories.

Now, is it possible to give a 2-categorical presentation of term *graph* rewriting analogous to the one just described? In this talk some initial results are illustrated. We show that, while terms can be seen as arrows of a cartesian category (the Lawvere theory), so *term graphs* are the arrows of a *s-monoidal category*, i.e., a symmetric strict monoidal category equipped with two "symmetric transformations" ∇ (the *duplicator*) and ! (the *discharger*). Our current work tries to prove that, representing term graph rules as cells, the free 2-category of the resulting computed faithfully represents term graph rewriting sequences.

In the talk we introduce *ranked term graphs*, i.e., equivalence classes of directed acyclic graphs labeled over a signature Σ, with distinguished lists of *variable* and *root* nodes, which are used for their "composition", which is the counterpart of term substitution. One key result shows that every ranked term graph can be obtained form a small set of atomic term graphs using the operations of composition and "union", i.e., disjoint union with concatenation of the lists of

roots and variables.

The 2-categorical presentation of term graph rewriting allows us to clarify in a formal framework both the similarities and the intrinsic difference between term rewriting and term graph rewriting, disregarding any representation mismatch. This topic has been addressed in many paper in literature, where all the authors agree that term graph rewriting is a sort of term rewriting with explicit sharing of subterms. However, this simple fact is often made more complex by the need of encoding and decoding functions between terms and term graphs.

On the contrary, the categorical framework allows to show that the only difference between terms and term graphs (regarded as arrows of suitable categories) is in the *naturality* of the transformations ∇ and !: in fact a s-monoidal category where these transformations are *natural* is also cartesian.

Rule-Based Refinement of High-Level Structures

Julia Padberg

Technical University of Berlin, Germany[1]

The concept of refinement is an important technique within software engineering. Based on the idea of high-level replacement systems in the double-pushout approach, we propose rule-based refinement to present rules denoting the replacement of a substructure by another one, without changing the remaining part. This has the advantage of a simple local presentation of the refinement, even if the whole system is large and complex.

We discuss the relation of rule-based refinement to other notions of refinement. Assume that we have an arbitrary refinement relating some structure to some other structure. Then this relation of the two structures can be regarded as the left and the right hand side of a transformation. Hence, each refinement discussed in literature, can be considered as a transformation in our sense. A more substantiated approach to include well-known refinements is the new concept of $\mathcal{Q}$-transformations within the frame of high-level replacement systems. This concept combines well-known approaches of refinement with the new concept of rule-based refinement. The main idea is to supply rules with an additional morphism, which belongs to a specific class $\mathcal{Q}$ of morphisms.

[1]This work is part of the research project *Forschergruppe Petrinetz Technologie* and has been partially supported by the German Research Council (DFG).

Moreover, we review the concepts and results for independence and parallelism of transformations in high-level replacement systems and extend them to $\mathcal{Q}$-transformations. We introduce horizontal structuring techniques, union and fusion, that generalize constructions known from coloured nets to high-level structures using the categorical concepts of coequalizers and pushouts within the frame of high-level replacement systems. Furthermore, we sketch the compatibility of these constructions with both transformations and $\mathcal{Q}$-transformations.

Finally, we demonstrate the application of these techniques to our case study, concerning the requirements engineering of a medical information system.

Rewriting Fuzzy Graphs

Yasuo Kawahara

Kyushu University, Japan
joint work with Masao Mori

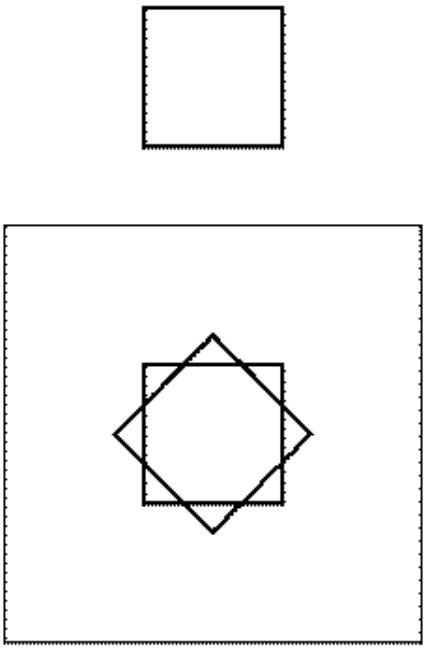
The aim of this talk is to formalize a fuzzy graph rewriting with single-pushout approach from a viewpoint of relational calculus. A fuzzy graph here means a pair of an (ordinary or crisp) set of nodes and a fuzzy (connection) relation on the nodes. To formalize a fuzzy graph rewriting in this setting we first discussed about the algebraic and logical structure of fuzzy relations. A partial morphism between fuzzy graphs is a partial function between the sets of nodes preserving fuzzy graph structures in a sense. As an application of relational calculus we showed that the category of fuzzy graphs and their partial morphisms has pushouts. Based on these facts we formalized a fuzzy graph rewriting with single-pushout approach. In general the definition (choice) of matchings to production rules is very important, and changes the aspect of graph rewritings. Thus we proposed two kinds of possible matchings for fuzzy graph rewriting. The former is a rigorous matching, which leads a fact that if the production rule is a partial morphism, then the rewriting square is a pushout. The latter is rather ambiguous one, called an ε-matching, where $0 < \varepsilon < 1$. The fuzzy graph rewriting with using ε-matching gives an ε-equivalent fuzzy graph regarded as an approximation to pushout rewritings.

Assisting Creativity by Graph Transformations

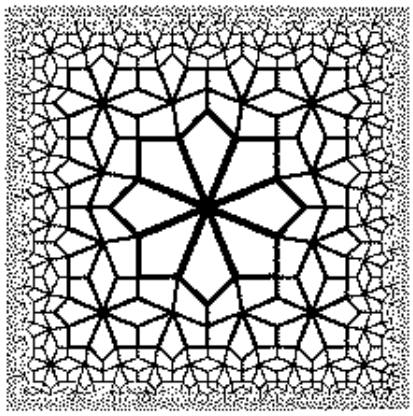
Ewa Grabska

Jagiellonian University, Cracow

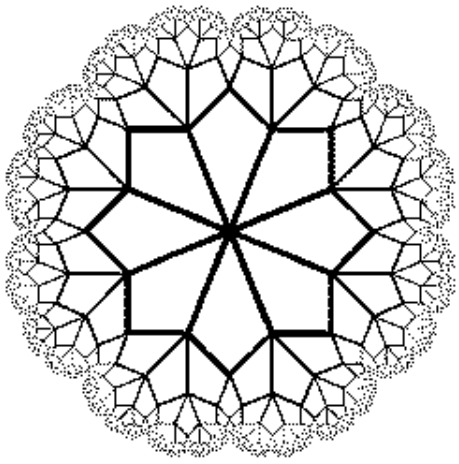
In this paper creative design is discussed within the framework of graph transformations. On the one hand, the increasing availability and use of the Worldwide Network has opened up the opportunity for designers to collaborate with each other in ways not possible previously. Designers can now work on the same design at different geographic locations. This new possibility requires a fresh look at the role of formal approaches to design. On the other hand, it is known that design is associated at many levels with patterning, shaping and form giving. Therefore representations of visual information have always played a very significant role in design problem solving. This is another reason why our considerations are restricted to graphical design in which a drawing is a basic unit of communication among designers as the transer of a message from one designer to another.



The term design may be discussed in relation to both product (designed object) and process. Our approach to design is based on the composite representation which allows to integrate the product designing and the process designing. The composite representation forces the designer to think about products at two levels: the higher level of structural properties described by means of the graph and the lower level of geometry and other attributes, like colour, texture, described by the realization scheme. This representation is to a great extent oriented towards visual evaluation.



Certain problems in visual communication are centered typically around the questions: what is intended and what is perceived? Concerning perception and creative thinking leads to the concept of emergence being in the heart of large scientific interest. On the level on graph structures, emergence can be defined as arising implicit graph structures from a given explicit graph structure during the evolution of problem requirements and solution concepts. The importance of emergence in design is that it allows to extend the space of possibilities to a solution. The composite representation allows easily to fix these new possibilities. Searching of emergent shapes can be seen in the relation to an attempt at achieving order. Some properties of the regular emergent graph structures are presented. By visualizing the graph structure of the designed object the designer has a chance to grasp the essence of the current solution.

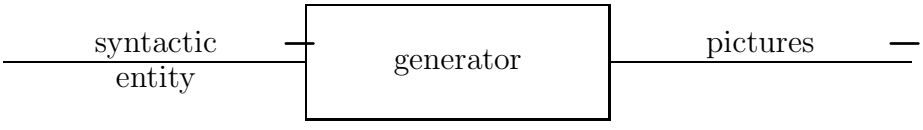


Syntactic Picture Generation: Methods and Tools

Hans-Jörg Kreowski

Universität Bremen, Germany

Methods for picture generation may be attributed as *syntactic* if they provide means for generating pictures from a given syntactic entity like a grammar or an automaton.



Examples of this kind are chaincode picture languages, L-systems with turtle interpretation, map-L-systems, cellular automata, iterated function systems, and collage grammars. In all these models of syntactic picture generation, derivations of certain configurations are built up where each occurring configuration can be

interpreted as a picture (or a scene if one works in higher dimensions). Hence one always gets sequences of pictures in a natural way, i.e. animation sequences or films.

Such syntactic films can be created in the studio system *BIZARR goes to Hollywood* which has been developed under UNIX/X-Windows in C++ at the University of Bremen. It provides all the methods mentioned above (except chaincode interpretation). The methods can be freely combined to produce sequences of 3D-scenes. One can observe the evolving scene and see the film from the point of view of a camera. There is a fast wireframe output that runs nearly interactively while the film is created. And there is a raytracer output that runs days and weeks for some seconds of film, but allows scenes with colors and textures.

Some Ideas Concerning Graph Transformation by Tree Transductions

Frank Drewes

Universität Bremen, Germany

A tree transduction is a nondeterministic transformation $\tau: T_\Sigma \to \wp(T_{\Sigma'})$ between sets of terms over finite signatures. If the symbols in Σ are graph operations and those in Σ' are operations on some domain D such tree transductions can compute mappings f of graphs into D. For this, it is required that, if $t \in T_\Sigma$ denotes a graph G then $\tau(t)$ consists of one or more trees that denote $f(G)$.

This notion of computation by tree transductions is known to yield some interesting decidability results. For example, if Σ' consists of the operations 0, 1, maximum, addition, and multiplication on the natural numbers and τ is a composition of top-down or bottom-up tree transductions then it can be decided for context-free graph languages L (represented by a corresponding grammar) whether f is bounded on L.

While this is quite a nice result it is often the case that one would like to verify properties of graph transformation systems and their derivations rather than of graph languages. This talk presents first ideas in this direction. The basic idea is to consider a tree transduction $\tau: T_\Sigma \to \wp(T_\Sigma)$ (where Σ is a signature of graph operations) that computes the single steps of a derivation relation $\Rightarrow$. Since the output signature equals the input signature iteration is possible. Thus, repeated

application $t_1 \to_\tau t_2 \to_\tau \cdots \to_\tau t_n$ of τ yields a derivation $G_1 \Rightarrow G_2 \Rightarrow \cdots \Rightarrow G_n$, where each G_i is the graph denoted by t_i.

In the talk, this notion of derivation is introduced. As an example it is shown how fully balanced trees can be recognized by an algorithm which is expressed in terms of a derivation relation computed by a linear bottom-up tree transduction. Furthermore, some very first results are presented. The first shows that, in a certain way, some types of derivation relations defined in the double-pushout approach can be computed by tree transductions. The second type of results concerns methods which allow to prove properties of the derivation relations computed by linear bottom-up tree transductions. Typical proofs of this kind are termination proofs and proofs of properties which remain invariant under the considered derivation relation.

Parameterized Graph Transformation Units

Sabine Kuske

Universität Bremen, Germany

Graph transformation units are a structuring principle of the graph-and-rule-centered language GRACE, currently under development involving researchers from Aachen, Berlin, Bremen, and Leiden. The components of a transformation unit are a set of rules, descriptions of initial and terminal graphs, a control condition, and a set of imported transformation units. Semantically, it transforms initial graphs to terminal ones by interleaving rule applications with calls to imported units such that the control condition is obeyed.

In the talk, an axiomatic approach to parameterized transformation units is presented which allows to describe classes of transformation units. For this purpose, the components of a parameterized transformation unit are typed expressions containing typed variables. Each instantiation of the variables with expressions induces a substitution of the parameterized transformation unit. The presented concept has the following useful properties: First, substitution of a parameterized transformation unit yields again a parameterized unit. In particular, if the actual parameters do not contain variables, we obtain a non-parameterized transformation unit. Second, the substitution is associative. This means roughly speaking that we get the same unit if we first substitute a parameterized transformation unit *trut* w.r.t. a variable instantiation *val* and then the result w.r.t.

val', or if we first substitute the expressions associated to the variables by *val* w.r.t. *val'* and then the unit *trut* w.r.t. the resulting variable instantiation.

Modelling Object-Oriented Databases by Attributed Two-Level Graph Grammars

Herbert Göttler

Universität Mainz, Germany
joint work with Bernd Himmelreich, GEFM Eschborn, Germany

Graphs are used at many stages of database design and implementation. This is the case for the relational and for the object-oriented paradigm as well. In the project PARES ('Picture Administration and Retrieval System', supported by the Ministry of Economics of the State of Rheinland-Pfalz) we investigate the usefulness of object-oriented databases for non-standard applications, especially for the storing of (Picasso) pictures as scans and additional information pertaining to them. Such information can be: A syntactical description (the set of graphical objects like lines, rectangles, circles, any kind of polygons, etc.), a semantical description (the set of objects a viewer is supposed to see, like a glass, a person, a table, etc.), and the relations between them (e.g.: This rectangle, representing a table, is parallel to one representing a mirror.). A query like: 'Show all tables in the database!' (not just the pictures containing a table!) requires an extremely complicated datamodel which can only be designed in a reasonable manner by the object-oriented approach.

Things become even more complicated if also dynamic aspects have to be considered: In the PARES-project we do not only deal with unrelated pictures but also with series of pictures which are related in the sense that one can define 'the difference' between one picture and its (logical) successor. This is very much alike the step-by-step construction of, say, a machine. Such a difference - in other words: such a construction step - could be: A rectangle representing a table is rotated by 30 degrees.

For our application it is necessary not to store just the sequence of pictures (this was done for the sake of animation, too) but also the operation which produces the successor picture of a given one.

In our framework we model the database as a graph representing the objects and the relations (inheritance included). Quantities like coordinates are handled

by attributes. An operation like 'Rotate a rectangle representing a table from a position parallel to the axes by 30 degrees!' gets modelled as the application of a graph production. Depending on the difference between two succeeding pictures the object scenario plus its attributes can change dramatically with the effect that the graph production implementing the change of the graph of the object scenario can be very complicated. This is due to the fact that many different aspects are incorporated into a single production, like consistency of the relations in respect to inheritance.

According to the principle 'Separation of concerns' we use two-level graph grammars to cope with the complexity of the problem. On the one level, the hyperrule level, we describe, so to speak, a simple specification of what has to be done. On the other level, the metarule level, we describe how it is done. The metarules guarantee, for example, the already mentioned consistency aspects.

The whole setting works only because the graph productions - the hyper- and the metarules as well - are considered to be graphs themselves. This way, the metarules can be applied to the hyperrules (until these become 'productive') in the very same way as the productive hyperrules can be applied to the graph representing the present state of the database.

Object-Based Specification of Communication-Based Systems Views and Synchronization

Annika Wagner

TU Berlin, Germany

We present an object-based specification technique which includes an advanced attribute concept, views, a synchronization mechanism and a mechanism for ensuring consistency. In this talk we concentrate on views and synchronization. Formally our approach is based on the single-pushout transformation of partial algebras, which are used to represent global system states.

Introducing a view concept as generalization of the knows-relationship between objects to a knows-relationship between an object and the global state, leads to a more complex description of a system state. This description by a so-called system graph consists of partial algebras and partial injective morphisms in

between them instead of just a single partial algebra. We introduce single-pushout transformation of these system graphs with a complex notion of a system transformation rule consisting of a (global) state transformation rule and (local) view transformation rules.

Integrating the concept of amalgamation of rules as synchronization mechanism we can specify complex actions in a modular way by a rule scheme containing different types of rules instead of a single system transformation rule. Valuation rules are used to specify the local effects an action has on the attributes of a single object. Interaction rules describe the synchronization of actions of different objects.

An object-based specification consists of a static object specification, namely an object signature and an algebraic specification of the used abstract data types, and a specification of the objects dynamics, namely a set of valuation and a set of interaction rules assigned with each object type, which are combined in rule schemes.

Furthermore we show how our approach is related to that of distributed graph transformation (presented by G. Taentzer) and to that of the synchronization of views (presented by R. Heckel).

Substitution-Based Graph Rewriting

Annegret Habel

Universität Hildesheim, Germany

The talk addresses the substitution-based approach to graph rewriting, introduced in Plump and Habel (1996). The approach is very simple to describe and needs neither pushouts nor embedding instructions. It allows to simulate the well-known double pushout (DPO) approach as well as term rewriting in such a way that a direct derivation in these models corresponds to a direct derivation in the substitution-based approach. Restricting the approach to *basic* substitution-based (SB) rewriting, one obtains a one-to-one correspondence between DPO rewriting and basic SB rewriting. This allows to transfer results like commutativity and parallelism theorems from the DPO approach to the basic SB approach. For general SB rewriting, a "weak commutativity problem" is investigated. Counterexamples to weak commutativity in the general case are given. For the case of weakly non-overlapping derivations $G \Rightarrow H$ by r and $G \Rightarrow H'$ by r', where r

deletes, preserves, or copies context and where r' preserves context, the existence of derivation sequences $H \Rightarrow X$ by r' and $H' \Rightarrow X$ by r is shown.

Constrained data types

Pieter Koopman

Leiden University, The Netherlands

This paper treats enforcing constraints on data types in functional programming languages. Examples of the intended kind of constrained data types are sorted lists and balanced trees. Since the standard type system is not able to check these constraints, the usual solution is to use an abstract data type for the constrained version of the data type. Since instances of abstract data types can only be constructed by the fixed number of functions in the signature, it is easy to verify that the constraints will be met.

The drawback of this approach is that the constrained data type is entirely different from the plain data type. This implies that none of the standard manipulating functions can be applied to the constrained data type. For instance, the standard function to compute the length of a list cannot be applied to the abstract data type sorted list. This can only be solved by defining new instances of the standard functions for the constrained data types, or by using explicit type conversions. Both solutions are undesirable.

We propose to use type annotations to indicate that the data type is constrained. By a proper extension of the notion of type instance the standard functions are applicable to the constrained data types. By indicating how the annotated types behave under construction and destruction, it is possible to derive an approximation of the possible annotations automatically. The functions can become polymorphic in the annotations on the types manipulated.

Like abstract data types the notion of subject reductions is lost. However, in annotated types the change is less dramatic. For abstract data types, the derived type after reductions can be totally different from the original type. For annotated types the original type is an instance of the derived type: it is not always possible to reconstruct the annotations. Apart from the annotations subject reduction holds.

Automorphism groups of rooted deterministic equational graphs

Laurent Pélecq

LaBRI, Université Bordeaux I, France

Equational graphs have been introduced by Bauderon and Courcelle. An equational graph is the least solution of sets of hypergraphs equations. Courcelle showed that monadic second order logic (MSOL) of equational graphs is decidable (cf. [Courcelle 89]). Moreover he showed that equational graphs are definable by a closed formula of MSOL (cf. [Courcelle 90]). Some other results can be found in [Caucal 90].

Thanks to these results, one can show that the orbit of any vertex of an equational graph is definable in MSOL. We consider a rooted deterministic equational graph G. Let x denotes a root of G. Using pushdown automata (pda), one can construct a context-free graph H that gives G by contraction of some edges (cf. [Muller, Schupp 85]). By successive transformations of the formula defining the orbit of x, we define a set of configurations of the pda which represents the orbit. This is an example of the Courcelle's general formalism of relative definability of structures. The set of configurations is proved to be rational.

From rationality, we deduce a decomposition of any automorphism over a set of generators. These generators are the automorphisms for which the image of x is the extremity of some positive path of bounded length with origin x. The bound depends only on the graph and the root. This set of generators is finite in deterministic graphs.

The result is that automorphism groups of deterministic rooted equational graphs are finitely generated. It generalizes the same result for context-free graphs (cf. [Pélecq, to appear]).

(Graph Transformation Systems) Transformation Systems

Francesco Parisi Presicce

Universitá di Roma La Sapienza, Italy

Graph Transformation Systems provide a powerful and flexible formalism for the specification of parallel and distribute systems. Configurations are represented by graphs and their evolution by the application of transformation rules.

In the process of developing an adequate transformation system, it is often necessary to modify the current GTS. This can be done either by replacing a subsystem with another one (global transformation) or by modifying in a uniform way some of the existing rules (local transformation).
The global transformation can be achieved via a High Level Replacement System based on the category of GTS and GTS morphisms. Each rule has the form $L \leftarrow K \rightarrow R$ (in the Double Pushout approach to graph rewriting) or $L \leftharpoonup R$ (in the Single Pushout approach) where L, K and R are GTS and the morphisms are total in the DPO and partial in the SPO. In the DPO approach we obtain an $HLR1$-category while in the SPO approach we have at least an $HLR0.5$- category . In both cases then the GTS-replacement systems satisfy the Local Confluence Theorem.
The local transformation is obtained by applying a transforming rule p to each production p_1 of the GTS. The expected result is that if $p : p_1 \Rightarrow p_2$, $p_1 : G_1 \Rightarrow H_1$ and $p : G_1 \Rightarrow G_2$, then $p : H_1 \Rightarrow H_2$ and $p_2 : G_2 \Rightarrow H_2$ both in the DPO and in the SPO.
While every local transformation can be viewed as a global one (replacing the old rules with the modified ones), the converse is not always true.
The results are based on compatibility of application morphisms and composition of diagrams and so the ideas and techniques can be applied to other approaches based on universal categorical constructions.

Application of Graph Grammars in an Educational Software Engineering Game

Kurt Schneider

Daimler Benz AG

SESAM is a software project simulator. It was developed at the University of Stuttgart, Germany. SESAM enables students of software engineering to lead their own (simulated) projects. Each student plays the role of a project manager. All other project participants (including customer, software developers, etc.), all documents (specifications, designs, code, test plans etc.) are simulated. Players interact with their simulated projects in an adventure game fashion: They assign tasks to developers, organize, plan, and schedule project activities, interact with developers, and they should control project progress. They also have to react upon unexpected events.

Internally, SESAM uses a situation model that can be characterized as an instantiation of an extended Entity-Relationship schema (including inheritance and attributes). The ER schema represents the universe of discourse; each concrete instantiation represents a concrete situation in a particular project. Such a situation model can be formally interpreted as graph. Nodes are instantiations of classes such as "Developer", "Specification Document", or "Customer". Edges represent relationships between nodes. Each instantiation has a set of attributes with individual values.

Quantitative changes in this model result from attribute value changes over simulated time. Qualitative changes stem from graph modifications performed on the situation model. Graph modifications are formalized as productions of an attributed graph grammar. Player actions may directly trigger a graph grammar rule. Depending on subgraph patterns detected in the situation model graph, other productions may be applied without direct user involvement. They produce autonomous and often unexpected project behaviour. Whenever a pattern ("left hand side of a graph production") is detected in the situation model graph, the production can be applied. This leads to subgraph replacement ("replacement of left hand side by right hand side"). Each production is called "event model" in the SESAM terminology. A new notation was developed for event models. It combines left and right hand sides of a graph production, annotating elements that are to be deleted by minus signs, and elements that are to be inserted by plus signs. We did not use existing formalisms or tools because we had to tightly integrate qualitative graph modifications with quantitative continuous attribute value changes over simulation time.

An algebraic graph grammar approach was used. The application of the graph grammar formalism allowed to directly describe a triggering situation pattern together with a resulting action in terms of situation model graph modifications. Many application domains for graph grammars reported in literature are themselves highly formalized calculi. In contrast, SESAM started with a rather vague specification of requirements. Graph grammars where pragmatically selected as modelling medium because they seemed to minimize the cognitive distance between observable events in real software projects and their description in event models. In both cases, patterns trigger events.

Not the formal properties of a calculus stimulated this application, but its intuitive appeal. From this perspective, a more pragmatic and comprehensive introduction to graph grammars would be highly desirable. Such a writeup could lead application providers to make a better informed choice among existing graph grammar approaches. And it should provide easy-to-read examples of how to get started in the respective approach. Excessive formality must be avoided in such a tutorial.

On Termination of Algebraic Graph Rewriting Systems

Jürgen Müller

Technische Universität Berlin, Germany
joint work with Detlef Plump, Universität Bremen, Germany

Two necessary and sufficient conditions for termination of algebraic graph rewriting systems are established. In the frameworks of the two algebraic graph rewriting approaches, single-pushout and double-pushout, it is shown first, that termination is equivalent to the finiteness of all *forward closures*, being certain minimal derivations in which each step depends on previous steps. This characterization is the same for both algebraic graph rewriting approaches. It is not possible to receive this result for the single-pushout approach from the double-pushout approach only by translating the rules. The translation of the rules does not reflect the application condition of rules in the double-pushout approach. Two necessary conditions for the proof of the first termination criteria are shown for both algebraic graph rewriting approaches, the compatibility of dependence and minimality of derivations, and the compatibility of independence and minimality of derivations.

It is shown second, that termination of a graph rewriting system is equivalent to the existence of a *reduction order* for this system. Reduction orders are well-founded, partial orders on graphs, including every possible derivation of a graph rewriting system.

Termination of Term Graph Rewriting

Detlef Plump

Universität Bremen, Germany

Term graph rewriting differs from term rewriting in that common subexpressions can be shared, due to the representation of expressions by graphs. Sharing saves not only space but also time since certain repeated computations can be avoided. This talk addresses the termination of term graph rewriting. First, it is shown that term graph rewriting terminates for more systems than term rewriting. However, when all rules are right-linear (i.e., no rule contains repeated variables in its right-hand side), then term graph rewriting terminates if and only if term rewriting terminates. This allows to transfer undecidability results for some restricted classes of term rewriting systems to term graph rewriting. It is also shown that termination of term rewriting is undecidable for those systems that are terminating under graph rewriting.

In the second part of the talk, *fully shared rewriting* is introduced as a particular kind of term graph rewriting in which term graphs are fully collapsed before a rewrite rule is applied. An advantage of this rewrite model is that terms have unique graph representations, so that termination may be proved by orders on terms. To this end, an extension of the well-known recursive path order is proposed. This *full sharing order* is, unlike the recursive path order, not a simplification order and hence is not a priori known to be well-founded. To overcome this problem, a notion of homeomorphic embedding is considered that is strictly included in the conventional notion. It is conjectured that this relation is still a well-quasi order on the set of ground terms (yielding a generalization of Kruskal's Tree Theorem). As the full sharing order contains the new embedding relation, the truth of the conjecture would imply that the full sharing order is well-founded.