Contents

Preface vii

Timed Process Algebra: Theory and Applications
J.C.M. Baeten 1

A Unified Relational Approach to Semantics of Non-deterministic Applicative Programs
Rudolf Berghammer 1

Architectural Specifications
Michel Bidoit 2

Evolving Algebras and Parnas Tables
Egon Börger 3

Proof systems for parametric specifications
María Victoria Cengarle 4

Specification of Concurrent Systems: From Petri Nets to Graph Grammars
Andrea Corradini 5

First-order Definable Automata
Jorge R. Cuéllar 5

Specifying Concurrent Information Systems Without Specifying Concurrency
Hans-Dieter Ehrich 6

Synchronization of views of system specifications based on graph transformations
H. Ehrig 6

Specification and Semantics of Software Architectures
José Luiz Fiadeiro 7

An Object Specification Language Implementation with Web User Interface based on Tycoon
Martin Gogolla, Mark Richters 8

Classical Logic and Exception Handling
Philippe de Groote 11

First Steps Towards an Institution of Algebra Replacement Systems
Martin Große–Rhode 13

On Proof Systems for Structured Specifications
Rolf Hennicker 13

Behavioural abstraction and behavioural satisfaction in higher-order logic, with higher-typed functions
Martin Hofmann and Don Sannella 14

Action Refinement and Inheritance of Properties in Systems of Sequential Agents
Michaela Huhn 15

Describing the Semantics of Concurrent Object-Oriented Languages
C B Jones 16

Basic concepts of rule-based specification
Hans-Jörg Kreowski 16

Three Techniques Used in Specification Development
Wei Li 17

Rewriting Logic as a Logical and Semantic Framework
Narciso Martí-Oliet (joint work with J. Meseguer) 18

Primitive subtyping $\wedge$ implicit polymorphism $\models$ object-orientation
Stephan Merz 19

Membership Algebra
José Meseguer 19

Refining Ideal Behaviours
Bernhard Möller 21

The Tile Model
Ugo Montanari 22

CoFI: The Common Framework Initiative for Algebraic Specification
Peter D. Mosses 23

A monotonic declarative semantics for normal logic programs
Fernando Orejas 24

Proof of Refinements Revisited
Peter Padawitz 24

An Algebraic Approach to Global-Search Algorithms
Peter Pepper 25

A variant of Quantified Dynamic Logic
Gerard R. Renardel de Lavalette 25

Mind the gap! Abstract versus concrete models of specifications
Don Sannella 27

Functional Specification Using HOPS
Gunther Schmidt 27

Behavioural satisfaction and equivalence in concrete model categories
Andrzej Tarlecki 28

Faster Asynchronous Systems
Walter Vogler 29

A Formal Approach to Object-Oriented Software Engineering
Martin Wirsing 29

Inductive Theorem Proving in
Partial Positive/Negative-Conditional Equational Specifications
Claus-Peter Wirth 30

Observations and questions concerning partiality and don't care non-strictness
Uwe Wolter 31

Preface

Specification and semantics are two branches of theoretical computer science on which formalisms for analyzing, constructing and verifying computer software have been successfully built. Over the last few years such formal methods have been increasingly accepted and applied in practice which in turn has stimulated new efforts in research.

In this second edition of the seminar on Specification and Semantics recent scientific results and new research directions in the area of foundations, methods and applications of mathematics-based software development were discussed by more than 40 scientists. The 36 talks focussed in particular on the following topics:

- mathematical foundations of specification and semantics
  including models and logic calculi, concepts of category and type theory, algebraic concepts and theorem provers;

- methods of formal semantics
  including denotational, operational and axiomatic semantics, algebraic and categorical semantics, transition systems and term and graph rewriting;

- approaches to formal specification
  including abstract data types, model-oriented specification, algebraic specification, graphical specification and formal specification languages;

- formal development and verification methods
  including formal requirement analysis and specification, structuring and modularisation techniques, verification and validation of modules and configurations and formal aspects of reusability;

- applications of specification and semantics
  including description of programming languages, methods for software development, parallel and distributed systems, safety-critical systems, modeling and specification languages, data and knowledge based systems.

On behalf of all the participants the organizers would like to thank the staff of Schloß Dagstuhl for providing an excellent environment to the conference.

The Organizers,

Hartmut Ehrig *Friedrich von Henke* *José Meseguer* *Martin Wirsing*

Timed Process Algebra: Theory and Applications

J.C.M. Baeten

Eindhoven University of Technology
P.O.Box 513
5600 MB Eindhoven, The Netherlands
josb@win.tue.nl

This is joint work with Jan Bergstra, University of Amsterdam and Utrecht University.

We give an overview of our work since 1989 on process algebra extended with timing constructs. Based on untimed process algebra, we have extensions with discrete timing (time divided into slices) and dense timing (continuous time). We have variants with relative timing, absolute timing and parametric timing (where relative and absolute are integrated). We have variants with timestamped actions and with two-phase timing (where execution of actions and passage of time are separated). We stress the fact that all these styles are needed, and should interact closely. We have an integrated framework, based on conservative extensions. All timing constructs found in the literature can be translated into this framework.

We mention a number of applications. More information can be found on http://www.win.tue.nl/win/cs/fm/josb.

A Unified Relational Approach to Semantics of Non-deterministic Applicative Programs

Rudolf Berghammer

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität Kiel
Preusserstraße 1–9, D–24105 Kiel, Germany
rub@informatil.uni-kiel.de

The natural meaning of a program written in an applicative language like LISP or ML is a (possibly partial) function. Functions are specific relations, and the idea of relational semantics for applicative programs is to regard the semantics of programs as elements of an (abstract) relational algebra.

In general, relations are neither univalent nor total. Thus, using them rather than functions in semantics, one is able to avoid the complexity introduced by artificial bottom elements denoting undefinedness and has, in addition, natural candidates for modelling

non-determinism. A further reason for using relations is that they can be calculated with so well. Using equational-like reasoning for applicative programs, we can construct proofs that are intuitive, memorable and even machine-checkable.

The two basic ingredients of functional / applicative programs are composition and application. Composition is a built-in operator of relational algebra and based on it application easily can be defined. Other control constructs like conditional and recursive declarations as well as concrete data like truth values and natural numbers must be translated into the language of relations.

The talk is organized as follows. First, we collect relational descriptions of the data domains the programs defined later will operate on. Then, we introduce the idea that an applicative program *is* a relation, and look at some ways of combining simple programs into more complex ones. In the third part we define the formal syntax and relational semantics of a simple non-deterministic applicative language. And, finally, we explore the three different kinds of non-determinism known from the literature and their significance for termination problems.

Architectural Specifications

Michel Bidoit

LIENS, CNRS URA 1327 & Ecole Normale Supérieure
45 rue d'Ulm
F–75230 Paris Cedex 05, France
Michel.Bidoit@ens.fr

In this talk we explain the motivations that have led to the introduction of "architectural specifications" in the specification language currently designed by the Common Framework Initiative for Algebraic Specifications (CoFI, cf. Peter Mosses's talk). Architectural specifications are intended to fulfill the following requirements:

- To be able to describe in a design specification the architecture of the system to be implemented. Thereby this architecture is considered to be a crucial part of the design specification.
- To be able to identify sub-components and to provide to implementation teams specifications of these sub-components as independent implementation tasks. Each subcomponent specification should be implementable independently.
- Any combination of (arbitrary) correct realizations of the component specifications should provide a correct realization of the whole system specification, and only such combinations are indeed correct reazlizations: it is mandatory to respect the architecture as specified in the architectural design specification.

Then we explain why the usual model class semantics for loose specifications is not fine enough, and we provide some ideas on how to define both the syntax and the semantics of architectural specifications.

Evolving Algebras and Parnas Tables

Egon Börger

Dipartimento di Informatica, Universita di Pisa
On sabbatical leave at Siemens Research and Development
München, Germany
boerger@di.unipi.it

We show that Parnas' approach to use function tables for a precise program documentation can be generalized and gentilized in a natural way by using evolving algebras for well documented program development as proposed by us in [BBD+95].

Parnas' function table approach and the evolving algebra approach to program documentation share a large common ground, pragmatically, conceptually and methodologically. Pragmatically, they both aim at supporting the understanding of programs by humans - for purposes like review, maintenance, use by application domain experts. Conceptually, they both are based on the use of functions, of the notion of states and their dynamics as a function of time, and of the distinction of environmental (monitored) and controlled quantities. Methodologically, they both are concerned a) to use only standard mathematical language for providing unambiguous complete descriptions using simple notation, b) to reveal the structure of programs by identifying byilding blocks and by keeping traces of their development, c) to "show" the correctness of programs or program parts by mathematical reasoning.

Evolving algebras offer a fainer grained use of functions (which by the way are allowed to come with an arbitrary finite number of arguments and are classified also into dynamic and static functions). Evolving algebras provide methods for *semantical* structuring of programs (which are based on systematic use of stepwise refinements). In addition by a simple yet precise semantics they yield a safe mathematical foundation for the use of function tables. (This semantics is based only on first-order logic and directly supports the intuitive understanding of programs by progrmamers.)

In this talk we illustrate our claims by translating some typical Parnas tables, taken from the literature, into evolving algebra rules.

References

[BBD+95] C. Beierle, E. Börger, I. Durdanovic, U. Glässer, and E. Riccobene. Refining abstract machine specifications of the steam-boiler control to well documented executable code. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam-Boiler Control*. Springer Lecture Notes In Computer Science, 1995 (to appear).

Proof systems for parametric specifications

María Victoria Cengarle

Ludwig-Maximilians-Universität München
Oettingenstr. 67
80538 München, Germany
cengarle@informatik.uni-muenchen.de

In this talk we consider a kernel ASL language for formal specification over an arbitrary institution whose category of signatures is inclusive, and has initial object, inclusion preserving pushouts, and semi-composable signatures. This language is equipped with the three specification-building operators of amalgamated sum, renaming and restriction (also called export). The semantics of a specification expression is based on the loose approach.

The language is enhanced with simply typed λ-calculus parameterization. Given a sound proof system for the underlying institution, a proof system for parameterized specifications is presented. This system extends a structured proof system for the non-parametric fragment and its judgements are supplied with a context assuming values for variables possibly occurring free in the parameterized specification term. The system is correct, and is moreover complete w.r.t. denotable assumptions if the structured proof system is complete (what depends on the completeness of the proof system chosen for the institution, applicability of the deduction theorem, the interpolation property, the form of formulas, and some other hypothesis on finiteness of formulas and/or inference rules). The proof system permits the derivation of the refinement approach to implementation relation by means of an inference system which again extends a system for non-parametric specification expressions. With these tools combined, we can support the definition of parameter restrictions in terms of refinement, and furthermore their derivation. In this way, the β-equality of λ-terms becomes a derived relation. In other words, parameter verification is performed.

Specification of Concurrent Systems: From Petri Nets to Graph Grammars

Andrea Corradini

Dipartimento di Informatica
Corso Italia 40
Pisa, Italy
andrea@di.unipi.it

Graph Rewriting Systems are a powerful formalism for the specification of parallel and distributed systems, and the corresponding theory is rich of results concerning parallelism and concurrency.

I will review some results of the theory of concurrency for the algebraic approach to graph rewriting, emphasizing the relationship with the theory of Petri nets. In fact, graph rewriting systems can be regarded as a proper generalization of Petri nets, where the current state of a system is described by a graph instead of by a collection of tokens. Recently, this point of view allowed for the generalization to graph rewriting of some interesting results and constructions of the concurrent semantics of nets, including processes, unfoldings, and categorical semantics based on pair of adjoint functors.

First-order Definable Automata

Jorge R. Cuéllar

Siemens, ZFE T SE 1
Munich, Germany
Jorge.Cuellar@zfe.siemens.de

Think of an Evolving Algebra as describing a set of traces. (Not only *one* trace, due to some non-determinism, say through the presence of oracles for the actions of the environment or the indeterminism of the "timing conditions" of agents). A trace is simply a sequence of states (= Σ-Algebras, for some signature Σ). Let us assume that the composition of agents (Evolving Algebras on signatures that may intersect) is defined in such a way that the set of traces of the composition is related to the sets of traces of the agents by the condition that, π is a trace of the composition *iff* when restricted to the local states of each agent, the restriction is a trace of the corresponding agent. If this is the case, and the composition is seen as an Evolving Algebra, then the resulting Evolving Algebra has fairness properties.

This leads us to the following definition: a FOL-Automaton is $(\Pi, \Pi_0, \rightarrow, \mathcal{F}$ where Π is a set of states (algebras), Π_0 a subset of Π (described by a first-order formula) $\rightarrow$ a subset

of $\Pi \times \Pi$, (also described by a first-order formula, as in TLA) and $\mathcal{F}$) a set of *acceptance* (or *fairness*) conditions. A technical point is that the *acceptance* conditions should be *stutter invariant*, due to the fact that composition (as well as refinement) introduce stuttering. This is achieved by the introduction of *observable transition formulas* (a generalization of Lamport's $\langle A \rangle_f$).

Then a definition of traces of FOL-Automata is given, together with a notion of composition (or product, or conjunction) of FOL-Automata. The traces of the composition is the intersection of the traces of the automata. Further, with an appropriate method of hiding (existential quantification), the notion of *refinement* corresponds to trace inclusion.

Moreover, this definition of FOL-Automata may be extended to the case where the Automata "communicate" via "actions" (and not "variables").

One application of the methods described is the use of model-checking and automata-based reasoning for TLA or, in other words, the combination of TLA-Theorem proving and model-checking.

Specifying Concurrent Information Systems Without Specifying Concurrency

Hans-Dieter Ehrich

Technische Universität Braunschweig
Postfach 3329
D-38023 Braunschweig
HD.Ehrich@tu-bs.de

An information system is a reactive, open, and typically distributed system maintaining data bases and application programs. A crucial point is to obtain high-level specification techniques that are suitable for such systems, i.e., that cope with data and programs as well as distribution and concurrency and communication. The talk presents recent ideas on semantic fundamentals. The key idea is not to specify concurrency explicitly but assume it as the basic mode of operation. Explicit specification is given for constraining concurrency, e.g., sequential execution, synchronous interaction, etc. For doing this, a distributed temporal logic is used. Interpretation is given in locally sequential prime event structures. The model categories of interest have final elements that may serve for assigning standard semantics. In-the-large specification concepts like inheritance, hiding, generalization and aggregation can be given semantics as limits and colimits in appropriate model categories.

Synchronization of views of system specifications based on graph transformations

H. Ehrig

FB 13, Technische Universität Berlin
Franklinstr. 28/29
D-10587 Berlin, Germany
ehrig@cs.tu-berlin.de

coauthored by
R. Heckel, U. Wolter (TU Berlin), A. Corradini (Univ. Pisa), G. Engels (Leiden)

This lecture bridges the gap between system specifications based on algebraic specification techniques with those based on graph transformations. In the case of algebraic specifications the synchronization of views can be done using amalgamation, which can be considered as a pushout construction in the Grothendieck category of generalized algebras. The synchronization construction for typed graph transformation systems, on one hand has been developed by Leila Ribeiro according to the requirements of the industrial company NUTEC within a German-Brazil cooperation project, on the other hand turns out to be a pullback construction in the category of graph transformation systems. In fact this category is also a Grothendieck category with respect to a split opfibration functor defined by typed graph transformation systems. Moreover, a loose semantics for graph transformation systems is defined which can be shown to be a cofree functor. Hence it preserves pullbacks which implies compositionality of the loose semantics w.r.t. the synchronization operation.

Specification and Semantics of Software Architectures

José Luiz Fiadeiro

Department of Informatics
Faculty of Sciences, University of Lisbon
Campo Grande, 1700 Lisboa
Portugal
llf@di.fc.ul.pt

We provide a categorical semantics for connectors as used in software architectures, namely by Allen and Garlan, that uses and generalises notions of parameterisation as developed

for abstract data type specification. More concretely, we investigate the applicability of pushout-based notions of parameter passing in the context of categories of, on the one hand, temporal specifications and, on the other hand, COMMUNITY programs. The fact that these two categories are concrete and reflective over categories of signatures is exploited for generalising parameterisation to connectors specified in the form of diagrams. Finally, we extend the notion of connector of Allen and Garlan by taking roles as temporal specifications and the glue as a COMMUNITY program.

An Object Specification Language Implementation with Web User Interface based on Tycoon

Martin Gogolla, Mark Richters

Bremen University
Postfach 330440
D-28334 Bremen
Germany
{gogolla|snurp}@informatik.uni-bremen.de

TROLL *light* is a specification language suitable for the description of structure and behavior of objects in information systems [GCH93, GCD+95]. The language allows to describe the part of the world to be modeled as a community of concurrently existing and communicating objects. Apart from a set-theoretic semantics TROLL *light* has a pure algebraic semantics [GH95]. The main underlying idea is to present a transition system where the states represent the states of the specified information system, and state transitions are caused by the occurrence of finite sets of events.

Tycoon is a persistent programming environment [Mat93, MMS95] developed at Hamburg University. It is an open system in the sense that it allows different methods for persistent storage of objects and programs (for example file systems, relational databases, object-oriented databases, etc.) and different methods for handling of user interfaces (for example TCL/TK, Web Browsers, etc.). The underlying programming language TL has a rich type system, and allows for generic programming and external communication. It is orthogonal with respect to persistence, type completeness, and interator abstraction. TL is strictly typed, has higher order functions, is neutral with respect to the data model, and has a small language kernel. TL is the basis for all services in the Tycoon programming environment.

We concentrate on the user interface of a TROLL *light* implementation based on Tycoon. The implementation employs a normal Web browser (Netscape, Mosaic, etc.) for both the exploration of template, i.e. object type, descriptions (for an example see the window in Fig. 1 displaying an author template) and objects, i.e. instances (for an example

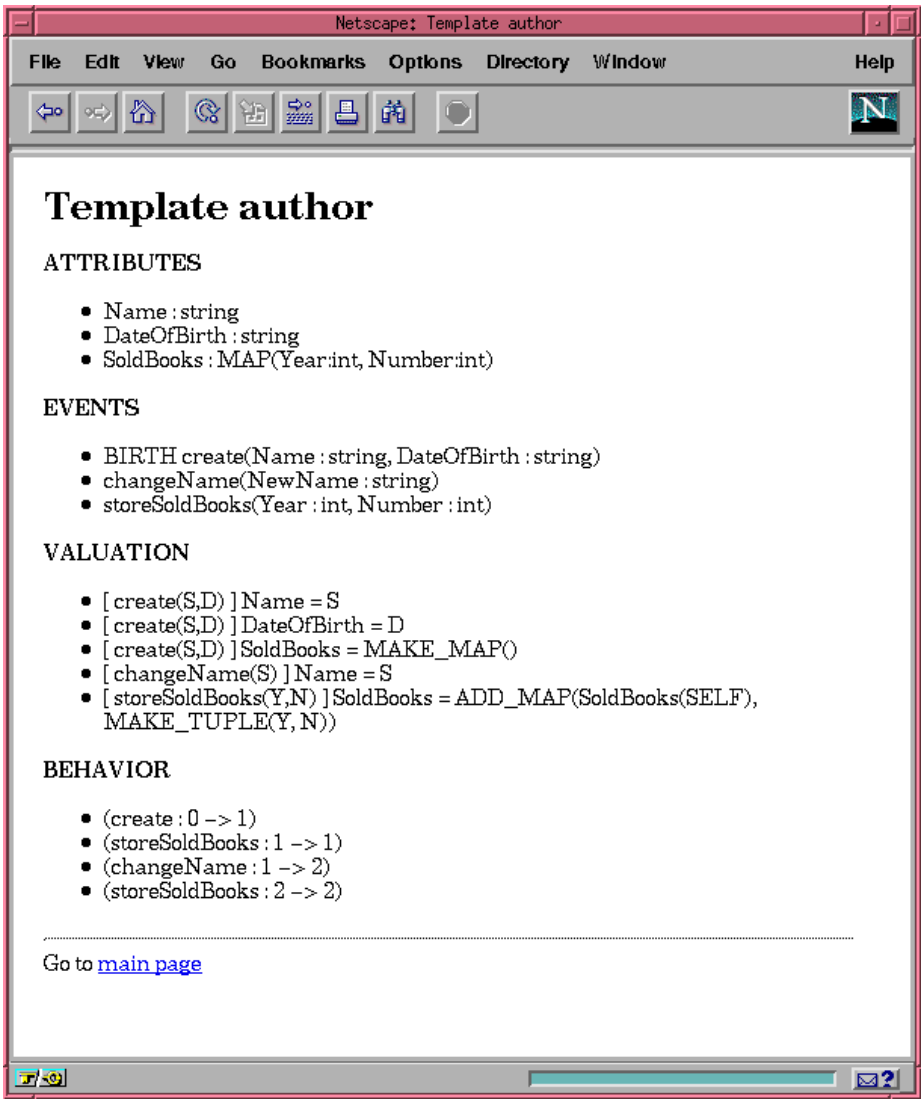

Figure 1: Object template

see the window in Fig. 2 displaying an author object), of these templates. In particular, objects are represented by HTML documents and object references can be followed simply with the Web browser.

References

[GCD+95] M. Gogolla, S. Conrad, G. Denker, R. Herzig, and N. Vlachantonis. A Development Environment for an Object Specification Language. *IEEE Transactions on Knowledge and Data Engineering*, 7(3):505–508, 1995.

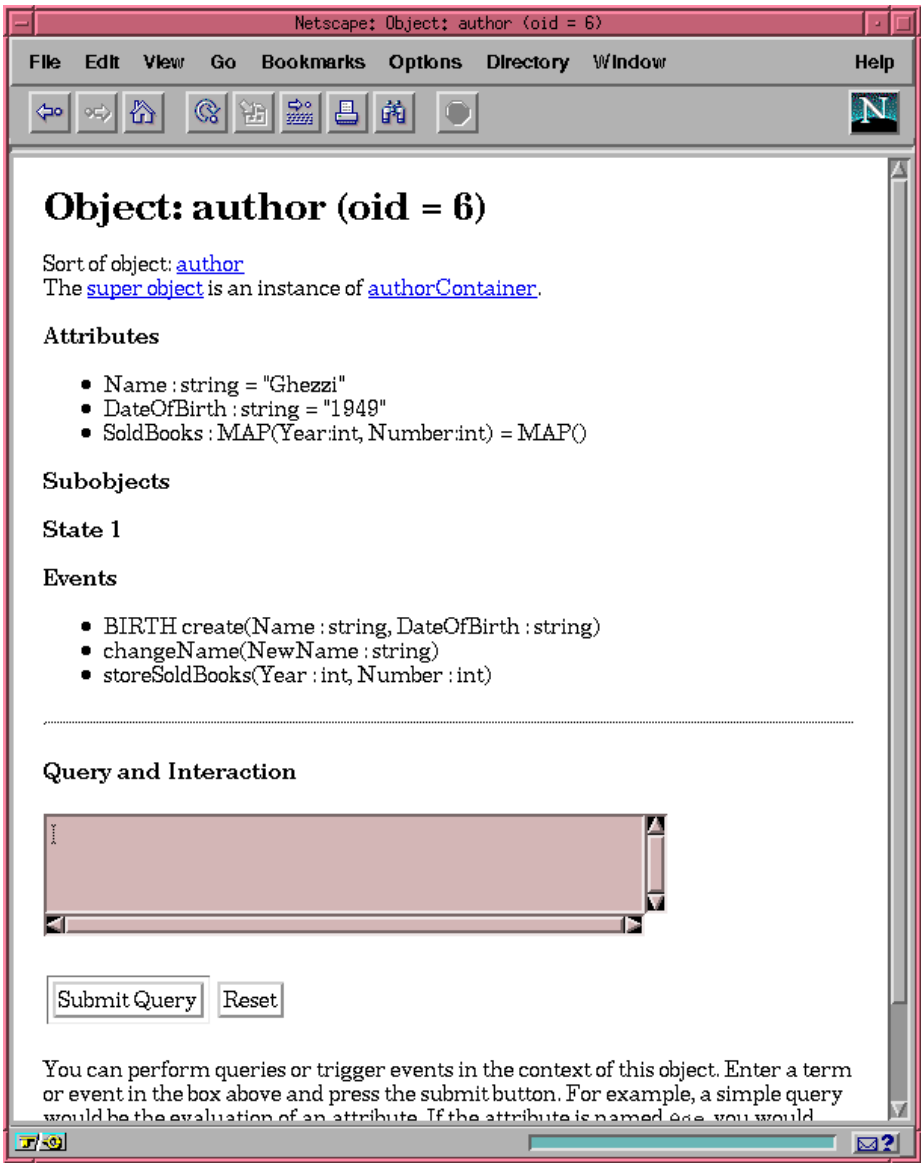

Figure 2: Object instance

[GCH93] M. Gogolla, S. Conrad, and R. Herzig. Sketching Concepts and Computational Model of TROLL light. In A. Miola, editor, *Proc. 3rd Int. Conf. Design and Implementation of Symbolic Computation Systems (DISCO'93)*, pages 17–32. Springer, Berlin, LNCS 722, 1993.

[GH95] M. Gogolla and R. Herzig. An Algebraic Semantics for the Object Specification Language TROLL light. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Type Specification (WADT'94)*, pages 288–304. Springer, Berlin, LNCS 906, 1995.

[Mat93] F. Matthes. *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung.* Springer, Berlin, 1993.

[MMS95] B. Mathiske, F. Matthes, and J.W. Schmidt. Scaling Database Languages to Higher-Order Distributed Programming. In P. Atzeni and V. Tannen, editors, *Proc. of the 5th Int. Workshop on Database Programming Languages (DBPL'95)*, 1995.

Classical Logic and Exception Handling

Philippe de Groote

Inria-Lorraine & CRIN-CNRS
615 rue du jardin botanique, B.P. 101
54602 Villers-lès-Nancy Cedex, FRANCE
`Philippe.de.Groote@loria.fr`

The notion of exception and the one of data type constructor, in Standard ML, are unified. This unification, which follows a proposal by D.B. MacQueen, is based on the special datatype *exn* that stands for the type of exception. Values of type *exn* are first-class citizen: they may be stored, they may be pass as parameters, returned as results, etc. In addition, and contrarily to the other values, they may also be turned into *packets* by being *raised*.

The typing and reduction rules of the operator `raise` are the following:

$$\frac{\Gamma \vdash N : \mathit{exn}}{\Gamma \vdash \mathtt{raise}\, M : \alpha}$$

$$\begin{array}{lcll} V\,(\mathtt{raise}\; M) & \rightarrow & (\mathtt{raise}\; M) & (\text{for } V \text{ a value}) \\ (\mathtt{raise}\; M)\, N & \rightarrow & (\mathtt{raise}\; M) & (\text{for } N \text{ any expression}) \end{array}$$

These rules correspond respectively to the deduction rule and the proof reduction rules that are used in natural deduction for *falsity*.

$$\frac{\bot}{\alpha} \qquad \frac{\begin{array}{cc} \vdots & \vdots \\ \alpha \to \beta & \dfrac{\bot}{\alpha} \end{array}}{\beta} \;\rightarrow\; \frac{\vdots}{\beta} \bot \qquad \frac{\begin{array}{ccc} & \vdots & \\ & \bot & \\ \dfrac{}{\alpha \to \beta} & & \vdots \\ & \alpha & \end{array}}{\beta} \;\rightarrow\; \frac{\vdots}{\beta} \bot$$

This observation allows us to draw the following conclusion: *it makes sense to identify, through the Curry-Howard isomorphism, the type of exceptions (exn) with the logical notion*

of falsity ($\bot$). Let us accept this identification and proceed further with our type-theoretic analysis of exception handling.

Packets, i.e. raised exceptions, are propagated and then possibly handled. The typing rule for exception handlers is akin to the following:

$$\frac{\Gamma \vdash M : \alpha \qquad \Gamma \vdash N : \mathit{exn} \to \alpha}{\Gamma \vdash M \texttt{ handle } N : \alpha}$$

This rule is certainly sound, but not satisfactory. On the one hand, we would like to have a rule that allows exception declarations to be discarded. This is mandatory if we want to preserve the logical consistency of the type system because we have identified the type of exception with falsity. On the other hand, as it is stated, this rule does not reflect the SML exception handling mechanism properly. Indeed in SML the right hand side of the operator `handle` is not an expression but a *match*.

A solution to these two problems is to consider the following typing rule:

$$\frac{\Gamma, y : \neg\alpha \vdash M : \beta \qquad \Gamma, x : \alpha \vdash N : \beta}{\Gamma \vdash \texttt{let } y : \neg\alpha \texttt{ in } M \texttt{ handle } (y\,x) \Rightarrow N \texttt{ end} : \beta}$$

This rule, which is consistent with the definition of SML, corresponds to the elimination of the disjunction for the particular case of the excluded middle. Therefore it is sound with respect to classical logic. This is not too surprising because it is known, since Griffin's work, that there is a strong connection, through the Curry-Howard isomorphism between classical logic and sequential control.

Because classical logic is consistent, it is not possible to build any closed expression of type $\bot$ (that is of type *exn*) by using the above rule. Consequently, our typing system seems to ensure the interesting property that well-typed programs cannot give rise to uncaught exceptions. Unfortunately it is not so because the operational semantics of ML does not fit our interpretation. For this reason, we propose a modified semantics. As far as non-exceptional values are concerned, this modified semantics is equivalent to the original one. Moreover, it ensures that any raised exception is eventually handled.

First Steps Towards an Institution of Algebra Replacement Systems

Martin Große–Rhode

Technische Universität Berlin
Franklinstr. 28/29
Berlin, Germany
mgr@cs.tu-berlin.de

For the specification of interactive (open, distributed) systems a notion of states and state transitions is needed. The purpose of the *Algebra Replacement Systems* – approach is to introduce such notions on top of purely functional algebraic data type specifications in order to allow for a layered system specification that is as functional as possible and adds state changes on top of the functional part. This idea is expressed by the slogan *states are algebras* and *transitions are replacements of (sub–)algebras*. The framework is developed systematically as an institution, that is, signatures, models and morphisms, and axioms (replacement rules) and satisfaction are defined in general. In particular a loose semantics (model category) for replacement rules is defined that allows to interpret rules as requirement specifications instead of concstructions. It can be shown that the constructive interpretation of replacement rules then yields an initial model in the category defined above.

On Proof Systems for Structured Specifications

Rolf Hennicker

Institut für Informatik, Ludwig-Maximilians-Universität München
Oettingenstr. 67
D-80538 München, Germany
hennicke@informatik.uni-muenchen.de

– In cooperation with Michel Bidoit, María Victoria Cengarle, and Martin Wirsing –

Reasoning about specifications is a fundamental task when using formal specifications in program development. For this purpose proof methods are needed that allow one to prove consequences of a specification.

We provide an overview on proof systems for structured ASL-like specifications in a first-order logical framework. Two different kinds of proof systems are considered: non-compositional and structured proof systems. While non-compositional proof systems either

compute a normal form of a given specification or associate a flat, unstructured set of non-logical axioms and rules to a specification, structured proof systems allow one to perform proofs according to the modular structure of a given specification.

First we show that the considered proof systems are sound and (relatively) complete for a kernel specification language that includes basic (flat) specifications, renaming, export (hiding) and the combination of specifications. Then we extend the kernel language by reachability and observability operators and we show that using infinitary first-order sentences and/or infinitary (semi-formal) proof rules the soundness and completeness results carry over to the extended language.

Behavioural abstraction and behavioural satisfaction in higher-order logic, with higher-typed functions

Martin Hofmann and Don Sannella

TH Darmstadt
Schloßgartenstr. 7
D-64289 Darmstadt
Germany
mh@mathematik.th-darmstadt.de

The behavioural semantics of specifications in higher-order logic with function types is analysed. A characterisation of behavioural abstraction based on a reinterpretation of equality as indistinguishability, originally due to Reichel, recently generalised to first-order logic by Bidoit, Hennicker, and Wirsing, and subsequently extended by the authors to higher-order predicate logic, is further generalised to higher-order logic with function types and Hilbert's ϵ-operator. This generalisation enables for example a direct formulation of specifications using recursively defined local functions.

The possibility of having higher-order functions not only in specifications, but also in signatures, is discussed.

Action Refinement and Inheritance of Properties in Systems of Sequential Agents

Michaela Huhn

Institut für Informatik, Universität Hildesheim,
Marienburger Platz 22,
Hildesheim, Germany
huhn@informatik.uni-hildesheim.de

For systems of sequential agents, fundamental relations between events - *causality* and *conflict* - are naturally connected to a global dependency relation on the system's alphabet. *Action refinement* as a strictly hierarchical approach to system design should preserve this connection. It can be shown that bisimulation is a congruence with respect to action refinement and temporal logic specifications can be inherited from the abstract to the concrete level.

To model the behaviour of systems of sequential agents we use synchronisation structures, a subclass of prime event structures respecting localities. Action refinement on synchronisation structures does not strictly inherit causality and conflict as it is done in the standard approach but parameterised with a dependency relation. The dependency relation is derived from the structuring of the system into components which we assume to be fixed already in an early design phase. Then parameterised inheritance of causality allows to execute parts of refinements concurrently even if the corresponding abstract actions are causally ordered which has been shown to be useful in case studies. Parameterised inheritance of conflict may lead to deadlocks on the refined level although the abstract system and the refinement function are deadlock-free. These deadlocks occur if the refinements of different actions interfere and they indicate that additional control structure is required to maintain the behaviour.

To express temporal properties of systems we use νTrPTL, a linear time partial order logic. νTrPTL contains local next operators that correspond exactly to local transitions of the agents. This correspondence is fundamental for the definiton of a family of refinement transformations compatible with the action refinement operator on the models. A refinement transformation is based on regular expressions that describe causal chains leading through a refinement. The regular expressions are used to substitute an abstract modality "Next a" by a sequence of modalities. Under reasonable constraints on the refinement function satisfaction of formulae on the abstract system turns out to be equivalent to satisfaction of the transformed formulae for the refined system.

Describing the Semantics of Concurrent Object-Oriented Languages

C B Jones

Department of Computer Science
Manchester University
Manchester
M13 9PL, UK
cbj@cs.man.ac.uk

This talk will review attempts to ascribe semantics to concurrent object-based languages using both operational (SOS) and translational (to process algebras) approaches. The two semantic approaches will be compared as to their usefulness for gaining an intuitive understanding of the COOL in question. Thus far there is nothing very novel but the question of proving the soundness of some equivalences will be shown to be a non-trivial challenge using either semantics.

The foils can be viewed by ftp'ing to `ftp.cs.man.ac.uk` in `pub/cbj/fools.ps.gz`.

Basic concepts of rule-based specification

Hans-Jörg Kreowski

Universität Bremen
Fachbereich Mathematik/Informatik
Postfach 33 04 40
D-28334 Bremen
kreo@informatik.uni-bremen.de

Deriving configurations from configurations by applying rules is the key notion of any rule-based specification framework (like Chomsky grammars, L-systems, term rewriting, graph transformation, Petri nets, and many others). Moreover, the derivation process may start in certain initial configurations only, and only certain terminal configurations may be accepted as results. In this way, each set of rules together with descriptions of initial and terminal configurations, called a *transformation unit*, specifies a binary relation on configurations. In practical applications, such rule-based specifications may comprise hundreds, thousands, even millions of rules so that one needs structuring principles to build up large systems from small units. For this purpose, a transformation unit is allowed to import a set of items. If each imported item refers to a binary relation on configurations on the semantic level, one may compose derivation steps, applying rules of the transformation

unit, with calls of the semantic relations of the imported items. This yields an *interleaving semantics* for transformation units genralizing the derivation semantics of elementary, unstructured transformation units. To obtain a proper structuring principle, one may assume that the imported items are again transformation units. If the import structure in such a case is hierarchical, the interleaving semantics constructed level by level yields a unique interpretation. But even the assumption of a nested import structure with cycles is meaningful. In this case, one can show that the interleaving semantics considered as an operator on binary semantic relations has got a least fixed point, which can be as limit of iterated interleaving semantics starting from empty relations.

Three Techniques Used in Specification Development

Wei Li

Dept of Computer Science
Beijing University of Aeronautics and Astronautics
Beijing,100083, P.R.China
liwei@cs.sebuaa.ac.cn

The processes of the development of a software system can be described by a sequence of versions of the software system. Its ultimate precise specification of the software system is the limit of sequence of the versions of the software system. Our approach is to introduce the idea of approximation, convergence, and limits to the study on the specification development.

Three key techniques are briefly introduced. They play an important role in building the interactive specification development environments. they are: Fast algorithms on the satisfiability problem in order to guarantee that the new laws added are consistent with the current version; Efficient revision calculus so that the modification for generating a new version can be done as less as possible when error is found in the current version; And good development strategies to make the sequence of versions of the specification converging to its limit as fast as possible.

Rewriting Logic as a Logical and Semantic Framework

Narciso Martí-Oliet (joint work with J. Meseguer)

Departamento de Informática y Automática
Escuela Superior de Informática
Universidad Complutense de Madrid, Spain
narciso@eucmos.sim.ucm.es

Rewriting logic, described in [J. Meseguer, Conditional Rewriting Logic as a Unified Model of Concurrency, *Theoretical Computer Science* **96**, 1992, 73–155], is proposed as a logical framework in which other logics can be represented, and as a semantic framework for the specification of languages and systems.

Using concepts from the theory of general logics introduced in [J. Meseguer, General Logics, in: H.-D. Ebbinghaus *et al.* (eds.), *Logic Colloquium'87*, North-Holland, 1989, 275–329], representations of an object logic $\mathcal{L}$ in a framework logic $\mathcal{F}$ are understood as mappings $\mathcal{L} \to \mathcal{F}$ that translate one logic into the other in a conservative way. The ease with which such maps can be defined for a number of quite different logics of interest, including equational logic, Horn logic with equality, linear logic, logics with quantifiers, and any sequent calculus presentation of a logic for a very general notion of "sequent," is discussed in detail. Representation maps of this kind provide executable and modular specifications of the corresponding object logics within rewriting logic which can be very useful for prototyping purposes.

Regarding the different but related use of rewriting logic as a semantic framework, the straightforward way in which very diverse models of concurrency can be expressed and unified within rewriting logic is emphasized and illustrated with examples such as concurrent object-oriented programming and CCS. The relationship with structural operational semantics, which can be naturally regarded as a special case of rewriting logic, is discussed by means of examples. In addition, the way in which constraint solving fits within the rewriting logic framework is briefly explained. Finally, the use of rewriting logic as a logic of change that overcomes the frame problem in AI is also discussed.

Primitive subtyping $\wedge$ implicit polymorphism $\models$ object-orientation

Stephan Merz

Universität München
Oettingenstr. 67
80538 München, Germany
merz@informatik.uni-muenchen.de

We present a new predicative and decidable type system, called $\mathrm{ML}_{\leq}$ suitable for languages that integrate functional programming and parametric polymorphism in the tradition of ML, and class-based object-oriented programming and higher-order multi-methods in the tradition of CLOS. Instead of using extensible records as a foundation for object-oriented extensions of functional languages, we propose to reinterpret classical datatype declarations as abstract and concrete class declarations, and to replace pattern-matching on run-time values by dynamic dispatch on run-time types. $\mathrm{ML}_{\leq}$ is based on universally quantified polymorphic constrained types, where constraints are conjunctions of inequalities between monotypes built from extensible and partially ordered classes of type constructors. We define subtyping, domains, subdomains, and type application, all of which are based on a basic notion of constraint implication. We give type-checking rules for a small, explicitly typed functional language with multi-methods, show that the resulting system has decidable minimal typing, and prove a subject reduction theorem.

(Joint work with François Bourdoncle, Ecole des Mines, Paris.)

Membership Algebra

José Meseguer

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025, USA
meseguer@csl.sri.com

Membership equational logic is the outcome of a long-term search for greater simplicity, expressiveness and generality in algebraic specification, coupled with the desire for efficient implementability by rewriting, to pass from specifications to efficient declarative programs. It supports partiality, subsorts, operator overloading, and error specification. It can be fruitfully used as a common logical framework for algebraic specification in which

other advanced algebraic specification formalisms can be conservatively translated with particularly good preservation of their model-theoretic properties.

A *signature* in membership equational logic is a triple $\Omega = (K, \Sigma, \pi : S \to K)$ with K a set of *kinds*, (K, Σ) a many-sorted (although it would be better to call it "many-kinded") signature, and $\pi : S \longrightarrow K$ a function from a set of *sorts* S to K. We denote by S_k the inverse image of k along π.

An Ω-*algebra* is then a (K, Σ)-algebra A together with the assignment to each sort $s \in S_k$ of a subset $A_s \subseteq A_k$.

Atomic formulas are either Σ-equations, or *membership assertions* of the form $t : s$, where the term t has kind k and $s \in S_k$. General sentences are Horn clauses on these atomic formulae, quantified by finite sets of K-kinded variables.

Therefore, membership equational logic is a special case of Horn logic with equality so that we have all the usual good properties: soundness and completeness of appropriate rules of deduction, initial and free algebras, relatively free algebras along theory morphisms, and so on.

Intuitively, the elements in sorts are the good, or correct, or nonerror, or defined, elements, whereas the elements without a sort are error elements.

The important point is that, although membership equational logic is a very simple logic, it can faithfully represent very nicely many other logics, even more complex ones, used in algebraic specification. In particular, denoting membership equational logic by *Eqtl*$^:$, we have a conservative map of logics

$$\Phi : \mathit{OSEqtl} \longrightarrow \mathit{Eqtl}^:$$

from (an appropriate version of) order-sorted equational logic to membership equational logic, and also a conservative map

$$\Psi : \mathit{PEqtl} \longrightarrow \mathit{Eqtl}^:$$

from partial equational logic with conditional existence equations to membership equational logic. In this way, both partial and order-sorted algebra are subsumed in membership algebra.

These extensions to membership algebra are both particularly nice, in that they are *bicompatible extensions*, so that for each order-sorted (resp. partial) theory T there is a full inclusion (called the *extension functor*) of the category of algebras of T into the category of membership algebras for $\Phi(T)$ (resp. $\Psi(T)$) that has a right adjoint in the other direction, called the *restriction functor*. It then follows that initial algebras, free algebras, and relatively free algebras—for example, in parameterized constructions—are all preserved by both extension and restriction. Therefore, for all practical purposes we can do our computation and our proof-theoretic and model-theoretic reasoning for order-sorted or partial algebra specifications in their corresponding translations into membership equational logic.

These ideas can be seen as an extension of ideas in order-sorted algebra. They have been efficiently implemented in Maude, in joint work with Steven Eker, Patrick Lincoln,

and Manuel Clavel. They also have a very nice rewriting operational semantics and Knuth-Bendix completion theory, as well as powerful rewriting-based inductive proof techniques, that have been developed in joint work with Jean-Pierre Jouannaud and Adel Bouhoula.

Refining Ideal Behaviours

Bernhard Möller

Institut für Informatik
Universität Augsburg
D-86135 Augsburg, Germany
moeller@uni-augsburg.de

We provide some mathematical properties of *behaviours* of systems, where the individual elements of a behaviour are modeled by *ideals* of a suitable partial order. It is well-known that the associated ideal completion provides a simple way of constructing algebraic cpos. An ideal can be viewed as a set of consistent finite or compact approximations of an object which itself may even be infinite.

We introduce a special way of characterising behaviours through sets of relevant approximations. This is a generalisation of the technique we have used before for the case of streams. Given a subset $P \subseteq M$ of a partial order $(M, \leq)$, we define

$$\mathsf{ide}\, P := \{Q^{\leq} : Q \subseteq P \text{ and } Q \text{ directed}\} ,$$

where $Q^{\leq} := \{x \in M : \exists\, y \in Q : x \leq y\}$ is the downward closure of Q. So $\mathsf{ide}\, P$ is the set of all ideals "spanned" by directed subsets of P. Of particular interest are the infinite, non-compact ideals among these, since they correspond to "lively" runs of the system. We prove a number of distributivity and monotonicity laws for ide and related operators. They are the basis for correct refinement of specifications into implementations.

The Tile Model

Ugo Montanari

Dipartimento di Informatica, University of Pisa
Corso Italia, 40
Pisa, I-56100 Italy
ugo@di.unipi.it

In this paper we introduce a model for a wide class of computational systems, whose behaviours can be described by certain rewriting rules. We gathered our inspiration both from the world of term rewriting, in particular from the *rewriting logic* framework of Meseguer, and of concurrency theory: among the others, the *structured operational semantics* (Plotkin), the *context systems* (Larsen and Xinxin) and the *structured transition systems* (Corradini and Montanari) approaches.

Our model recollects many properties of these sources: first, it provides a compositional way to describe both the states and the sequences of transitions performed by a given system, stressing their distributed nature. Second, a suitable notion of typed proof allows to take into account also those formalisms relying on the notions of *synchronization* and *side-effects* to determine the actual behaviour of a system. Finally, an equivalence relation over sequences of transitions is defined, equipping the system under analysis with a concurrent semantics, where each equivalence class denotes a family of "computationally equivalent" behaviours, intended to correspond to the execution of the same set of (causally unrelated) events.

As a further abstraction step, our model is conveniently represented using double-categories: its operational semantics is recovered with a free construction, by means of a suitable adjunction.

CoFI: The Common Framework Initiative for Algebraic Specification

Peter D. Mosses

BRICS, Department of Computer Science
University of Aarhus
Ny Munkegade bldg. 540
DK-8000 Aarhus C, Denmark
pdmosses@brics.dk

The Common Framework Initiative for Algebraic Specification (CoFI[1]) is an open international collaboration. The main immediate aim is to design a coherent family of algebraic specification languages, based on a critical selection of constructs from the many previously-existing such languages—without sacrificing conceptual clarity of concepts and firm mathematical foundations. The long-term aims include the provision of tools and methods for supporting industrial use of the CoFI languages.

After motivating the CoFI we outline its aim and scopes, and explain one general design decision that has already been taken: that the family of languages should be obtained as restrictions or extensions of a simple general-purpose algebraic specification language. The tentative design of this central language was recently agreed at a meeting of the CoFI Language Design Task Group (Munich, 5–7 July 1996); it includes features such as partial functions, predicates, subsorts, first-order axioms, structured specifications, and constraints. The restricted languages are intended for use with tools (e.g., for prototyping) and may correspond to various existing algebraic specification languages; the extensions will accommodate advanced specification constructs (e.g., for specifying reactive systems).

Up-to-date information about the progress of the CoFI and details of the tentative language design may be obtained from the CoFI Home Page on WWW, URL:

`http://www.brics.dk/Projects/CoFI`.

[1] CoFI is pronounced like 'coffee'.

A monotonic declarative semantics for normal logic programs

Fernando Orejas

Dept. L.S.I, Universitat Politècnica de Catalunya
Pau Gargallo 5
08028 Barcelona, Spain
orejas@lsi.upc.es

In a previous paper we developed a methodology for studying the semantics of several modular constructs in Logic Programming. This methodology was heavily based first on the definition of adequate institutions characterizing the behaviour of the given class of logic programs and then on the existence of several categorical constructs in that institution. In particular, in that paper we studied the class of definite logic programs and sketched how these results could be straightforwardly extended to the class of logic programs with constraints.

In this talk we will present preliminary results for applying this methodology to the class of normal logic programs (i.e. logic programs with negation). In particular, we present a new model-theoretic declarative semantics for normal logic programs that allowed us to define an institution of logic programs with negation coping with the problems of the inherent non-monotonicity of negation in logic programming. In addition, we will show that this institution is semi-exact and has free constructions.

The presentation will consist, first, in showing how to solve the problems of non-monotonicity for the class of propositional programs and, then, in showing how to solve the problems for generalizing the constructions to the first-order case.

Proof of Refinements Revisited

Peter Padawitz

Universität Dortmund
peter@ls5.informatik.uni-dortmund.de

A specification SP' refines or implements a specification SP if a certain class of models of SP' satisfies the axioms of SP. We investigate the particular case where the semantics of SP and SP', respectively, is given by a single model, either an *initial* or a *final* one. Initial models are suitable for *static* data types with free constructors, structural-recursive functions, safety predicates and an object equality that is derivable from the axioms that define

the type. Final models capture the semantics of *dynamic* types with non-free constructors, infinite objects, actions, liveness predicates and an object identity that depends on the observable object *behaviour*. Working with these two model constructions puts a number of powerful proof rules at one's disposal in order to show the correctness of a refinement in the above sense. More precisely, we call the validity of the axioms of SP in the model of SP' the *partial* correctness of the refinement of SP by SP'. *Total* correctness requires a further condition, namely that the union of SP and SP' is consistent w.r.t. SP. This condition is dual to partial correctness. It ensures that adding SP' to SP does not change the model of SP as far as theorems about SP are concerned. Consistency can mostly be reduced to rewrite-oriented properties of $SP \cup SP'$ such as ground confluence and normal form completeness, which in turn can be guaranteed by designing the specifications according to specific axiom schemata.

An Algebraic Approach to Global-Search Algorithms

Peter Pepper

Institut für Kommunikations- und Softwaretechnik
Technische Universität Berlin
Franklinstraße 28/29
Berlin, Germany
pepper@cs.tu-berlin.de

The underlying idea of this approach — which is done jointly with Doug Smith from Kestrel Institute — is to use libraries of specifications and diagrams over these specifications to develop efficient solutions for complex problems. This brings greater flexibility to earlier concepts that have been implemented in the KIDS system.

As illustrating examples with a moderate degree of complexity we use classical backtrack problems such as the 8 queens or a simple scheduling task. Here it can be seen, how the global-search paradigm is combined with constraint propagation in order to achieve smaller and smaller search spaces — which is the key to efficiency.

An essential feature is the introduction of specific higher-order predicates that serve as the basis for the problem specification. But at the same time these predicates also induce the appropriate strategies for the algorithm development. This way we obtain a uniform framework that covers the whole spectrum from problem specification to efficient implementation.

A variant of Quantified Dynamic Logic

Gerard R. Renardel de Lavalette

Department of Computing Science
University of Groningen
P.O.box 800
9700 AV Groningen, the Netherlands
grl@cs.rug.nl

MLCM (Modal Logic of Creation and Modification) is a variant of Quantified Dynamic Logic. It emerged from the development of the wide-spectrum specification language COLD at Philips Research Laboratories Eindhoven (NL) in the ESPRIT project METEOR (1984-1989), sponsored by the European Community.

Like QDL, MLCM is an extension of first order logic with modal operators $[\alpha]$, where α is a program expression. MLCM differs from QDL in that it is not the variable assignment that is being changed by a program, but the interpretation of the signature elements. This idea is also incorporated in Gurevich's Evolving Algebras.

MLCM has the following atomic properties:

- NEW c extends the universe with a new object and makes c refer to it;
- $f(t_1, \ldots, t_n) := t$ changes the value of function f on the arguments $t_1, \ldots, t_n$ to t;
- $p(t_1, \ldots, t_n) :\leftrightarrow A$ changes the value of predicate p on the arguments $t_1, \ldots, t_n$ to the truth value of A.

We present syntax, semantics and a complete axiomatisation of MLCM. Finally we consider an extension of MLCM with a program construct borrowed from Evolving Algebras: α, β (the *join* of α and β) combines the state changes of α and β provided they are consistent. Extending the completeness result to this extension of MLCM is an object of current research.

Mind the gap! Abstract versus concrete models of specifications

Don Sannella

Laboratory for Foundations of Computer Science
Edinburgh University
Edinburgh EH9 3JZ, Scotland
dts@dcs.ed.ac.uk

In the theory of algebraic specifications, many-sorted algebras are used to model programs: the representation of values is arbitrary and operations are modelled as ordinary functions. The theory that underlies the formal development of programs from specifications takes advantage of the many useful properties that these models enjoy.

The models that underlie the semantics of concrete programming languages are different. For example, the semantics of Standard ML uses rather concrete models, where values are represented as closed constructor terms and functions are represented as "closures". The properties of these models are quite different from those of many-sorted algebras.

This discrepancy brings into question the applicability of the theory of specification and formal program development in the context of a concrete programming language, as has been attempted in the Extended ML framework for the formal development of Standard ML programs. We investigate the difference between abstract and concrete models of specifications, inspired by the kind of concrete models used in the semantics of Standard ML, in an attempt to determine the consequences of the discrepancy.

This is joint work with Andrzej Tarlecki (Warsaw University and Polish Academy of Sciences).

Functional Specification Using HOPS

Gunther Schmidt

Universität der Bundeswehr München
Werner-Heisenberg-Weg 39
85577 Neubiberg, Germany
schmidt@informatik.unibw-muenchen.de

The Higher Order Programming System allows editing, moulding, and transforming directed acyclic graphs (DAGs). These DAGs represent either types or programs. In the first case, they are built from constructors 1, variable, $\times$, $+$, $\rightarrow$, recursive data types etc.,

while in the latter they stem from λ-calculus, from the combinator field, or from the product/projection/function pairing and sum/injection/function sum area. Program DAGs are always typed according to the generic typing of their constituents.

An overview was given on programming styles ranging from data flow to functional, applicative, and imperative.

The applications of HOPS programming mentioned include an HTTP demon, a BIBTEX browser with graphical user interaction, a translation of parts of π-calculus to ADA, a generated parser, and monad considerations.

Behavioural satisfaction and equivalence in concrete model categories

Andrzej Tarlecki

Institute of Informatics
Warsaw University
ul. Banacha 2
Warsaw, Poland
tarlecki@mimuw.edu.pl

— joint work with Michel Bidoit —

We use the well-known framework of *concrete categories* to show how much of standard universal algebra may be done in an abstract and still rather intuitive way. This is used to recast the unifying view of behavioural semantics of specifications based on behavioural satisfaction and, respectively, on behavioural equivalence of models abstracting away from many particular features of standard algebras. We also give an explicit representation of behavioural equivalence between models in terms of behavioural correspondences. The work is illustrated not only in the usual algebraic framework but also by the development of the corresponding notions for partial and for regular algebras.

Faster Asynchronous Systems

Walter Vogler

Institut für Informatik, Uni. Augsburg
D-86135 Augsburg, Germany
vogler@informatik.uni-augsburg.de

A testing scenario in the sense of De Nicola and Hennessy is developed to measure the worst-case efficiency of asynchronous systems. The resulting testing-preorder is characterized with a variant of refusal traces and shown to satisfy some properties that make it attractive as a faster-than relation. Finally, one implementation of a bounded buffer is shown to be strictly faster than two others – in contrast to a result obtained with a different approach by Arun-Kumar and Hennessy.

A Formal Approach to Object-Oriented Software Engineering

Martin Wirsing

Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstr. 67, D-80538 Munich, Germany
wirsing@informatik.uni-muenchen.de

Current object-oriented design methods, such as those of Rumbaugh, Shlaer-Mellor, Jacobson and Booch use a combination of diagrammatic notations including object and class diagrams, state transition diagrams and scenarios. Other, academic approaches, such as Reggio's entity algebras, Meseguer's Maude and Ehrich/Sernadas' Troll propose fully formal descriptions for design specifications. Both approaches have their advantages and disadvantages: the informal diagrammatic methods are easier to understand and to apply but they can be ambiguous. Due to the different nature of the employed diagrams and descriptions it is often difficult to get a comprehensive view of all functional and dynamic properties. On the other hand, the formal approaches are more difficult to learn and require mathematical training. But they provide mathematical rigour for analysis and prototyping of designs.

To close partly this gap we propose a combination of algebraic specification with rewriting as integrating formal model for design description. Similar to entity algebras and Maude, the static and functional part of a software system is described by classical algebraic specification whereas the dynamic behaviour is modeled by nondeterministic rewriting. We show that the diagrammatic notations mentioned above can be formally based on

this framework. The construction of subsystems, the design of timing constraints and the control of the flow of messages are other characteristics of our approach. Moreover, the notion of refinement provides a formal tool for checking the correctness of implementations.

Therefore the combination of algebraic specification with rewriting gives a coherent view of object-oriented design and implementation. Formal specification techniques are complementary to diagrammatic ones. The integration of both leads to an improved design and provides new techniques for prototyping and testing.

Inductive Theorem Proving in Partial Positive/Negative-Conditional Equational Specifications

Claus-Peter Wirth

Fb. Informatik, Univ. Kaiserslautern, D-67663 Kaiserslautern, Germany
wirth@informatik.uni-kl.de

In general, the class of models of algebraic specifications given by positive/negative-conditional equations (i.e. universally quantified first-order implications with a single equation in the succedent and a conjunction of positive and negative (i.e. negated) equations in the antecedent) does not contain a minimum model in the sense that it can be mapped to any other model by some homomorphism. We present a constructor-based approach for assigning appropriate semantics to such specifications by introducing two syntactic restrictions: firstly, for each term of a negative equation in a condition, this condition also has to contain a literal requiring the "definedness" of this term; secondly, we restrict the rules whose left-hand sides are constructor terms to have "Horn"-form and to be "constructor-preserving". Under the assumption of confluence, the factor algebra of the term algebra modulo the congruence induced by our reduction relation is a minimal model which is — beyond that — the minimum of all models that do not identify more constructor terms than necessary.

Furthermore, we present some conceivable notions of inductive validity for first-order equational clauses w.r.t. constructor-based partial positive/negative-conditional equational specifications. Monotonicity of validity w.r.t. consistent extension of the specification admits an incremental construction of specifications without destroying the validity of already proved formulas. For inductive validity, monotonicity is essential because — contrary to deductive theorem proving — such extensions are required by the inductive proofs themselves. Therefore it is important that our notions of inductive validity are monotonic w.r.t. consistent extension of the specification.

Finally, we give an overview of our inference system for clausal theorem proving w.r.t.

various kinds of inductive validity in theories specified by constructor-based positive/negative-conditional rule systems that have to be (ground) confluent, but need not be terminating. Our constructor-based approach is well-suited for inductive theorem proving even in the presence of partially defined functions. It provides explicit induction hypotheses and can be instantiated with various wellfounded induction orderings. The soundness proof for the inference system is developed systematically by presenting an abstract frame inference system a priori and then designing each concrete inference rule locally as a subrule of some abstract frame inference rule. While this emphasizes a well structured clear design of the concrete inference system, our fundamental design goal is user-orientation and practical usefulness rather than theoretical elegance. The resulting inference system is comprehensive and quite powerful, but requires a sophisticated concept of proof guidance.

Observations and questions concerning partiality and don't care non-strictness

Uwe Wolter

Technical University Berlin
Sekr. FR 6-1, Franklinstr. 28/29
10587 Berlin, Germany
wolter@cs.tu-berlin.de

The talk addresses the question of an algebraic semantics for functional languages based on algebraic specifications of partial algebras with conditional existence equations. Especially we discuss the relation ships between partial algebras and cpo's. We end up with the following observations and conclusions.

1. A pure algebraic treatment of call-by-name non-strictness is not possible. We need cpo's for this purpose.

2. Strict function combined with a non-strict *if_then_else* can be described within partial algebraic specifications.

3. Partial algebras and cpo's reflect different aspects of functional languages:

   - partial algebras $\Rightarrow$ functions on values
   - cpo's $\Rightarrow$ computations with values

   Both aspects are not fully compatible and sometimes we have to treat both aspects in parallel (strictness analysis).

4. Within cpo-approaches we can not describe intensional function spaces and the combination with imperative procedures. But the algebraic approach covers naturally these features.