REPORT ON THE 5TH INTERNATIONAL WORKSHOP ON THE SEMANTICS OF PROGRAMMING LANGUAGES IN BAD HONNEF, March 11 - 15, 1985

A 85/09

Edited by

Krzysztof Apt, Université Paris VII
Klaus Indermark, Technische Hochschule Aachen
Jacques Loeckx, Universität des Saarlandes

REPORT ON THE 5TH INTERNATIONAL WORKSHOP ON THE SEMANTICS OF PROGRAMMING LANGUAGES IN BAD HONNEF

The Workshop was held in Bad Honnef, BRD, March 11 - 15, 1985, and organized by Krzysztof Apt, Klaus Indermark and Jacques Loeckx. There were several cool and rainy days in Bad Honnef during that early Spring week. The conference site, the Physik-Zentrum, provided an authentic, cozy atmosphere for contemplative science -- pictures from the Twenties of Einstein and collaborators at the "Zentrum" lined the dark corridors. Food and accommodations can politely be called austere: certainly more modest than at the Oberwolfach conference center, and not comparable to the first-class hotels where the American STOC, FOCS and POPL meetings are typically held (at many times the cost, of course).

Still, I had a more exciting scientific experience that week than at any of these larger meetings, which should not have surprised me. After all, the fun of our academic research-oriented life-style, for me at least, comes from the chance to trade ideas with other researchers who are deeply into the same problems. When a talented group of active researchers are carefully selected and brought together as they were in this workshop, neither the weather, food, or rooms matters much.

While I love the opportunity to argue about the significance and quality of papers, and thrive on the opportunity to button-hole an author in the hall and learn about his work by asking him questions about it, I've learned over the years that listening to prepared lectures is much less useful to me. So I rarely attend more than a couple a day, which makes me unfit as a balanced reviewer. Having thus warned you, dear reader, that my views are absolutely idiosyncratic, reflecting personal interests and random attendence, I will say that I was most interested in the on-going work reported by BERRY on real-time programming, PLOTKIN on new foundations for domain theory,

KLOP on comparative concurrency semantics, JONES on bootstrapping methodology, MESEGUER on equationally based programming languages, BROOKES on matching correctness assertions to program assertions, DERSHOWITZ and LESCANNE on rewrite rule systems, and DE BAKKER on data flow semantics.

The abstracts below don't convey the vitality of the workshop atmosphere, but will give a more accurate picture of the complete activities there. I was delighted to attend and look forward to repeating the experience.

Albert R. Meyer
Cambridge, Massachusetts
July 7, 1985

A MICROPROGRAMMING LOGIC

Werner Damm (Aachen)

We present a universal syntax-directed proof-system for the verification of horizontal computer-architectures.
The system is based on the axiomatic architecture description language AADL, which is sufficiently rich to allow the specification of target-architectures while providing a concise model for clocked microarchitectures.
For each description A $\in$ AADL of a host we show how to systematically construct a (Hoare-style) axiomatic definition of an A-dependent high-level microprogramming-language based on S*.
The axiomatization of A's microoperations together with a powerful proof-rule dealing with the inherent low-level parallelism of horizontal architectures allow for a complete axiomatic treatment of the timing behaviour and dynamic conflicts of microprograms written in S*(A).

SPECIFICATIONS IN AN ARBITRARY INSTITUTION

Andrzej Tarlecki (Edinburgh)
(joint work with Don Sanella)

The pioneering papers on algebraic specification used many-sorted equational logic as a logical framework in which specifications were written and analysed. Nowadays, however, examples of logical systems in use include first-order logic (with and without equality), Horn-clause logic, higher-order logic, infinitary logic, temporal logic and many others. Note that all these logical systems may be considered with or without predicates, admitting partial operations or not. This leads to different concepts of signature and of model, perhaps even more obvious in examples like polymorphic signatures, order-sorted signatures, continuous algebras or error algebras.

The informal notion of logical system has been formalised by Goguen and Burstall who introduced for this purpose the notion of *institution*.

The first and presumably most important application of the notion of institution is its use in the theory of algebraic specifications. It turns out that most of the work on algebraic specifications, especially concerning specification languages, may be done in an institution-independent way. We present a collection of simple but very powerful specification-building operations and give their semantics in an arbitrary institution. In this context we outline a very simple and mathematically elegant view of the formal development of programs from their specifications.

We also show how to use the framework of an arbitrary institution to formulate (and prove) some classical model-theoretic results at an appropriately general level. Finally, we briefly discuss the need for some tools for constructing new institutions and for combining institutions ("putting institutions together").

THE ESTEREL SYNCHRONOUS PROGRAMMING LANGUAGE AND ITS MATHEMATICAL SEMANTICS

Gérard Berry (Sophia-Antipolis)
(joint work with Laurent Cosserat)

We present the real-time programming language ESTEREL and its mathematical semantics. Contrarily to CSP-like languages, ESTEREL is a synchronous deterministic language based on a multiform notion of time and where communication is done by broadcasting signals conveying possibly values. The basic idea is to program a conceptually infinitely fast machine which can react instantaneously to external signals and produce itself signals.

The mathematical semantics is a structural operational semantics given by conditional rewrite rules in Plotkin's style, and it characterizes completely the termporal behavior of programs. There are indeed three semantics: a static semantics which detects "races" in programs, a behavioral semantics which defines the

behavior of a program in a non-constructive way, and a computational semantics which is a constructive version of the behavioral semantics using a wavefront propagation algorithm.

The ESTEREL compiler implements a translation of ESTEREL programs into small and efficient finite control automata, using both the mathemtical semantics and a residual algorithm similar to Brozowski's algorithm for translating regular expressions into finite automata. The fact that the resulting automata are (usually) small depends critically on the synchronous nature of the language.

FAIRNESS FOR FORK-JOIN PARALLELISM

Irène Guessarian (Paris)

We address the issue of fairness for fork-join parallelism. Fork join parallelism models for instance a synchronous merge and more general synchronous sets. Surprisingly, fairness turns out to lead to problems quite different from the fairness for asynchronous sets. We define a global and a local notion of fairness, show that they are incomparable. The global notion of fairness can be described via oracles and rational languages on infinite words. Finally, we show that there exist no "most general" fair behaviour for synchronous nets.

NONREGULAR PDL: A RECENT RESULT + OPEN PROBLEMS

David Harel (Rehovot/Israel)
(joint work with M. Paterson)

A specialized recurring domino problem is introduced and shown to be highly undecidable (π_1^1-complete). It involves tiling the strict upper positive octant of the integer plane with a particular domino appearing in each row/column combination. This problem, coupled with some simple number-theoretic properties of powers of 2, is used to prove that PDL with the new program

$L = \{a^i \mid i \text{ is a power of } 2\}$ is highly undecidable (π_1^1-complete). The proof is based on representing the sums $\{2^i + 2^j \mid i, j \geq o\}$ as the upper octant of the integer grid, and finding ways to "reach" the upper and right-hand neighbor of points therein, and to single out the row/column combinations.

The result, and the way its proof depends so heavily on the exponential nature of L, seem to strengthen interest in various open problems, notably the status of the decision problem for PDL + $\{L_p\}$, for various polynomials P, where $L_p = \{a^{P(i)} \mid i \geq o\}$.

(During the workshop Harel and A. Pnueli showed that the result holds for all fixed exponential programs, i.e., $\forall k$, PDL + L_k is π_1^1-complete, where $L_k = \{a^i \mid i \text{ a power of } k\}$.)

SOLVING RECURSIVE EQUATIONS BY ITERATIVELY DEFINED FUCTIONALS

Corrado Boehm (Rome)

Following results contained in [1] are first illustrated: If data systems are given as heterogeneous free(anarchic) algebras and if second order typed lambda-calculus is accepted as a programming language then automatic synthesis paradigms exist both for representing elements of - and total functions belonging to the class of iteratively definable functions on - such algebras.

As examples of free algebras, Booleans, natural numbers and binary strings are exhibited together with some simple application of the paradigms.

MAIN RESULT. Three schemas for recursive definition of binary functions or predicates on N x N, N x B and N x N are proposed. They share the property that each binary operator is curried (i. e. it is considered as a functional which applied to the first argument produces a unary function as result) and they can be solved once for all by a double iteration.

Many program schemas or single programs like bounded <u>while</u>, conversion of integers into binary strings, generalized numbers, primitive recursion, etc. are proved to be instances of some of the given schemas. The descriptive power of the best

schema, due to D. Fragassi, goes beyond the primitive recursive functions.

[1] C. Boehm and A. Berarducci "Automatic synthesis of typed λ-programs on term algebras", submitted for publication to T.C.S.

TYPES AND PARTIAL FUNCTIONS

Gordon Plotkin (Edinburgh)

We present a reformulation of Scott's theory with partial instead of total functions. That is, we drop ⊥ as nontermination (which corresponds to undefinedness instead - as in many classical treatments, especially that of Kleene) but retain it when it occurs as "no information". More generally elements of data types are thought of as data and not as computations for data. Mathematically, we treat the category of predomains (cpos less ⊥) and partial continuous functions (continuous functions on open subsets). We show this permits the usual development of product, sum , function-space, lifting and the solution of recursive domain equations. Further there is a suitable metalanguage which permits an operational semantics (at all types). This is call-by-value but because of lifting, which enables computations to be treated as data, call-by-name and lazy data types are included too. The main theorem is that indeed for this language undefinedness and nontermination coincide (at all types).

SYNCHRONIZATION TREE LOGICS

Joseph Sifakis (Grenoble)
(joint work with S. Graf)

We present a logic, called synchronization tree logic, for the specification and proof of programs described in a simple term language obtained from a constant Nil by using a set A of unary operators, a binary operation + and recursion. The elements of A represent names of actions, + represents non deterministic

choice, and Nil is the program performing no action.

The language of formulas of the logic proposed, contains the term language used for the description of programs, i.e. programs are formulas of the logic. This provides a uniform frame to deal with programs and their properties as the verification of an assertion $t \models f$ (t is a program, f is a formula) is reduced to the proof of the validity of the formula $t \supset f$.

We propose a sound and under some conditions complete deductive system for synchronization tree logics and discuss their relation with modal logics used for the specification of programs.

IMPLEMENTATION PROBLEMS OF PROGRAMMING LANGUAGES PROPER FOR HIERARCHICAL DATA TYPES

Hans Langmaack (Kiel)

A programming language proper for Hierarchical Data Types is LOGLAN. It is an extension of SIMULA 67 and especially allows prefixing of modules by classes at many levels. This language construct entails semantics specification and implementation problems. Based on the notion of static scope (ALGOL-like) expansion of programs induced by procedure calls, class initializations and prefix eliminations a pure static scope semantics of LOGLAN-programs can be defined in the algebraic style of the Paris school of Nivat-Guessarian. For implementation a new principle of associating lists of display register numbers to modules is introduced by means of complement modules. The number of necessary display registers is bounded by the height of the nesting tree of program modules. The proposed scheme of addressing does not cause display register reloadings while computing in one prefix chain. This pure static scope implementation of prefixing at many levels is more efficient at run time than the existing implementation of LOGLAN with its quasi static scope semantics because the number of necessary display registers is given by the number of modules in a program.

AN EXPERIMENT IN PARTIAL EVALUATION: THE GENERATION OF A COMPILER GENERATOR

Neil Jones (Copenhagen)
(joint work with Peter Sestoft, Harald Sondergaard)

A partial evaluator is a program (call it mix) written in a programming language L, which takes as input a program p and a known value d_1 of p's first input argument. It produces as output a so-called residual program:

$$\text{resid} = \text{L mix } \langle p, d_1 \rangle$$

which, if run on p's remaining input $d_2,\ldots,d_n$, will yield the same result as if p were run on all of its input data. (Notation: L l $\langle d_1,\ldots,d_n \rangle$ denotes the output (if any) obtained by running L-program l on input data $d_1,\ldots,d_n$).

Let int be an interpreter written in L which implements another programming language S. Futamura argued in 1971 that partial evaluation could be used to compile from S into L:

$$\text{target} = \text{L mix } \langle \text{int}, s \rangle$$

(where s is the S source program) and even to generate a compiler:

$$\text{comp} = \text{L mix } \langle \text{mix}, \text{int} \rangle$$

by partially evaluating the partial evaluator itself. The logical next step is the generation of a compiler generator by:

$$\text{cocom} = \text{L mix } \langle \text{mix}, \text{mix} \rangle$$

To our knowledge these promising constructions had not been carried out in practice prior to summer 1984, although some work in this direction has been done by Beckman *et al*, Ershov, and Turchin. This paper describes what we believe to be the first running compiler generator ever automatically constructed by means of partial evaluation.

The compilers produced turn out to be natural in structure, reasonably efficient, and to produce reasonably efficient target programs, which typically run about 20 times as fast as the interpreted source programs. The language L is a subset of pure Lisp, and a program is a system of functions defined by mutual recursion. A residual program consists of a set of variants of the original equations, each specialized to certain known argument values.

Partial evaluation is done in two stages. First, the program is flow analyzed by a simple abstract interpretation, yielding as output a heavily annotated version of the same program. Second, the annotated version is transformed into a residual program by applying rewriting rules.

The paper contains an analysis of some rather subtle problems which had to be solved before a running mix could be constructed. These included developing an adequate strategy for unfolding function calls, and an analysis of cause and effect in the computation of L mix <mix,int>, particularly as regards the size and speed of the resulting compiler.

PRINCIPLES OF OBJ2

José Meseguer (SRI International)

The talk described joint work with K. Futatsugi, J.A. Goguen and J.-P. Jouannaud on the OBJ2 Programming Language [1].

Besides implementing equational logic, including of course Abstract Data Types, OBJ2 has two main new semantic ideas:

1. semantic - not just syntactic - interface specifications for generics
   - using "pushout" semantics for theories, as in CLEAR,
   - which supports reliable reusability of modules, and
2. subsorts, which support
   - multiple inheritance
   - both compile and run time error parse error detection
   - specification of partial operations
   - polymorphism, and
   - exception handling

Subsorts and generics give much of the syntactic flexibility of untyped languages, in particular,
- runtime typechecking and
- polymorphism

plus all advantages of strong typing.

Besides giving an overview of OBJ2, its mathematical and operational semantics, both based on the theory of order-sorted algebras [2], [3] were discussed. During the last part of the talk, the language Eqlog [4], which contains both OBJ2 and Prolog as sublanguages, was presented. Eqlog unifies functional programming and Horn-clause programming in a single logical framework (Horn-clause logic with equality). Its operational semantics is logically complete, and allows computation of functions (via rewrite-rules), querying of predicates (à la Prolog, but using narrowing instead of ordinary unification), and solution of equations (by narrowing). Eqlog has parameterized user-definable abstract data types (à la CLEAR) and inheritace, all with a rigorous semantics.

[1] Futatsugi, K., Goguen, J-A., Jouannaud, J.P., Meseguer, J., "Principles of OBJ2", Proc. 1985 POPL Conference, 52 - 66, ACM

[2] J.A. Goguen, J. Meseguer, "Order-Sorted Algebra I", SRI Tech. Report, 1985

[3] J.A. Goguen, J.-P. Jouannaud, J. Meseguer, "Operational Semantics for Order-Sorted Algebra", to appear in ICALP '85

[4] J.A. Goguen, J. Meseguer, "Equality, Tapes, Modules, and (why not?) Generics for Logic Programming", J. Logic Programming, 2: 179 - 210, 1984

FUNCTIONAL DEPENDENCIES BETWEEN OBJECTS IN INTERACTIVE PROGRAMS

Harald Ganzinger (Dortmund)

Interactive programs maintain collections of objects. These objects are not totally unrelated. Some objects may be representations of others. Updating an object requires to subsequently update those objects which depend on the former. In many cases these dependencies are of a functional nature. Examples include

the relation between target and source programs and the representation of a piece of text on a display depending on an internal text object together with formatting information. We propose that such programs explicitly notify the system's kernel about all currently existing dependencies. This allows the kernel to automatically update object representations upon changes. Such a mechanism must allow for dynamically changing the dependency graphs. Incremental updating requires, additionally, knowledge about the current position of complex objects (e.g. display) into logical submits (e.g. windows). This position itself may change in time, leeding to a multi-level model of object dependencies.

AN AXIOMATIC TREATMENT OF A PARALLEL PROGRAMMING LANGUAGE

Stephen D. Brookes (Pittsburgh)

The talk described a semantically-based axiomatic treatment of a parallel programming language with shared variable concurrency and conditional critical regions, essentially the language discussed by Owicki and Gries. We use a structural operational semantics for this language, based on work of Hennessy and Plotkin, and we use the semantic structure to suggest a class of assertions for expressing properties of commands. We then define syntactic operations on assertions which correspond precisely to syntactic constructs of the programming language; in particular, we define sequential and parallel composition of assertions. This enables us to design a truly compositional proof system for program properties. Our proof system is sound and relatively complete. We examine the relationship between our proof system and the Owicki-Gries proof system. Our assertions are more expressive than Owicki's, and her *proof outlines* correspond roughly to a special subset of our assertion language. Owicki's parallel rule can be thought of as being based on a slightly different form of parallel composition of assertions; our form does not require *interference-freedom*, and our proof system is relatively complete without the need for auxiliary variables. Connections with other work, including the "Generalized Hoare Logic" of Lamport and Schneider, and with the Transition Logic of Gerth, are discussed briefly.

COMPOSITIONAL SEMANTICS FOR REAL-TIME DISTRIBUTED COMPUTING

Willem P. de Roever (Nijmegen, Utrecht)

A compositional denotational semantics for a real-time distributed language is given, based on linear history semantics. Concurrent execution ist not modelled by interleaving but by an extension of the maximal parallelism model of Saluschi's, that allows for the modelling of transmission time for communications. The importance of constructing a compositional semantics (and in general a compositional proof theory) is stressed from the point of view of layered top-down design of real-time programs, since the connection between a compositional semantics and a compositional proof theory is a close one, as argumented in the work of Soundararajan.

A SEMANTIC ALGEBRA FOR PASCAL

Peter Mosses (Aarhus)
(joint work with David Watt)

Existing standards and reference manuals for programming languages usually give formal (=mathematically-precise) specifications of syntax, but semantics is specified informally, in natural language. Why are language designers and standardizers so reluctant to use formal semantics - in particular, denotational semantics? Perhaps because the emphasis in the development of denotational semantics has been on obtaining the "right" denotations, regardless of the notation used to specify them. It is not easy for the reader of a denotational description to deduce the operational properties of constructs (e.g. order of evaluation, scope rules) from their denotations. Moreover, there are some pragmatic difficulties concerning modifiability and modularity.

We advocate the use of semantic algebras to improve the pragmatic aspects of denotational semantics. A semantic algebra is just an abstract data type whose values include "actions" (as well as ordinary data), and whose operations include "conbinators" corresponding to simple ways of putting actions together (e.g.

sequentially). It is possible to obtain semantic algebras with actions by systematic combination of standard semantic algebras with purely functional, imperative and declarative actions.

We have obtained a semantic algebra with actions that correspond to the statements, expressions, declarations, etc. of PASCAL. (There are about 30 action combinators and constants, which is not so bad!). We are currently fomulating semantic equations for PASCAL, with the aim of demonstrating that the essence of the recent ISO Standard can be captured in a comprehensible, formal specification.

COMPLETENESS OF COMPLETION

Nachum Dershowitz (Urbana)
(joint work with Leo Marcus, Andrzej Tarlecki)

We mention some problems concerning the construction of term rewriting systems, specifically by the Knuth-Bendix completion procedure, and their connection with equational theories. We look for conditions that might ensure the existence of a finite canonical rewriting system for a given equational theory and that might guarantee that the completion procedure will find it. We examine uniqueness of term-rewriting systems and the need for backtracking in implementing Knuth-Bendix.

THE NOTIONS OF SYMMETRY AND GENERICITY: THEIR APPLICATION TO CSP DISTRIBUTED SYSTEMS

Luc Bougé (Paris)

What criteria are to be considered for assessing quality of a given distributed system? Traditionally, efficiency as measured by bit/message complexity is required almost exclusively. We propose here another family of criteria: knowledge-based criteria. Those criteria analyse distributed systems in terms of knowledge. Initially, each process has some piece of knowledge. A process may increase its knowledge only through explicit interactions with

its environment, namely exchanges of message.

Symmetry expresses that knowledge transfers within the system are isotrope. If some direction is privileged by some computation then some other direction could be privileged as well by some other computation. Previous definitions of symmetry were syntactical and suffer thus several flaws. We propose a semantical definition in the framework of CSP. We have studied the existence of symmetric electoral system and we have obtained several positive and negative results. Symmetry is preserved by "layer" composition. We show that if its communication graph admits a symmetric electoral system then an algorithm can be transformed into a symmetric equivalent one. Several methods of symmetrization can be described: differed allocation, superposition, cooperation.

Genericity expresses that processes have initially no knowledge of their global environment. In other words, processes may adapt themselves to various environments in various systems. Systems of a generic family of systems are thus made up of standard "chips" (processes) plugged in their communication graph. - We give a syntactial definition of genericity for family of distributed systems. Genericity is preserved by "layer" composition. We consider families where processes differ only by explicit assignment of communication graphs to some variable (explicit knowledge). Such families can be transformed into equivalent generic ones. Genericization is possible thanks to the existence of a (symmetric) generic solution to the Graph Exploration problem: this solution is then used to let processes learn their global environment.

TIME, KNOWLEDGE, BELIEF AND THE LOGICS OF PERSISTENCE

Daniel Lehmann (Hebrew University/Israel)

The logics of time and knowledge were studied in a previous paper (P.O.D.C. 84).

The notion of belief should play an important role in the description of many A.I. systems. Somebody believes formula b if

he is ready to act on the assumption that b is true. A logical system for belief and knowledge (without time) is proposed. Questions concerning the persistence of beliefs are raised.

TERMINATION PROBLEMS IN REWRITING SYSTEMS

Pierre Lescanne (Nancy)

Termination (sometimes called uniform termination) is essential for proving total correctness of rewriting programs but also for deriving confluence or Church-Rosser property from local confluence. In this talk we present works done at Nancy with Alhem Bencheriffa and Isabelle Gnaedig on tractable procedures for proving termination of rewriting systems. One is based on a polynomial interpretation and another one on an auxiliary rewriting system. These methods are illustrated by running the Knuth-Bendix completion procedure on the three axioms

(A) $(x + y) + z = x + (y + z)$

(C) $x + y = y + x$

(E) $f(x) + f(y) = f(x + y)$

or only on (A) + (E). They are indeed really sensible to the ordering that is used to prove the termination.

A PARTIAL CORRECTNESS LOGIC FOR PROCEDURES

Kurt Sieber (Saarbrücken)

We extend Hoare's logic by allowing quantifiers and other logical connectives to be used on the level of Hoare formulas. This leads to a logic in which partial correctness properties of procedures (and not only of statements) can be formulated adequately. In particular it is possible to argue about free procedures, i.e. procedures which are not bound by a declaration but only "specified" semantically. This property of our logic (and of the corresponding calculus) is important from both a practical and a theoretical point of view, namely:

- Formal proofs of programs can be written in the style of stepwise refinement.
- Procedures on parameter position can be handled adequately, so that some sophisticated programs can be verified, which are beyond the power of other calculi.

Our approach is presented more precisely in: A partial correctness logic for procedures, Internal Report A 84/13, Fachbereich Informatik, Universität Saarbrücken, 1984

A COMPOSITIONAL MODEL FOR BIDIRECTIONAL CIRCUITS

Glynn Winskel (Edinburgh)

Languages and semantics are presented for static and dynamic circuits. The languages include a parallel composition - connecting circuits at common connections - and restriction - hiding, or insulating, from the environment all connections not in a set of connections. The model explains the behaviour of a compound circuit in terms of the behaviour of its constituents. It captures the bidirectional nature of circuits and works for a variety of voltage and conduction strengths. A static circuit is modelled as the set of static configurations it can settle into. The language for dynamic circuits is a variant of Milner's SCCS. Its monoid of actions consists of the static configurations with a binary composition. The language and its model provide a foundation for the construction of simulators and proof systems for circuits, and the relation of models at different levels of approximation.

ISSUES IN THE SEMANTICS OF VARIOUS CONCURRENT MODELS

Amir Pnueli (Rehovot/Israel)

Different features of computational models for concurrency are considered, with an evaluation of their effect on the resulting temporal semantics. A simple language of nonterminating programs is presented with its associated temporal semantics. The semantics

is simple, continuous, compositional, fair but not fully abstract. The temporal semantics leads to a compositional proof system that uses maximal fixpoint induction to handle recursion.

FAILURE SEMANTICS WITH FAIR ABSTRACTION

J.W. Klop (Amsterdam)
(joint work with J. A. Bergstra, E. R. Olderog)

We investigate different treatments of divergence in some well-known semantics for processes, notably bisimulation semantics and failure semantics. Abstraction from divergence in some process algebra means that a given process expression which entails divergence, such as $x = ix + a$ where i is an internal step, is equal to an expression without divergence (after hiding i). 'Fair' refers to the assumption that not always the internal option i is chosen.

While in bisimulation semantics abstraction from divergence is easy, it is more problematic in failure semantics and some authors even adopt the principle of "catastrophic divergence" which declares every process with an immediate divergence possibility equal to CHAOS. Indeed, adoption of the abstraction principle KFAR (valid in bisimulation semantics) would lead in failure semantics at once to inconsistency, understood here as the identification of processes with different deadlock behaviour. However, there is an intermediate possibility (between KFAR and CHAOS), namely abstraction from 'unstable divergence', where a divergence is unstable if one of its exit options starts with an initial τ-step (as in $x = ix + \tau.a$). Thus we arrive at three main theories whose difference in the treatment of divergence can be concisely described by means of a divergence or delay operator Δ : (i) $\Delta = \tau$ in bisimulation semantics yields fair abstraction from divergence with periodically recurring exit options; (ii) in failure semantics $\Delta\tau = \tau$ leads to abstraction from unstable divergence; (iii) in failure semantics $\Delta\delta = \delta$ leads to catastrophic divergence. Here 'τ' is Milner's silent move and δ is deadlock. All three theories are mutually inconsistent.

MODAL LOGICS FOR APPLICATIVE PROGRAMS

Peter Pepper (München)

We consider the suitability of various modal calculi for reasoning about applicative programs.

With "positional logic" we can talk about properties holding at specific points of a program (thus obtaining an analogue of Hoare-style logic). The resulting formalism has important applications in particular in connection with program transformations.

By combining ideas from "positional" and from "temporal" logic, we arrive at a system that allows us to reason about the operational behaviour of applicative programs. This leads in particular to a formalism for specifying and verifying applicative communicating processes ("stream-processing functions").

Finally one can add the "choice modalities" *possibly* and *necessarily* in order to cope with nondeterminism. By combining this logic with (linear-time) temporal logic we obtain the calculus of "branching-time logic".

The presentation concentrates on the model-theoretic foundations rather than on the calculi themselves and their meta-logical properties.

ALGEBRAIC CHARACTERIZATION OF MODELS FOR λ-CALCULUS

Manfred Broy (Passau)

Churchs λ-calculus is specified by a partial heterogeneous hierarchical abstract type. Then a model of λ-calculus is a partial heterogeneous algebra fulfilling the basic axioms and the hierarchy-constraints. Basically the existence of such an algebra follows from general theorems about models of partial abstract types. Therefore a model-theoretic treatment of λ-calculus is possible without looking at monotonicity or continuity properties nor solving the problem of reflexive domains.

REAL TIME CLOCKS VERSUS VIRTUAL CLOCKS

Krzysztof R. Apt (Paris)

The problem of detection of termination in a distributed environment is one of the classical issues in the area of distributed computing. It has been first posed and solved by Francez in the context of CSP programs and soon after rediscovered in an abstract setting by Dijkstra and Scholten. The solutions presented in the literature assume an existence of a single process which is supposed to detect termination of the system. In some situations this assumption can be unsatisfactory.

The aim of this talk is to systematically develop symmetric solutions to the problem. The initially used global real time clock is eventually replaced by local virtual clocks. A dependence between the degree of clock synchronization and the efficiency of the solutions is indicated. The development of the algorithm shows how the initial assumption of an existence of a global real time clock can simplify the task of designing distributed programs.

BRINGING COLOR INTO THE SEMANTICS OF NONDETERMINISTIC DATAFLOW

J. W. de Bakker (Amsterdam)
(joint work with J. J. CH. Meijer, J.I. Zucker)

Ever since the introduction of Kahn's highly successful model of deterministic dataflow computation attempts have been made at generalizing his ideas to a nondeterministic setting. References include KELLER, BROCK & ACKERMAN, ARNOLD, BOUSSINOT, ABRAMSKY, KOSINSKI, PARK, PRATT, BROY, FAUSTINI, BACK & MANNILA, BERGSTRA & KLOP and STAPLES & NGUYEN. Our paper presents a new approach, based on a denotational model incorporating the notion of *coloring* the data ("tokens") which flow around in a dataflow net, in the sense as described in DENNIS or WATSON & GURD. In an operational framework, different colors are used to distinguish the data caused by different, in particular *nested*, iterations. (There are more reasons for using color in actual implementations which need not

concern us here.) Denotationally, we exploit this idea by using different bottom elements, viz. as many as there are different iteration (or, rather, recursion) constructs in a given net. The presence of different bottom elements induces a corresponding variety of associated denotational notion. That is, color is used as a *parameter* in each of the usual concepts constituting the model. Thus, ordering, complete partially ordered set (cpo), closedness etc. are all defined with respect to a given color. It was a pleasant surprise for us to discover that this approach is not only natural in sofar as it derives from actual operational considerations, but also allows us to overcome the various substantial problems present in the design of a natural model for nondeterministic dataflow.

A second technical tool is the notion of c-prolongation of a function which throws out operationally undesirable elements. The main technical result is the equivalence of simultaneous vs. iterated least fixed points in the setting with colors and c-prolongation.

PARALLEL DECOMPOSITION OF SPECIFICATIONS

E.-R. Olderog (Kiel)

We are aiming at a methodology for the construction of concurrent processes from their specifications. As known from sequential programming, this construction should proceed by a systematic application of transformation rules. But for concurrent programming a new type of transformation is important: the decomposition of specifications into parallel components.

We present our ideas in the framework of TCSP, a simple kernel language for concurrency. Our specifications describe both safety and liveness properties.

A key point in our approach is to allow a free mixture of specifications with the programming notation of TCSP. This enables a smooth formulation of our transformation rules which deal with initial communication, parallel decomposition and hiding. The strategy in applying these transformations is to derive a system

of recursive implications from the initial specification, which finally yields a recursive TCSP program.

In the talk we concentrate on parallel decomposition and demonstrate its power in constructing increasingly complex schedulers for systems of readers and writers. Our approach is described more fully in:

E.-R. Olderog, Specification-oriented programming in TCSP, Bericht Nr. 8411, Institut für Informatik u. Prakt. Math., Univ. Kiel, 1984

SAFETY AND LIVENESS PROPERTIES IN PTL: CHARACTERIZATION AND DECIDABILITY

W. Thomas (Aachen)

In [1], formulas of propositional temporal logic (PTL) including operators for the past are considered, and such a formula is said to represent a safety property if it is equivalent to some formula $\Box q$ where q only speaks about the past. Liveness properties are defined similarly in terms of formulas $\Diamond q$, $\Box\Diamond q$, $\Diamond\Box q$. We characterize these properties of PTL-formulas, e.g. by counter-free ω-automata with appropriate acceptance conditions (as introduced by Landweber). Combining this with a decidability result of Perrin which implies that PTL-definability is decidable for ω-regular sequence sets, the above safety and liveness properties are shown to be decidable for PTL-formulas.

[1] O. Lichtenstein, A. Pnueli, L. Zuck, The glory of the past, Logics of Programs Conf. 1985

COMPLEXITY OF PROGRAM FLOW-ANALYSIS FOR STRICTNESS: APPLICATION OF A FUNDAMENTAL THEOREM OF DENOTATIONAL SEMANTICS

Albert R. Meyer (Massachusetts Institute of Technology)

Call-by-value strategy specifies that evaluation of the following functional expression would not terminate.

```
LETREC
     f(x,y,z) = IF x≤0 THEN y ELSE f(x-1,z,y) FI
  AND
     g(z) = g(z)+1
IN
  f(2*2,1,g(0))
END
```

The source of the trouble is the divergent argument g(0). In contrast, call-by-need strategy postpones evaluation of g(0) until it is needed in evaluating the body of f -- which it isn't -- and ultimately terminates with the value 1. A function is OPERATIONALLY STRICT in its kth argument if its CALL-BY-NEED application to some arguments fails to terminate whenever evaluation of the actual kth argument fails to terminate. It is OK to evaluate operationally strict arguments at "apply time" according to call-by-value strategy, even when call-by-need semantics is specified. The f above is NOT operationally strict in its third or second arguments, but is in its first.

Call-by-need yields a mathematically more attractive semantics, but call-by-value is generally more efficient. This motivates the question of analyzing declarations to determine which arguments are strict. We discuss the possibility of carrying out an abstract "strictness flow-analysis" of functional programs, pointing out undecidability and complexity results. The investigation provides a case study of how denotational semantics yields an algorithmic solution to an operationally specified program optimization problem.

In the finitely typed case without any interpreted functions (including conditional), the problem is decidable but of iterated exponential complexity. Strictness analysis for first-order declarations (like f above) turns out to be complete in deterministic exponential time.