Contents

Preface

In this seminar recent scientific results and new research directions in the area of formal foundations of software development were discussed by members of the working group FG 0.1.7 "Specification and Semantics" of the German Society for Computer Science (GI) and other well-known scientists. The 30 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, theorem provers
- methods of formal semantics
  including denotational, operational and axiomatic semantics, algebraic and categorical semantics, transition systems, term and graph rewriting
- approaches to formal specification including abstract data types, model-oriented specification, algebraic specification, graphical specification, formal specification languages
- formal development and verification methods
  including formal requirement analysis and specification, structuring and modularization techniques, verification and validation of modules and configurations, formal aspects of reusability
- applications of specification and semantics
  including description of programming languages, methods for software development, parallel and distributed systems, modeling and specification languages, data and knowledge based systems

On behalf of all 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 Jose Meseguer Martin Wirsing

D-oids Models and languages for dynamic and object systems

Egidio Astesiano

DISI-Universitá di Genova, Italy

(joint work with Elena Zucca)

D-oids are a model for systems of dynamic entities,based on the following features: States are modelled by structures (algebras,usually) in some category; we assume that every structure (called instant structure) has an underlying support consisting in an S-sorted family of sets;

Variations of state are modelled by dynamic operations (also called methods) calls, whose semantics is the most typical features of the approach: the effect of a call is a transformation of the source state into a new state, consisting in a map between the underlying carriers; this map is called "tracking map", since intuitively it keeps track of the identities of the evolving dynamic entities. An appropriate rather natural notion of morphism is given such that the d-oids over a category of instant structures are a category.

The emphasis of this talk is on a recent new result,consisting in the construction of the d-oid of dynamic terms over a dynamic signature and a family of variables. In the case that the instant structures are provided of a fre construction of terms over families of generators,then the d-oid of dynamic terms is a free construction. A most notable feature of the construction is the unique representation ;in other words a canonical representation is provided (no identification) as it happens in the classical static case. It can be shown that usual imperative method expressions can be translated into the language of dynamic terms,which then may be seen as the basis of a kernel language for defining methods with the typical flavour of an applicative language.

Prototyping relational specifications using higher-order objects

Rudolf Berghammer

Universität der Bundeswehr München Neubiberg, Germany

In the last two decades, the axiomatic relation calculus has widely been accepted by computer scientists who view it as a convenient formalism to describe fundamental concepts of programming. Also this talk deals with relation algebra as a formal programming tool.

First, we describe an approach for the development of executable relational specifications out of non-constructive first-order problem descriptions. Here certain functionals on relations (e.g., residuals, symmetric quotients, and functionals for bounds and extremal elements) in combination with a relational description of higher-order objects (e.g., sets, sets of sets, and mappings) play an important rôle.

In the second part of the talk, we give a short description of the computer system RELVIEW and demonstrate how it can be used to perform executable relational specifications.

The algorithms described by the executable specifications frequently may be fairly inefficient compared to hand-made ones. However, they are built-up very quickly and their proofs of correctness are very easy. Furthermore, as we show in the last part of the talk, they can be the starting points for the development of more efficient algorithms using the calculus of relations. Hence, we have the typical situation of rapid prototyping.

Modularity, Strongly Persistent Functors and "Stability"

Michel Bidoit

LIENS, CNRS & Ecole Normale Superieure, Paris, France

(joint work with Rolf Hennicker, Munich University)

In this talk we first briefly recall our approach to modularity w.r.t. the semantics and implementation of software specifications. Based on the stratified loose semantics approach, our specification framework meets the following requirements: the independent construction of implementation for the single constituent parts (modules) of a system specification and the encapsulated development of each implementation part using the principle of stepwise refinement.

We show that implementation and parameterization can be handled within a uniform concept and we prove compatibility theorems like the horizontal composition property.

Then we discuss whether the semantics of a specification module should be a class of functors (as in our approach) or just a class of mappings (as advocated in some other related works). A careful study points out the following facts:

- To be of practical interest, the logical framework considered should include an observability concept.
- If observability is handled through a behavioural approach (by means of some behavioural equivalence between algebras), then the mappings should be "stable" in order to ensure the required compositional properties.

- If observability is handled through an observational approach (by means of an observational satisfaction relation), then "stability" is required as well, but can be expressed by the "persistency" of some "observational morphisms" through the mappings. Hence we show that a "stable mapping" is nothing else but a strongly persistent functor in the category having these observational morphisms as arrows.

Relating two (Parametric) Categorical Semantics for Rule-Based Systems

Andrea Corradini

Dipartimento di Informatica - Pisa, ITALY

J. Meseguer and U. Montanari proposed in [MM 90] a new categorical semantics for P/T Petri nets. Generalizing their approach, two parametric categorical semantics for rulebsed systems have been proposed by J. Meseguer [Mes 92] and by A. Corradini and U. Montanari [CM 92], respectively. Each one of those semantics is applicable to a wide class of formalisms, and both are paramentric: however, their precise relationship have never been explored before. This is the topic of this presentation, which reports on an ongoing research.

After introducing the categorical methodology proposed in [CM 92], we show how it can be applied to Term Rewriting Systems, and that this specific application captures the unconditional case of Meseguers' Rewriting Logic, in the sense that the same class of models is determined. However, on the one hand the conditional part of Rewriting Logic does not seem to fit in the methodology of [CM 92]; on the other hand there are applications of such methodology that determine models which do not seem to be characterizable in a natural way in Rewriting Logic. Thus we claim that the two categorical semantics are in general uncomparable, although they have a significant intersection.

References

[MM 90] J. Meseguer and U. Montanari, Petri Nets are monoids, Info & Co. 88 (1990).

[Mes 92] J. Meseguer, Conditional rewriting logic as a model of concurrency, TCS 96 (1992).

[CM 92] A. Corradini and U. Montanari, An algebraic semantics for structured transition systems and its application to logic programs, TCS 103(1992).

Object Roles and Phases

Hans-Dieter Ehrich

Abteilung Datenbanken, Technische Universität Braunschweig

in cooperation with Ralf Jungclaus and Grit Denker

Object roles are temporary specializations where the object can go through, possibly more than once. An example is a person who can be a patient at times. There may be several roles like patient, employee, tax payer, etc., the phases of which may be entered and left independently, in any order. The TROLL language for specifying communities of concurrent interacting objects offers language features for specifying roles. In his recent dissertation, Ralf Jungclaus offered a close to complete formal semantics of TROLL, based on temporal logic. The semantics of roles, however, as well as that of creation and destruction, was left out - it was not formalizable in OSL, the logic adopted. The talk offers an approach to extend OSL so that roles as well as creation and destruction of objects can be expressed in a natural way. The essential idea is to introduce an "in-scope" predicate for actions, in addition to the "enabled" predicate of OSL. While the latter is a way to cope with nondeterminism as introduced by hiding, the former captures visibility of attributes and actions as governed by creating and destroying objects as well as entering and leaving roles. The relationships between roles are made precise by specification morphisms. Their semantics is given by process morphisms. These morphisms also capture other kinds of object relationships, like generalization, aggregation, and interfaces.

FLEX: A Flexible Extension and Integration Concept for Software Development in KORSO and COMPASS

Hartmut Ehrig

Technical University of Berlin

Motivated by the experience within the ESPRIT BR WG COMPASS and the BMFT-project KORSO (Korrekte Software) we propose a flexible extension and integration concept for software development, called FLEX. In addition to support software development by formal methods the aims of FLEX are mainly to allow a smooth change from classical semi-formal techniques for software development to formal techniques with flexible tool support for all levels of development. These aims lead to a number of requirements for FLEX which are already partially supported by concepts available in the literature, while several other aspects concerning extension and integration of different subtechniques and

languages still have to be investigated. To meet these aims and requirements a general and first specific tasks are defined and it is shown how far recent publications in the area of algebraic specifications are contributions for these tasks already. The general task according to FLEX - which is considered to be a long term project - is to provide generic language, tool and methodology development for KORSO and COMPASS in the spirit of development of generic software systems with flexible extension and interconnection mechanisms.

A Computational Model for TROLL *light*

Martin Gogolla

Technische Universität Braunschweig, Informatik, Abt. Datenbanken

The object specification language TROLL *light* is intended to be used for conceptual modeling of information systems. It is designed to describe the Universe of Discourse (UoD) as a system of concurrently existing and interacting objects, i.e., an object community. TROLL *light* is a dialect of the language TROLL. However, TROLL *light* is not just a subset of TROLL. For example, in TROLL *light* classes are understood as composite objects having the class extension as subobjects. Therefore in contrast to TROLL an extra notion of class is not needed in TROLL *light*. This leads to a more orthogonal use of object descriptions. Over and above that, concepts like class attributes, meta-classes, or heterogeneous classes are inherent in TROLL *light*. Second, TROLL *light* incorporates a query calculus providing a general declarative query facility for object-oriented databases.

The first part of the talk introduces the various language concepts offered by TROLL *light*. TROLL *light* objects have observable properties modeled by attributes, and the behavior of objects is described by events. Possible object observations may be restricted by constraints, whereas event occurrences may be restricted to specified life-cycles. TROLL *light* objects are organized in an object hierarchy established by sub-object relationships. Communication among objects is supported by event calling.

The second part of our paper outlines a simplified computational model for TROLL *light*. After introducing signatures for collections of object types (or templates as they are called in TROLL *light*) we explain how single states of an object community are constructed. By parallel occurrence of a finite set of events the states of object communities change. The object community itself is regarded as a graph where the nodes are the object community states reachable from an initial state and the edges represent transitions between states. We shortly indicate how this operational semantics of TROLL *light* could be also expressed in terms of graph grammars and graph productions.

Temporal Logics and the Semantics of Concurrent Systems

Ursula Goltz

Institut für Informatik, Universität Hildesheim, Germany

joint work with Ruurd Kuiper, Wojciech Penczek

For a design calculus for concurrent systems or, more specifically, for reactive systems, the following components are being proposed:

- a (logical) language for the property oriented specification of systems,
- a system description language to represent abstract and more concrete design specifications,
- models for concurrent systems which are used as semantic domains for both the logical language and the system description language.

Based on such a framework, methods for the construction of correct systems may be developed.

A widely used language for the property oriented specification of reactive systems are temporal logics. For the description of reactive systems, process algebras are being used. Here a link between these two worlds is established by interpreting temporal logics on semantic models which are being used for (causality based) semantics of process algebras. In particular, it is investigated which semantic equivalences are induced by various versions of temporal logics.

U. Goltz, R. Kuiper and W. Penczek: Propositional temporal logics and equivalences. In W.R. Cleaveland, editor, Concur '92, volume 630 of Lecture Notes in Computer Science, pages 222-236. Springer-Verlag, 1992.

Types, Specifications, and Software Development

Friedrich von Henke

Universität Ulm, Germany

We show how a notion of types as specifications can be evolved for both specification in-the-small and specifications in-the-large. This requires an extension of the type system to

include dependent Sigma- and Pi-types, as provided, for instance, in the Extended Calculus of Construction of Luo. In this kind of setting, the semantics part of a specification is expressed by (higher-order) predicates which impose constraints on signatures; a realization then consists of an element of the signature type satisfying the constraints. Using this framework, we indicate how to formalize both presentation and verification of derivation steps in the development of software from specifications.

Towards Correct, Efficient and Reusable Transformational Developments

Bernd Krieg-Brückner, Junbo Liu, Burkhart Wolff and Hui Shi

Universität Bremen, Germany

Transformational development attempts to achieve correct software by integrating the construction and verification process. Refinement is achieved by stepwise application of preconceived, correctness-preserving transformations. These transformations are applied by the system with interactive guidance from the implementor and represent corresponding design decisions. Transformations are correct modulo applicability conditions. Transformation can be seen as a form of deduction, applying transformation rules in a kind of rewriting process.

A semantic framework is called for that allows formal reasoning at the meta-level (correctness of rules, their composition, and tactics) in relation to the semantics of the object language. The framework envisaged tries to abstract from the semantics/logics of a particular object language such that a set of rules need not be adapted and proved over and over again for each new object language. Proofs at the meta-level should be possible with system support to guarantee eventual complete correctness of the approach.

At the object-level, applicability conditions need to be established before a transformation rule can be applied. The requirement for efficiency calls for utmost automation, to relieve the user from superflous and tedious interactive proofs and to allow composition of rules and the use of rules in tactics without the need for intermediate user interaction. Techniques from compiler construction can be applied to evaluate, and to incrementally re-evaluate after application of a rule, such contextconditions in an efficient way. Context information not only encompass static semantic attributes and conditions such as those on typing or visibility of identifiers in the object language, but also e.g. the so-called "local theory" that allows access to visible axioms. This information can be used in interactive or automated proofs, but also in automatic simplification or normalisation and to obtain parameters for a transformation rule implicitly.

One of the greatest challenges is the reusability of transformational developments. One would like to abstract from a particular development (let us say a sequence of transformations) to a class of developments to enable reusability in a similar (but not identical) situation. This calls for powerful matching descriptions and powerful application tactics to alleviate the need for the tedious application of single rules. Transformations are used at the meta-level as well to develop optimised meta-programs (transformation tactics and strategies).

References

Krieg-Brückner, B., Hoffmann, B. (eds.): PROgram development by SPECification and TRAnsformation: The PROSPECTRA Methodology, Language Family, and System. LNCS 680 (1993).

Krieg-Brückner, B., Liu, J., Wolff, B., Shi, H.: Towards Correctness, Efficiency and Reusability of Transformational Developments. KORSO Report, FB3 Informatik, Universitat Bremen, 1993. Extended Abstract in Proc. GI Tagung 1993, Informatik Fachberichte, Springer (1993).

Initial Semantics For Specification Developments

Michael Löwe

Technische Universität Berlin, Germany

The talk summarises the framework for specification development for specifications with loose semantics that has been worked out by Sannella, Wirsing, Broy, and others in the last years: It provides a set of (horizontal) operators for building bigger specifications from smaller pieces and a nice, straightforward, and very attractive notion of (vertical) refinement or development of specifications which is compatible with the horizontal operators. The talk provides first ideas how this framework can be carried over to specifications with initial semantics, functorial refinements and constructive horizontal operators. Constructive horizontal operators are those which do not only build classes of algebras from classes of algebras, but also provide a pointwise mapping which defines how individual objects in the result class can be obtained from individual algebras in the argument classes. We show that, in order to provide free-functor semantics for refinements (which is also called initial semantics for specification developments), the operators "impose" and "derive" have to be adjusted slightly. "Impose" must be restricted to conditional equations (infinite premises are allowed) and "derive" must be extended by a restriction construction allowing only "interface"- reachable algebras as models. In a second step, we sketch further adjustments of the operators to become constructive. Especially the operators "union" and "translate" are problematic in this respect. Finally we outline how these concepts can be generalised

to the parametric case of data type constructors and state the compatibility problem of horizontal and vertical steps as a major issue for future research.

Positive/negative conditional equational specifications (Constraints in built-in algebras and theories)

Klaus Madlener

Universität Kaiserslautern, Germany

(Joint work SFB 314 Project D4)

We study algebraic specifications given by finite sets R of positive/ negative-conditional equations with possible constraints in a built-in theory (i.e. universally quantified first-order implications with a single equation in the succedent and a conjunction of positive and negative (negated) equations and further constraints in the antecedent). The class of models of such a specifica- tion R does not contain in general 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. We introduce two syntactic restrictions: firstly, for a condition to be fulfilled we require the evaluation values of the terms of the negative equations (contraints) to be in the constructor sub-universe which contains the set of evaluation values of all constructor ground terms; secondly, we restrict the constructor equations to have "Horn"-form and to be "constructor-preserving". A reduction relation for R is defined, which allows to generalize the fundamental results for positive-con- ditional rewrite systems, which does not need to be noetherian or restricted to ground terms, and which is monotonic w.r.t. consistent extension of the specifi- cation. Under the assumption of confluence, the factor algebra of the ground term algebra modulo the congruence of the reduction relation is a minimal model which is the minimum of all models that do not identify more constructor ground terms than necessary. To achieve decidability of reducibility we define several kinds of compatibility of R with a reduction ordering and present critical-pair tests for the confluence of the reduction relation. Consequences to hierarchical- rewriting and inductive-proofs are discussed.

Rewriting Logic as a Logical and Semantic Framework

Narciso Martí-Oliet and José Meseguer

SRI International, and
Center for the Study of Language and Information
Stanford University

Rewriting logic [2] 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 [1], 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 is discussed in detail.

Regarding the different but related use of rewriting logic as a semantic framework, the straightforward way in which very different 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 also discussed by means of examples.

References

[1] J. Meseguer, General Logics, in: H.-D. Ebbinghaus *et al.* (eds.), *Logic Colloquium'87*, North-Holland, 1989, pages 275–329.

[2] J. Meseguer, Conditional Rewriting Logic as a Unified Model of Concurrency, *Theoretical Computer Science* **96**, 1992, pages 73–155.

An Algebraic Approach To Program Derivation

Bernhard Möller

Universität Augsburg, Germany

The transformational or calculational approach to program development has by now a long tradition. There one starts from a (possibly non-executable) specification and transforms

it into a (hopefully efficient) program using semantics-preserving rules. Many specifications and derivations, however, suffer from the use of lengthy, obscure and unstructured expressions involving formulae from predicate calculus. This makes writing tedious and error-prone and reading difficult. Frequently, for the sake of completeness, long and boring trivial assertions have to be included. Moreover, the lack of structure leads to large search spaces for supporting tools.

The aim of modern algebraic approaches is to make program specification and calculation more compact and perspicuous. They attempt to identify frequently occurring patterns and to express them by operators satisfying strong algebraic properties. This way the formulas involved become smaller and contain less repetition, which also makes their writing safer and speeds up reading (once one is used to the operators). In addition, the search spaces for machine assistance become smaller, since the search can be directed by the combinations of occurring operators. If one succeeds in finding equational laws, proof chains rather than complicated proof trees result as another advantage. Moreover, frequently aggregations of quantifiers can be packaged into operators; an equational law involving them usually combines a series of inference steps in pure predicate calculus into a single one.

We illustrate these ideas by treating some sorting problems within an algebra of formal languages. Essential operators are (besides the usual set theoretic ones) concatenation, join (i.e. glueing upon a one-letter overlap) and shuffle. Based on the join we define the path closure of a binary relation which, for a partial order, is the set of all ordered words over the underlying alphabet. The set of permutations of a word is defined as the set of all possible shuffles of its letters. Based on these concepts, the specification of the sorting problem is straightforward. We give a number of algebraic properties of path closure, shuffle and permutation set which capture the decompositions involved in various sorting algorithms. Based on these properties we derive mergesort, quicksort and an algorithm for constructing sorted binary trees.

Different sets of operators have been used to derive algorithms on graphs, pointer structures and binary search trees. A long term goal is the construction of a database of such operators, enhanced by indexing and external representation using informal language referring to the intuitive meaning of the operators. Such a component would serve as a "specifier's workbench" as a front end to a formal development tool.

Action Semantics

Peter D. Mosses

Computer Science Dept., Aarhus Univ., Denmark

Action Semantics is a framework that combines some of the best features of denotational semantics, operational semantics, and algebraic specifications. The combinator-based action

notation provided by the framework appears to solve the modularity problems of previous semantic description techniques, and experiments have shown that action semantic descriptions scale up smoothly from simple pedagogical examples to practical programming languages.

In this talk, action semantics is introduced by showing how it can be used to describe a fragment of Standard ML. The intended interpretation of the required part of action notation is explained. The syntax of the described language is then extended with the main constructs of Concurrent ML, and it is shown how the semantic description can be extended to the new constructs – without changing the original semantic equations at all! Finally, the fundamental operational concepts underlying action notation are discussed, and it is argued that the asynchronous communication primitives provided by the standard action notation are more appropriate than synchronous ones would be – even for describing languages which, like CML, are based on the notion of synchronization.

On the Correctness of Modular Systems

M. Navarro, F. Orejas, A. Sanchez

UPC Barcelona, Spain

In modular software design it is expected that the correctness of an implementation of a complete system will be a consequence of the correctness of each module. Very often, this property has been associated to the satisfaction of the so-called horizontal and vertical composition properties, both at the specification and at the programming language levels.

In this presentation, we introduce an abstract framework that allow us to represent, as specific instances, most concrete modular frameworks. In particular, the framework presented is "parameterized" by the specification and programming language formalisms, by the semantic constructs associated to modules and by the behavioural equivalence relation used to define module refinement.

In this framework, it is shown that, to achieve modular correctness, it is sufficient that the given programming language satisfies a property of "stability" with respect to the given behavioural equivalence relation. In particular, modular correctness is shown to be independent of the satisfaction (or not) of the horizontal and vertical composition properties at the specification level.

Specification and Programming: Where is the Difference?

Peter Padawitz

Universität Dortmund, Germany

The talk deals with recent developments of Expander, a prototyping system for declarative programs and the data types they use. The system processes programs, given as Horn clause axioms, and requirements to programs, given as Gentzen clauses of the form gs¡=hs where gs and hs are goal sets. A goal set is a universally quantified disjunction of existentially quantified conjunctions of atomic formulas. An Expander proof of gs¡=hs consists in the concurrent strengthening of gs and weakening of hs until the point where syntactical criteria imply that hs "subsumes" gs. We may then conclude that gs¡=hs is valid with respect to the initial semantics of the underlying data type specification.

Rules applied in the proof process are resolution of atoms, rewriting of terms and narrowing transformations, each followed by automatic simplifications of the current goal sets. So far, simplification steps are partial function and predicate evaluation, constructor equation splitting, beta-reduction and continuation passing. They stem from programming or compilation techniques and thus decrease the gap between specification and programming in that, on the verification side, they enhance the transparency and efficiency of proofs, while, on the prototyping side, they improve the executability of specifications.

Parameterization or Polymorphism or Both?

P. Pepper

Fachbereich Informatik, Technische Universität Berlin, Germany

In algebraic specification languages and in functional programming languages, two competing concepts have evolved for increasing the flexibilty of (strong) typing mechanisms. One is usually referred to under the buzzword "parameterization", and the other under the buzzword "polymorphism". Since both concepts are complementary in their respective strengths and weaknesses, we try here to integrate them into a unified framework (that is influenced by the type classes of Haskell).

On the syntactic level we employ the concept of "origins" (that is already used in the Opal compiler) to cope with the resulting type analyses; the goal here is that most of the traditional techniques can still be used. On the semantic level we aim at a solution that is as simple and orthogonal as possible. Here the concept of comma categories – that has

already been successfully used for the description of heterogeneous algebras – appears to be a promising approach. The talk reports about ongoing research in this direction.

Modelling and Specification of Reactive Systems

Axel Poigné

GMD Bonn, Germany

We present a general framework for event-based behaviour and its specification in particular. Our main observation is that a *logic of events* is needed as well as a *logic of actions*. The former prescribes in fine detail how computations proceed while the latter provides generic scripts for events to happen.

The work is based on a novel, generic model of event-based behaviour, referred to as *event automata*.The requirement is that every state records the events which have occurred so far an that transitions correspond to the occurrence of a unique event.

Our framework provides a new semantic basis for the theory of information systems as proposed by the IsCore ring where "processes" are considered as the basic semantic framework to deal with the dynamic behaviour of objects.

Specification of Concurrent Object Systems

Horst Reichel

Technische Universität Dresden, Germany

Two aspects of specifying concurrent object systems are discussed:

1. Structuring objects (describing the internal structure of objects up to some point);
2. System structure (describing the global state of a concurrent object system and its possible changes).

By means of the projective universal property of object types, which makes object instances to final co-algebras, one has a framework to define and study object building operations like sequential composition and parallel composition. In the second part it is skeched how

the Rewriting Logic of J. Meseguer can be used to specify the behavior of concurrent object systems. As a specific topic the addressing of messages to objects is discussed.

Specification and Verification of Modular Systems

Wolfgang Reif

Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe, Germany

We present the KIV approach to specification and verification of modular sequential systems. KIV stands for *Karlsruhe Interactive Verifier* ([HRS 90, Re 92a], and it is an advanced support tool for correct software development. The basic motivation of this approach is to provide an economic perspective of correct software development. Two major problems are the enormous complexity of the deduction problems that are involved in verifying large systems, and the problem of scaling up solutions in the small to solutions in the large.

To overcome these problems we adopt a strict decompositional design discipline leading to modular software systems with compositional correctness. This is achieved by restricting the designer's freedom to hierarchy persistent structured specifications and hierarchy respecting implementations. As a result the correctness of large systems can be completely reduced to the correctness of its modules. The verification effort for a modular system then becomes linear in the number of its modules ([Re 92b]).

Single modules are verified by translating them into a set of proof obligations formulated in a logic of programs. The set of proof obligations is shown to be necessary and sufficient for the correctness of the module ([Re 92c]). These proof obligations are tackled by a specific proof method that has been implemented in the KIV system. Finally, we report on a number of case studies with the system, and give some statistical performance and productivity results.

References

[HRS 90] M. Heisel, W. Reif, W. Stephan : *Tactical Theorem Proving in Program Verification.* 10th International Conference on Automated Deduction, Kaiserslautern, FRG, Springer LNCS 1990.

[Re 92a] W. Reif : *The KIV-System: Systematic Construction of Verified Software.* 11th Conference on Automated Deduction, Albany, NY, USA, D. Kapur (ed.), Springer LNCS 1992.

[Re 92b] W. Reif : *Verification of Large Software Systems*. Conference on Foundations of Software Technology and Theoretical Computer Science, New Dehli, India, Shyamasundar (ed.), Springer LNCS 1992.

[Re 92c] W. Reif : *Correctness of Generic Modules*. Symposium on Logical Foundations of Computer Science, "Logic at Tver", Tver, Russia, Nerode, Taitslin (eds.), Springer LNCS 1992.

Static and logical correctness conditions in formal development of modular programs

Don Sannella

University of Edinburgh

Formal program development by top-down stepwise refinement involves creation of a sequence of partial solutions, commencing with the original specification of requirements and concluding with an executable program. Each successive version must satisfy certain correctness conditions with respect to the previous version, including both "static" conditions requiring the newer version to define objects having the same names and same types as the older version, and "logical" conditions (so-called "proof obligations") requiring the definitions in the newer version to satisfy properties (e.g. axioms) required by the older version. This paper studies the static and logical correctness conditions that arise in the formal development of *modular* software systems in a framework such as that of Extended ML. Modules are equipped with explicit interfaces, may have hierarchical structure, and may be parameterised; each of these aspects of modularity plays an important and distinct role in formal development of software "in the large". A formal system for development is given in which these static and logical conditions are explicit, and it is shown that these conditions suffice to ensure that any program produced by this means satisfies the original specification.

Ordering Isomorphism Classes of Semantic Domains

Gunther Schmidt

Universität der Bundeswehr München, Germany

The idea of constructing semantic domains using sprouts [1], has been elaborated covering explicit domains, natural extensions, sum, product, function space and recursively defined domains. Domains with sprouts are, as usual, considered modulo isomorphism. The relation A ¡= B is defined to express the existence of an adjoint pair from A to B. For the solutions of recursive domain equations to be uniquely determined, it is necessary that ¡= is an ordering of the isomorphism classes of semantic domains. While some categories proposed do not have this property, we prove such a result for the category of domains with sprouts. They constrast to consistently complete cpos, which restrict the cpo but admit all continuous functions. For domains with sprouts the continuous functions are restricted to those which are in addition sprout-respecting. We also prove that in these categories the ordering ¡= can fully be represented by the system of approximations of the domain in the different sprouts.

[1] Describing semantic domains with spronts. ACTA INFORMATICA 27 (1989) 16 - 25

Formalized Developments in Deva

Martin Simons and Stefan Jähnichen

TU Berlin/GMD First.

Deva is a platform designed to provide formal support for rigorous developments; formal meaning that the correctness of a development has to be checkable by a machine. In a rigorous development — such as a VDM refinement or a Squiggol derivation — one is not forced to provide that amount of detail required by formality: proofs of proof obligations may be omitted and assumptions may be left implicit. In order to stay on the formal, i.e. machine verifiable side, yet retain the "sketchiness" of the rigorous side — without which formal developments seem unrealistic — it would be nice to have a formal framework in which "sketches" of developments may be expressed. An implementation of the formalism would then fill in the necessary details. Deva is a step into that direction. Basically, Deva is a higher-order typed λ-calculus with dependent types in the tradition of the AUTOMATH family of languages, the Calculus of Construction, or the Edinburgh Logical Framework. It is equipped with an *implicit level* to allow sketches of developments to be expressed. The talk introduced the motivation behind Deva, reported on the tool support — which has been used to develop an impressive range of Deva formalizations — and illustrated by

an example drawn from the area of transformational algorithm design (Bird's maximum segment sum derivation) that a Deva *formalization* of that *rigorous* development resembles closely in size and shape the original one.

Recent Results on Actor Semantics

Carolyn Talcott

Stanford University

joint work with Gul Agha, University of Illinois, Ian Mason, Stanford University, Scott Smith, Johns Hopkins University

Actors are reactive independent agents that communicate via message passing. In this talk we report on our investigations of the mathematical properties of actor systems. We begin by presenting an actor language that serves as a concrete example for study. This language is the extension of a simple higher-order functional language with primitives for actors: creation; communication; and change of behavior. We define a notion of actor system, and a set of labelled transitions on actor systems. This gives an operational semantics. We then consider two notions of equivalence of actor expressions and systems: the first is a form of observational/testing equivalence; the second is a form of trace equivalence called interaction equivalence. In the full paper [soon available by request tp clt@sail.stanford.edu] many laws of equivalence are given. In addition some general methods for establishing equivalence are developed. The study of expression equivalence is a step towards establishing a sound semantic basis for implementation of such languages. The study of actor system equivalence is a step towards developing a calculus of actor systems.

Timed Testing of Concurrent Systems

Walter Vogler

Institut für Mathematik, Universität Augsburg, Germany

My concern are timing considerations for concurrent systems where the time needed for the individual actions is not known beforehand; it has long been suspected that partial order semantics is useful here. I develop a suitable testing scenario to study this idea. With some view of timed behaviour, I can confirm that interval semiword semantics, a special partial order semantics, is indeed useful. With another view, the testing scenario leads to timed-refusal-trace semantics, where no relation to partial order semantics is obvious.

Untyped Constrained Lambda Calculus

Martin Wirsing

Institut für Informatik, Universität München

joint work with John N. Crossley, Monash University, Melbourne, Australia and Luis Mandel, TU München, Germany

The aim of the present work is to develop an enhancement of the traditional untyped lambda-Calculus in a general setting.

For that purpose terms of the form {C}M are introduced where {C} denotes a constraint and M another (possibly constrained) lambda term, and rules are added for describing the interaction between constraints and lambda terms. The constraints are used in two ways: on one hand, the passive use of a constraint {C} in {C}M restricts the universe of validity of M; on the other hand, the active use of an inference that is made by the constraint solver from the constraint {C} causes the replacement of subterms in the target term M.

The constraint language is considered as a black-box. For different solvers (over possibly different domains) we get different instantiations of the constraint lambda-calculus. Typical examples for constraint domains are linear equations, finite domains, logic programs and also assignment constraints. For constraints which admit only unique solutions the calculus is locally confluent i.e. it satisfies the Weak Church Rosser property. A denotational semantics can be given in the usual way based on the solution of a domain equation.

Categorical Concepts for Parameterized Partial Specifications

Uwe Wolter

TU Berlin, Germany

Joint Work with Ingo Claßen and Martin Große–Rhode

We show that canonical finite limits are the underlying categorical structure of algebras with strict partial operations and conditional existence equations.

Specifications are shown to be equivalent to freely generated categories with canonical finite limits and partial algebras are functors preserving these finite limits. Further also specification morphisms turn out to be finite limit preserving functors between the finite limit categories freely generated by specifications.

Using this categorical description we obtain straigthforwardly a theory of parameterized specifications with free semantics and amalgamation thus providing a partial algebra semantics for specification languages like ACT ONE, ACT TWO,....