Design, Analysis and Reasoning about Tools: Abstracts from the Second Workshop

Flemming Nielson
(editor)

September 1992

1 Introduction

The second DART workshop took place on Thursday August 20'th and Friday August 21'st at Limfjordshotellet in Aalborg. The primary aim of the workshop was to increase the awareness of DART participants for each other's work, and to stimulate collaboration between the different groups. In addition to this, the Thursday programme was planned so as to be of interest also to computer scientists not participating in DART; hence a number of survey talks about recent research were presented by the contact persons for each area.

A brief overview of DART will be given in the remainder of this Introduction; for a more detailed description please consult Section 3 of [95]. The remaining sections list the abstracts of the talks given (Section 2), report on the current state of activities (Sections 3 and 4) and give an overview of all research publications from DART participants (Section 5).

The DART project concerns the "Design, Analysis and Reasoning about Tools". It is funded by the Danish Research Councils until 1994 and comprises researchers at Aarhus University, University of Copenhagen and Aalborg University Centre. The research has a strong semantic basis and is grouped under six main headings:

- Semantics as a Descriptive Tool
- Semantics as an Analytical Tool
- Semantics of Concurrency
- Semantics Based Deduction
- Semantics Based Program Manipulation
- Operational Semantics, Types and Language Implementation

This workshop attracted som 28 attendees and a total of 20 (mainly half-hour) talks were given. It was organized mainly by Peter D. Mosses and Kim G. Larsen - and Susanne Brøndberg did a great job in compiling this booklet. The next workshop is foreseen in the Fall of 1993.

2 Abstracts of Talks given

On Thursday, August 20'th, a number of survey presentations were given. These were aimed at a broad audience, beyond merely the participants of the DART project, and presented recent and not-so-recent research so as to give a good overview of the activities. On Friday, August 21'st, a number of more detailed research presentations were given. These were primarily aimed at participants of the DART-project.

2.1 Survey Presentations

Semantics as a Descriptive Tool

Peter D. Mosses
Aarhus University

Action semantics, a framework developed mainly by Peter Mosses at DAIMI, is intermediate between denotational and operational semantics. It uses *actions* to represent the semantics of programming constructs. These actions

are specified in a specially-designed *action notation*, which has a straightforward operational interpretation. The combinators of action notation satisfy various algebraic laws, allowing elementary reasoning about action notation.

Action semantics has significant pragmatic advantages over other frameworks. E.g., action semantic descriptions (ASDs) scale up smoothly to realistic programming languages, and extensions to a described language only require extensions of its previous description—not reformulation.

A book providing the first comprehensive exposition of action semantics has recently been published. It includes the ASD of a substantial sublanguage of Ada. A showcase example of action semantics (an ASD of Standard Pascal) is currently being completed, in collaboration with David Watt (Glasgow).

Action semantics is an attractive basis for semantics-directed compiler generation. In his Ph.D. thesis Jens Palsberg (DAIMI) gives a translation from a subset of action notation to a RISC code—and proves it correct. The system implementing this, Cantor, provides a correct compiler generator for simple ASDs, although the code produced by the compilers is very inefficient. An M.Sc. student at DAIMI is now going to optimize the code produced by Cantor. Jens Palsberg is collaborating with Anders Bondorf (DIKU) on the application of the Similix system to generate compilers from programming languages into Scheme from ASDs, using an interpreter for action notation. A guest Ph.D. student at DAIMI is investigating compiler generation for concurrent languages based on their ASDs.

A system for action semantics, SAS, based on Mathematics, is being developed at DAIMI. It is intended to provide a user-friendly way of constructing, checking, and changing ASDs. The MSc. student who has implemented the prototype SAS is also going to investigate how to implement the operational semantics of action notation directly in Mathematica. Another M.Sc. student at DAIMI is extending the work of Even and Schmidt (Kansas) on type inference to the full action notation. Finally, Valentin Antimirov, an SNF post-dot. at DAIMI, is investigating the implementation of unified algebraic specifications of abstract data types; this work is expected to contribute to the type checking and evaluation of user-specified data in SAS.

The main defect of the action semantics framework is currently the weakness of the theory of action notation. Collaboration on developing action theory (or on any other aspect of action semantics) would be most welcome.

Semantics as an Analytical Tool

Hanne Riis Nielson
Aarhus University

The first part of the talk gives a brief summary of the results obtained within the main areas of program analysis, correct code generation and fixed point computation. The second part of the talk gives a more detailed survey of the results in the latter area.

In the context of abstract interpretation we study the number of times a functional needs to be unfolded in order to determine the least fixed bound. In the monotone framework where all functions are known to be monotone we get an exponential upper bound. If we restrict the functions to be strict and additive we get a quadratic upper bound. These bounds are tight in a certain sense.

By specializing the form of the functionals we show how the classical notions of fastness and k-boundedness carry over to the framework of abstract interpretation. This gives us an alternative way of bounding the number of unfoldings needed to compute the least fixed points. For iterative forms (often obtained when analysing tail recursive programs) and primitive recursive forms (often obtained for programs with a single recursive call) we have simple methods for bounding the number of unfoldings.

Finally, we suggest an algorithm for computing fixed points based on iterative squaring. In the monotone framework it has time complexity $O(log\ k \cdot n)$ when the functional is k-bounded and the domain of the functions has n elements. When the functions are strict and additive the time complexity of the algorithm can be reduced to $O(log\ k \cdot (log\ n)^3)$.

This work has been reported in [92] and [98].

Semantics of Concurrency

Kim G. Larsen
Aalborg University Centre

This talk summarises work within the Semantics of Concurrency section of DART.

The main activity in this area is concerned with the development of a theoretical basis, supporting methodologies and automatic tools for designing and reasoning about concurrent systems.

A substantial amount of work has been carried out for classical process algebras such as CCS, CSP and ACP using the modal μ-calculus as specification languages. In particular, so–called *local* techniques for checking properties of processes have been developed by members of this group, and the problem of compositionality has been studied to great length.

Current research also includes extensions of classical process algebras to deal with more quantitative aspects of processes such as: real–time, probabilistic non–determinism, value–passing, higher–order processes, processes with priorities. Particular resistant to standard techniques is the real–time and value–passing calculi, in particular with respect to axiomatization and decision procedures. New (often symbolic) techniques have been developed and are intended implemented.

As for application of theories and tools, a number of experiments have been carried out using the Aalborg TAV–system. Also intensive experimentation with Esterel and the associated programming environment has been carried out at DAIMI.

Semantics Based Deduction

Glynn Winskel
Aarhus University

This talk summarises work within the Semantics-based-deduction section of the DART project.

There are currently three projects in applying the proof assistant HOL:

- a metalanguage for domain theory in HOL (part of a PhD project)

- a basic theory of computability in HOL (student project)

- lattice theory in HOL (student project)

The work on domain theory and computability borrows heavily from the book "Introduction to formal semantics" by Winskel to be published by MIT

Press, The technique for proving adequacy of a denotational semantics with respect to an operational semantics, via information systems, has also be demonstrated to work for polymorphism.

In addition, there has been work on repairing a defect in Abramsky's "Computational interpretation of linear logic." This work is an essential preliminary to applications of linear logic, for example to a "resource-conscious" domain theory. This work is expected to tie-up with models treated in the handbook chapter "Models for concurrency" by Winskel and Nielsen, which provides helpful leads to describing and reasoning about concurrent computation.

Semantics Based Program Manipulation

Neil D. Jones
University of Copenhagen

Area SBPM is rather broad, covering semantically based work in which programs are interpreted as data objects. Substantial progress has been made in the following research areas, both in theory and in practice, and they have many interconnections despite their seeming diversity. First, an overview.

1. *Partial evaluation* has been applied to larger and more realistic problems and programming languages, including a large-scale lazy functional language, and a new partial evaluator for the widely used programming language C.

2. A number of new results have been obtained in *abstract interpretation*, both under its own heading below and as a tool in partial evaluation and other program optimizations.

3. Much more *efficiet algorithms* have been devised for abstract interpretation, partial evaluation, and type inference. Proven to run in near-linear time, they have also been seen to work extremely well in practice.

4. A better understanding of *complexity issues* in our research area has been achieved. Practical and theoretical progress towards two longstanding problems in partial evaluation has been made — to estimate

how much speedup will be achieved, and to develop analyses sufficient to guarantee that specialization will always terminate. In addition new lower run time bounds have been obtained for practically interesting problems.

5. The relationships among and relative expressivity of type inference, abstract interpretation, and abstract interpretation are becoming better understood.

Abstract interpretation

Some work emphasizes analyses of new program properties useful in efficiently implementing programming languages, and other work concerns implementation, proof that the analyses are correct, and studies of the relations with abstract interpretation and types.

Globalization, the problem of discovering which data can be statically allocated rather than more expensively on a stack or heap, was studied in work done before DART was begun. The earlier work did not, however, satisfactorily account for globalizable data captured in higher order functional values, and under DART the earlier results were strengthened considerably by Gomard and Sestoft in [36].

Evaluation order in lazy languages is a rather tricky subject and has been studied by researchers several places in the world. Stronger results were reported by Gomard and Sestoft in [35] and an improved version came out in [37]. The globalization and evaluation order results were based on results obtained in their two Ph.D. theses [32], [117].

Strictness analysis of attribute grammars was done by Rosendahl in [115], and his and Mycroft's paper [91] generalizes earlier work on "minimal function graphs". This work is being extended to cover lazy programming languages, jointly with Mycroft.

Partial evaluation

This is currently the largest area of activity. A breakthrough was a partial evaluator for the widely used programming language C, described in L.O. Andersen's M.Sc. thesis [12]. It received a prize given by the Dansk Datalogisk Selskab as the "best MSc. thesis of the year". Papers [13] and [10] describe parts of the project.

Our currently most recognized partial evaluator is Bondorf's *Similix* for the medium scale untyped functional language Scheme [16], [89]. It has been distributed to around 200 sites, and at least two papers done at other universities were based on experiments using Similix. Experiments are currently taking place to implement Mosses' "action semantics" using Similix — joint work between DIKU and DAIMI.

The system is continually evolving, see for example Bondorf's Lisp and Functional Programming article [17]. Similix has been used to implement lazy functional languages by specializing interpreters for them. Mogensen is currently working on production of RISC code using an interpreter in continuation passing style.

Two experiments producing Scheme code have been undertaken, the first reported by Bondorf in [14]. Jørgensen's M.Sc. thesis [66] dealt with a much larger language and was reported at POPL [67]. It is particularly interesting because of the language size and the generated compiler's efficiency — the target programs it produced were faster than those of Turner's commercially used Miranda compiler.

Noteworthy is that the compiler was generated automatically by a system which knew *nothing* about special compiling tricks for the language being implemented. The compiler generator's only input was a definitional interpreter of the source language. Target code efficiency was gained by rewriting the interpreter until good separation of compile- and run-time actions was obtained. Experience shows performing such equivalence-preserving transformations on a program that already works to be rather easier than conventional compiler writing.

Two of DART's themes: partial evaluation and the "self-optimizing" phenomenon seen in lazy evaluation of functional languages were discovered by Holst and Gomard to have a deep similarity, reported in [52]. Interestingly, the same idea was independently discovered a few months later by two Japanese researchers.

Bondorf and Mogensen report in [85] a self-applicable partial evaluator for Prolog. The core of the system is finished, but a number of preprocessing analyses remain to be done, The full correctness proof for Lambda-mix by Gomard has now come out in TOPLAS [33].

Work on the challenging problems of estimating speedup factors and ensuring termination in partial evaluation has continued, preliminary speedup results being reported in [11]. A chapter of a book being written on partial evaluation by Jones, Gomard and Sestoft concerns binding time analyses strong enough to guarantee termination, and Bondorf plans to add such an analysis to Similix.

Henglein developed a nearly linear time binding time analysis algorithm using constraint systems and "union-find" algorithms [47], with a smaller constant factor than observed in many theoretically good algorithms with low asymptotic running times. He is currently working on its polymorphic extension. Bondorf and Jørgensen have adapted his ideas to gain dramatic speedups of several Similix analyses. Recent work by Palsberg and Schwartzbach compares Bondorf's and Gomard's approaches to binding time analysis, proving the former to be strictly more powerful than the latter — another DAIMI-DIKU link.

Lambda calculus

This mathematically based small programming language has received considerable attention, including Klaus Grue's doctoral work which belongs under DART activity 6.7 [39].

Following Barendregt's publication of a mathematically simple self-interpreter for the pure lambda calculus without constants, Mogensen developed a *much* simpler and more efficient alternative, and a "self-reducer" as well, reported in [83]. Extension of these ideas led to an exceptionally small partial evaluator, reported in [84].

This work has already attracted some attention. Mitch Wand (Boston) has developed a new proof technique and formally proven Mogensen's partial evaluator to be correct; and Corrado Böhm (Rome) has a paper on an improved self-interpreter in the forthcoming conference *Computer Science Logic*.

Hannan (a postdoc supported by DART) showed that lambda Prolog can faithfully describe various reduction strategies in [44]. Current work includes studies and experiments on "parallel optimal" reduction strategies, being done by Rose and Jørgensen at DIKU, with related work by Birk Hansen and Hvid Sørensen at DAIMI.

Types

Jones reported on a notation for the types of language processors in [61] and its relation to compiler generation by partial evaluation. He visited Hagiya in Kyoto for further work, but much still remains to be done.

Henglein has worked on polymorphic extensions of binding time analysis, boxing/unboxing analysis, and strictness analysis. In [48] he shows the undecidability of type inference in Mycroft's scheme for polymorphic recursive types by reduction from the semiunification problem.

Dynamic typing with a minumum of tagging and untagging operations is studied both theoretically and practically by Henglein in [49] and [50]. This very promising work is the basis for a Scheme to ML translator worked on by Kees van Schaik, visiting from the University of Utrecht.

Complexity

An "oblivious" algorithm is one whose control flow depends only on the inputs known at partial evaluation time. Their role in relation to the predictability and efficiency of program specialization is just now becoming understood. Recent observations appear in a chapter by Jones of the partial evaluation book.

Andersen and Gomard present an automatic method to estimate speedup factors obtained by partial evaluation in [11]. Work by Henglein was described before on efficient algorithms in partial evaluation [47]. Reformulation of earlier analyses in terms of constraint systems has already led to considerable improvements in the speed of Similix.

Henrik Andersen from DAIMI is at DIKU for the Fall semester, working on efficient algorithms for model checking.

A lower bound complexity result recently obtained by Jones is that increasing computation time by a constant factor on a natural computing model properly increases computing power. The result is interesting for two reasons: it contradicts a known result for Turing machines and can be interpreted as showing that they are an unnatural computing model; and it shows that the constant speedups typically obtained in partial evaluation are nontrivial in a mathematical sense. The proof is interesting too, as the novel technique involved is a highly efficient self-interpreter for a simple imperative language.

Graph rewriting

A M.Sc. thesis by Rose [114] led to two papers [112], [113] concerning the use of graph rewriting as a semantic formalism. The new DIKU postdoc, David Sands, is working on the Gamma model of parallel computation, based on conditional multi-set rewriting (with an elegant chemical reaction metaphor).

Further work is planned, e.g. experiments with a "hashing apply" for online memoisation, and with Lamping's optimal graph reductions as mentioned in the lambda calculus discussion.

Diverse topics

Hannan describes derivation of machine architectures from interpreters in [43]. He and Pfenning have used the computer to prove correct a compiler from the lambda calculus to the Categorical Abstract Machine in [46], using Pfenning's logic programming language which is based on the Edinburgh Logical Framework. It is the most complex mechanically proven compiler of which this writer is aware.

Operational Semantics, Types and Language Implementation

Mads Tofte
University of Copenhagen

In this presentation we give an overview of the activities that are going on under area OST, namely the semantics of higher-order functors in ML, type inference and storage allocation, and the ML Kit.

Higher-older Functors

We have defined the static semantics of higher-order functors in a skeletal language and proved a central theorem which says that so-called principal signatures exist. An algorithm which finds principal signatures has been developed; it has been implemented in the ML Kit by Lars Birkedal. There has been close collaboration with David MacQueen of Bell Labs, one of the authors of Standard ML of New Jersey. The next version of New Jersey ML will support higher-order functors.

Type Inference and Storage Allocation

Within the past six months, we have had an exciting breakthrough in this area. We have found that it is indeed possible to use a refined notion of type inference, we call it ***region inference***, to partition the values a program produces into regions, which can be allocated and de-allocated in a stack-like manner. The exciting thing is that the scheme works for languages with higher-order functions, despite the fact that it is widely believed that stack allocation is incompatible with higher-order functions. So far, we have concentrated on developing a mathematically robust theory of region inference; we plan to test the ideas in practice in future. This work has been done in collaboration with Jean-Pierre Talpin, Ecole des Mines, Paris.

The ML Kit

During the past year the Kit has been developed further to an extent that it now implements virtually all of Standard ML. Also, we have written about half the planned documentation. A student, Lars Birkedal, has been employed as a half-time programmer, funded by DART; his job is to fill out the remaining holes in the Kit code and to write further documentation. This work has been done in collaboration with Nick Rothwell of Laboratory for Foundations of Computer Sciences University of Edinburgh.

Type Systems and Constraints

Michael Schwartzbach
Aarhus University

The general area of research has been *type systems*, interpreted in a broad sense, using *constraint-based* techniques.

Joint work with Jens Palsberg has focused on applications of *closure analysis* of the λ-calculus. Phrasing both as constraints, we have managed to show that *safety analysis*, based on closure analysis, is strictly more powerful than Milner-style *type inference* in accepting safe λ-terms; this result has been improved to allow even inference with recursive and partial types. We have employed a novel proof technique which is based on solvability-preserving maps on constraint systems. A further application of this has shown that the binding time analysis of Bondorf, based on closure analysis, is strictly more powerful than that of Gomard, based on type inference. We have defined an improvement of closure analysis, which ignores unreachable code,

and shown it to be sound with respect to both a strict and a lazy operational semantics. We have adapted closure analysis to object-oriented languages and shown how it can solve a number of static analysis problems. The use of program transformations to improve such analyses has been developed.

Algorithmic aspects of subtyping has been explored with Dexter Kozen and Jens Palsberg. Using again a constraint-based approach we have solved two open problems concerning type inference with Thatte's partial types: that typability with finite types can be decided in polynomial time, and that recursive types always allow a unique Böhm-minimal type. A further development of this approach has enabled us to decide the Amadio-Cardelli recursive subtyping relation in quadratic time, improving on the previously best-known exponential time.

Recent work with Nils Klarlund has enriched recursive datatypes with a particular kind of constraints. This yields the concept of *graph types*, allowing the systematic definition of non-tree shaped families of values, such as doubly-linked cyclic lists threaded trees, etc. We have developed a decidable monadic logic for reasoning about such types, as well as optimal algorithms for their manipulation at run-time. This has the promise of allowing more conventional data structures to be defined in functional languages.

Map Theory

Klaus Grue
University of Copenhagen

Map theory is a foundation of mathematics based on lambda-calculus instead of logic and sets, and thereby fulfills Church's original aim of introducing lambda-calculus. Map theory can do anything ZFC set theory can do. In particular, all of classical mathematics is contained in map theory. In addition, and contrary to set theory, map theory has unlimited abstraction and contains a computer programming language as a natural subset. Map theory is as simple and homogeneous as ZFC set theory.

The talk gives an intuitive understanding of the notion of a map and gives a short overview of implementation of map theory on a computer.

2.2 Research Presentations

Formal Specifications of CML Programs

Klaus Havelund
Ecole Normale Superieure (ENS), Paris

Purpose of this Work

The purpose of this work is to develop a specification language for (a subset of) the programming language CML.

What is CML?

CML (Concurrent ML) is an extension of the programming language ML with CCS/CSP-like concurrency primitives. The model behind CML is that of concurrently executing expressions that communicate by synchronous message passing on typed channels.

Channels can be created dynamically and they are themselves values. Concurrency is obtained in terms of a so-called **spawn** operator. In principle, the expression **spawn**(e) starts the evaluation of expression e whereafter evaluation of the remaining program immediately continues. The remaining program will then run in parallel with e.

Since functions are values, processes (functions that communicate) are values, thus making CML truly higher order.

CML allows for imperative programming with side-effects via ML reference types. This concept can be ignored since ML references can be theoretically modelled by the concurrency primitives added in CML.

A Specification Language

A specification language for CML must make it possible to state properties about CML programs. Inspiration can be obtained by studying diverse modal logics for concurrency specification. One should for example be able to state a property like: If during the evaluation of this expression a value x is received (input) on the channel i, then later in the expression evaluation, the value $2 * x$ will be sent (output) on the channel o.

Region Inference

Mads Tofte
University of Copenhagen

It is widely believed that higher-order functions are incompatible with stack implementation. In this talk we present *region inference*, a type-inference based technique which allows compile-time inference of where data is to be put in order that allocation at runtime can happen in a stack-like manner, even in the presence of higher-order functions. The technique has been proved sound for a skeletal language, namely the call-by-value λ-calculus with `let` typed according to Milner's polymorphic type discipline. Instead of a heap and garbage collection, our implementation scheme has *regions*, which are allocated and deallocated in a stack-like manner. Regions are inferred using types and effects, and there exists an algorithm which finds principal types, minimal effects and smallest possible regions.

In this talk we explain what region inference is, show the region inference rules and state the soundness theorem. (There will probably not be enough time to present the inference algorithm.)

The C-Lisp Language

Frank Jensen
Aalborg University Centre

C-Lisp is a higher-order functional language with facilities for synchronous communication (message passing), process creation and channel allocation.

We present a calculus for a simple language $\mathcal{L}$ that captures the essential features of C-Lisp. This calculus is the λ-calculus extended with primitives for communication (send/receive a value), for generation of new channels and processes (agents), and for sequencing and recursion.

The semantics of $\mathcal{L}$ is defined through two transition systems: one transition system defines the possible actions of individual agents, while another transition system defines the behavior of a collection of agents.

Finally, we outline how agents of the π-calculus (by Milner et al) can be translated into $\mathcal{L}$ in such a way that the (π) agent and its translation can simulate each other's actions quite closely.

Value-passing

Anna Ingolfsdottir
Aalborg University Centre

The standard theory of CCS, based on testing, is extended to a process algebra which supports transmission of simple values. This will be compared to a similar theory based on the idea of testing due to Hennessy and De Nicola.

Nontriviality of Constant Speedups

Neil D. Jones
University of Copenhagen

It is proven that that multiplying computing time by a sufficiently large constant properly increases computation power for a natural computation model. Specifically, given a $f(n)$ time-constructable function, there is a constant c and a set X such that X can be recognized in time $c \cdot f(n)$ but cannot be recognized in time $f(n)$.

The result strengthens Fürer's "tight time hierarchy", but is of course in direct contradiction to the constant speedup enjoyed by Turing machines. It makes clear what the price for having arbitrarily large tape alphabets is.

The computation model consists of syntactically restricted imperative programs manipulating Lisp-like list values. The result appears to carry over to other models allowing pointer reference in constant time, e.g. Tarjan's pointer machines with a uniform bound on record size.

Partial evaluation nearly always yields constant speedups, and it has been somewhat disconcerting to recall that such speedups give no increase at all in problem solving ability for Turing machine computation models. The result is of relevance to DART since it shows that constants do indeed matter, for models that are closer to computing practice.

The technique uses

1. A self-interpreter whose interpretation overhead is a constant multiplicative factor, *independent* of the program being interpreted. This is the key point to the proof.

2. Extending this to a self-timed version, using the fact that one can count the binary representation of n down to 0 in time $O(n)$.

3. The usual resource-bounded diagonalization argument.

So the constant speedups we get by partial evaluation are not trivial (in terms of complexity theory) after all. And list values and efficient interpreters are a Good Thing.

Binding Time Analysis: Abstract Interpretation versus Type Inference

Jens Palsberg
Aarhus University

Binding time analysis is important in partial evaluators. Its task is to determine which parts of a program can be evaluated if some of the expected input is known. Two approaches to do this are abstract interpretation and type inference. We compare two specific such analyses to see which one determines most program parts to be eliminable. The first is the abstract interpretation approach of Bondorf, and the second is the type inference approach of Gomard. Both apply to the untyped lambda calculus. We prove that Bondorf's analysis is better than Gomard's.

We present both binding time analyses via constraint systems. The motivation for this is to abstract away the algorithmic aspects of computeing binding time information. Rather the constraint systems yield two soundness predicates on all possible outputs of a binding time analysis. The outputs we consider are elements of the 2-level lambda calculus. They have a natural ordering such that "smaller" means "better". The algorithms of Gomard and Bondorf can then be understood as computing the least 2-level term which is sound relative to their respective predicates. Our comparison proceeds

by showing that if Gomard's predicate is true of a particular 2-level term, then so is Bondorf's. This implies our result: for any pure term, Bondorf's analysis produces a smaller 2-level term than does Gomard's analysis

This work is joint with Michael Schwartzbach.

Generation of Proof Obligations for Type Consistency

Bo Stig Hansen
DTH, Lyngby

This work concerns a type inference system for a simple applicative language. The system can be used for inferring types of expressions and, in addition, the necessary proof obligations for ruling out "dynamic" type errors such as taking the head of an empty list.

The system has been proved sound and complete with respect to a denotational semantics of the language. In this semantics, the head of an empty list is the special element WRONG, denoting type error, and not BOTTOM which is used only for modelling non-termination.

The work is joint with Ph.D. student Flemming Damm.

Reasoning about Infinite State Systems in the Modal mu-Calculus

Henrik R. Andersen
University of Copenhagen

We describe a method for performing model checking on infinite state systems in the modal μ-calculus. The method can assist in proving that subsets of states of infinite labelled transition systems satisfy formulae in the modal μ-calculus. Success of using the method in one particular situation will depend on proper *choices* in certain steps of applying the method and on the ability to show properties of infinite sets by properly chosen induction principles like mathematical or structural induction. The undecidability of the overall problem can be viewed as a combination of the impossibility of making these

choices algorithmically and of the impossibility of algorithmically deciding properties on infinite sets.

The method will be sound in the sense that, whenever a model is shown to satisfy an assertion of the logic using the method, this is certainly a valid conclusion, and it will be complete in the sense that, whenever a model satisfies an assertion it is possible to make correct choices, and — provided the infinite reasoning is possible — find a finite number of steps of the method proving this fact.

Recursive Binding as a Primitive

Kristoffer H. Rose
University of Copenhagen

Most programming languages include some kind of recursive binding, *i.e.*, cyclic definitions of data and computations through naming. The binding present in standard (λ-calculus based) models of computation, however, only allows acyclic binding directly, whereas cyclic definitions have to be 'encoded' using fixed point combinators. Consequently, models of recursive binding do not enjoy the same simplicity in presentation as models of acyclic binding (they need closures, stores, or other intermediate structure), making it difficult to represent and reason about 'self-sharing,' *i.e.*, cyclic data structures. In this paper we show how the 'Explicit Substitutions' technique of Abadi, Cardelli, Curien, and Lévy (in the POPL '90 proceedings) can be extended to handle the cyclic ease by applying it to a λ-calculus with an explicit substitution operator.

Constructor Specialization

Torben Mogensen
University of Copenhagen

We investigate the posibility of specializing constructors as well as functions during partial evauation. We present an example that shows that the idea has merit, and discuss implementation of binding time analysis and specialization.

Efficient Analyses for Realistic Off-Line Partial Evaluation

Jesper Jørgensen
University of Copenhagen

Based on Henglein's efficient binding time analysis for the lambda calculus (with constants and "fix"), we develop four efficient analyses for use in the preprocessing phase of Similix, a self-applicable partial evaluator for a higher order subset of Scheme.

A *flow analysis* determines possible value flow between function applications and lambda abstractions and between constructor applications and selector/predicate applications. A *binding time analysis* distinguishes static and dynamic values; the analysis treats both higher order functions and partially static data structures. An *is-used analysis* finds a non-minimal binding time annotation which is "safe" in a certain way: first order values may only become static if their result is "needed" during specialization; this "poor man's generalization" [52] increases termination of specialization. Finally, an *evaluation order dependency analysis* ensures that the order of side-effects is preserved in the residual program. The analyses are performed sequentially in the above mentioned order since they depend on results from the previous analyses.

The input to all four analyses are constraint sets generated from the program being analysed. The constraints are solved efficiently by a union/find-based algorithm in almost-linear time. Whenever possible, the constraint sets are partitioned into subsets which are solved in separate subphases; this simplifies constraint solving.

The framework elegantly allows expressing both forwards and backwards components of analyses. In particular, the is-used analysis is of backwards nature.

The four constraint solving algorithms have been proved correct (soundness, completeness, termination, existence of a best solution).

The analyses have been implemented and integrated in the Similix system. The implementation confirms that the analyses are indeed efficient in practice and that the complexity is almost-linear. For example, preprocessing a 75K

machine produced program that makes heavy use of higher order functions and partially static data structures (the compiler generator of Similix) takes around 15 seconds (Chew Scheme version 3.2 on a SPARC 2).

Minimal Function Graphs for a Lazy Language

Mads Rosendahl
University of Copenhagen

The minimal function graph technique was proposed by Jones and Mycroft as method to define the semantics of a first-order eager functional language. In the semantics, the meaning of a function is described as a set of argument-result pairs where only arguments to function calls which may be reached from a given initial call are included. The semantics may be used for defining program analyses which extract information about the possible arguments to the functions in the program.

In this work we describe a minimal function graph for a language with a call-by-need evaluation strategy for arguments to function calls. The extension requires that one not only collects arguments to function calls but also the needed arguments to functions. The semantics is presented in a framework which facilitates its use for abstract interpretation.

Possible applications of the semantics in program analysis are discussed. They include neededness analysis, automatic complexity analysis, and constant propagation.

The work is done in collaboration with Alan Mycroft.

3 Current Status of DART (Summer 1992)

This section presents the results obtained so far and the work that still lies ahead. It is grouped on reports from each of the "work areas" that were mentioned in the Introduction. For more detailed descriptions of the aims of each area please consult Section 3 of [95] "Abstracts from the first Workshop".

3.1 SDT (DART 6.1): Semantics as a Descriptive Tool (by P.D. Mosses)

Peter Mosses presented action semantics at the Marktoberdorf Summer School [86]. He has recently published a book [88] providing the first comprehensive exposition of action semantics. A showcase example of action semantics is currently being completed, in collaboration with David Watt (Glasgow).

Padmanabhan Krishnan is now a lecturer at Christ Church, New Zealand. During the last months (supported by DART) of his post-dot. at DAIMI he completed his investigations of the action semantics of real-time and distributed systems [70, 71, 72, 74].

In his Ph.D. thesis [103] Jens Palsberg has given a translation from a subset of action notation to a RISC code, and proved it correct. The system implementing this, Cantor, provides a correct compiler generator for simple action semantic descriptions. An M.Sc. student is going to optimize the code produced by Cantor. Jens Palsberg is collaborating with Anders Bondorf on the application of the Similix system to action semantics based compiler generation. Martín Musicante, a guest Ph.D. student, is investigating compiler generation for concurrent languages using action semantics.

A prototype system for action semantics, SAS, has been implemented. The MSc. student who programmed it in Mathematics is now going to investigate how to implement the operational semantics of action notation directly. Another M.Sc. student is extending the work of Even and Schmidt (Kansas) on type inference to the full action notation. Finally, Valentin Antimirov, an SNF post-dot., is investigating the implementation of unified algebraic specifications of abstract data types; this work is expected to contribute to the type checking and evaluation of userspecified data in SAS.

Some advances concerning the specification of sorts of actions have recently been made. However, the theory of action notation needs further development to allow direct reasoning about nontrivial action equivalence.

3.2 SAT (DART 6.2): Semantics as an Analytical Tool (by H.R. Nielson)

The main activities have been within the following areas:

Specification and correctness of program analyses:

The power of abstract interpretation relies on the constructions used for the abstract domains. It has been shown that using a *tensor product* one can avoid Wadler's case construct without loosing precision. Further precision can be obtained using *open sets* when analysing lists. A MSc-thesis (by Jens Mikkelsen) shows how *homomorphisms* can be used to construct abstract domains which give even more precision although more work is needed to study their computational tractability.

Computation of fixed points:

The efficient implementation of abstract interpretation relies on the cost of computing fixed points. When the abstract domains are finite (as often is the case) the cost can be bounded by the height of the domain of the functional; this has been studied in detail for three frameworks of abstract interpretations It has also been shown that one may obtain even lower bounds when the functionals have special forms.

Correctness of simple and optimizing code generation schemes:

Although the specification of code generation for lazy functional languages may seem obvious, it is rather complicated to formulate and to prove its correctness. The concept of *layered predicates* has been introduced to reduce the complexity of the problem and allows the proof to be conducted in a few stages. A MSc-thesis (by Torben Lange) shows that the technique easily adapts to optimizing code generations using abstract interpretation.

Also there has been work on logical specifications of *binding time analysis* (by Kirsten Solberg, PhD-student from Odense), *safety analysis* (by Jens Palsberg, research assistant employed by DART from 1/8-1991 to 31/12-1991), *correctness of code generation for Occam* (by Anders Gammelgaard, PhD-student, ProCoS) and *program transformations* (by Karin Glindtvad, former research assistant, ProCoS, and Torben Amtoft, PhD-student). Finally, an experimental system in Miranda supporting a number of the above

mentioned techniques is under construction (by Anders Pilegaard, former programmer, and Torben Lange, research assistant employed by DART from 1/7 1992).

In addition to three MSc-theses and one PhD-thesis the above work is documented in a couple of conference papers, (forthcoming) journal papers and the monograph, *Two-Level Functional Languages* (by Flemming Nielson and Hanne Riis Nielson, Cambridge University Press, 1992). Also an introductory book to the general area of semantics has appeared: *Semantics with Applications, A Formal Introduction* (by Hanne Riis Nielson, Flemming Nielson, Wiley, 1992).

The *future work* will most likely be centered around program analysis and their specification (by abstract interpretation, logical systems, projections, partial equivalence relations, etc). Both theoretical and practical aspects will be studied but the emphasis will be on the former. So far most of the work has been for functional languages but an opening towards other kinds of languages (e.g. with concurrency primitives) is expected. There is already a number of activities supporting this development:

- PhD-seminars on *Approaches to Program Analysis* (Spring 1992) and on *Functional Languages with Higher-Order Processes* (Fall 1992).
- DART task meetings on *Type Inference* (May 1992) and on *Linear Logic* (planned for Fall 1992). Also it is expected that the work on fixed point computation and layered predicates will be continued.

3.3 SOC (DART 6.3): Semantics of Concurrency (by K.G. Larsen)

The main activity in this area is concerned with the development of a theoretical basis together with supporting methodologies and automatic tools for designing provably correct distributed/concurrent systems. In particular, theories allowing modular design and compositional verification are sought; i.e. it should be possible to relate properties of a complex system to properties of its components.

Using the modal μ-calculus as a specification language for parallel systems,

we have described methods for decomposing specifications of a combined process into sufficient and necessary properties of its components [80, 8, 127]. This theoretical basis for compositional verification has been established for classical process algebras such as CCS, CSP and ACP. Furthermore, in [76, 127] the expressive power required of a specification formalism in order that it supports decomposition of properties has been characterized.

Related to these decomposition methods a number of *local techniques* for checking various properties of parallel systems have been developed. In contrast to traditional *global* techniques, the local techniques are designed so that precomputation of the global state–space (and hence state–explosion) may be avoided. In [77] a general description of the local technique underlying all the tools of the TAV system is given in terms of Boolean Equation Systems. In [7, 5, 127] local algorithms for model checking with respect to the modal μ–calculus are given. In particular, the efficiency of these local techniques is comparable to the best known global algorithms and currently work is made towards their practical implementation.

Using classical process algebras and classical modal and temporal logics it is possible to specify concisely the desired observable behaviour of a parallel system. However, such specifications will focus only on *qualitative* aspects of a system while leaving unspecified *quantitative* aspects which often are vital for several practical applications. Such quantitative aspects include:

- Real-time (dense or discrete)
- Probabilistic non-determinism
- Passing of values in communication
- Priorities among processes and/or actions

Within DART substantial research has been carried out during the last year in order to extend the classical results in the above directions. We give a more detailed account of the results obtained below.

For (dense) real-time system we have applied the basic model of timed graphs by Alur, Courcoubetis and Dill and the timed version of CCS (TCCS) by Wang. We have obtained decidability results for a regular part of TCCS [51] and more recently decidability for static networks of regular timed processes

has been obtained [22]. This lays the theoretical foundation for tool–building, which we intend to implement during the next year (in collaboration with Karlis Cerans). Real–time (dense) calculi are fairly resistant to standard techniques; in particular, we have shown that no expansion theorem will hold, and hence that parallel composition can not be reduced to non–determinism [31]. As for work in progress, we are currently investigating a notion of time–abstracting bisimulation on TCCS and to what extent this (rather abstract) equivalence may capture timing properties [128].

With respect to probabilistic behaviour we have studied various specification formalisms. In [65] a transition based specification formalism for probabilistic behaviour is introduced together with a characterization of the refinement ordering between specifications, The resulting theory can be seen as an extension of the theory for modal transition systems. In [79] a probabilistic calculi and a probabilistic modal logic is introduced. As main results it is shown how to decompose properties expressed in the logic with respect to operators of the calculus. Also a complete axiomatization of (validity of) the logic is given. Finally, in [128] a test theory for probabilistic processes is developed. In particular, a process will pass a test with a certain probability refining the notions of a process *must* or *may* pass a test in the classical test theory of Hennessy.

Substantial work has been made on the semantic basis for communicating processes with value–passing [58], and currently a group at DAIMI is making progress on the decidability problems associated with this class of processes. Similar to (dense) real–time systems, the infinite nature of data immediately leads to infinite–state systems and new techniques are therefore needed. We believe that the symbolic techniques used for real–timed processes and processes with values are related and we intend to investigate this relationship during the next year.

As argued above, infinite–state systems are unavoidable when introducing concepts such as time and values. During the last year, several fundamental results on the decidability of various equivalence for a class of (potentially) infinite–state process have been obtained [54]. In [56] bisimulation has been shown decidable for the class of normed BPA processes (a BPA process is built from a set of basic actions using summation, sequential composition and recursion). This result has recently been extended to the full class of BPA processes [23] (an open problem for nearly five years). In [55] decidability of

branching bisimulation for normed BPA is established. In contrast it is shown in [38] that all other standard equivalences are undecidable for BPA. As for model checking infinite state–systems techniques (rather than algorithms) are under development [5]. These techniques can be seen as combining modal reasoning with familiar induction priciples.

Also in the direction of processes with priorities work has been done both with respect to the development of a suitable semantic basis [21] and in developing tools supporting automatic verification of such systems [59].

In the direction of *application* intensive experimentation with Esterel and the associated programming environment has been carried out at DAIMI. Also, a wide range of experiments has also been carried out using the TAV system [20, 18, 19]; in particular, an interface to AUTO and its graphical interphase (AUTOGRAPH) has been developed, making it possible to apply TAV to the verification of (finite–state) Esterel programs. We hope the class of Esterel programs amenable to automatic verification can be enlarged using future decidability results for value–passing processes.

3.4 SBD (DART 6.4): Semantics Based Deduction (by G. Winskel)

A major aim is to strengthen and support expertise in automated proof, based in theoretical work in logic, domain theory and concurrency.

There are currently three projects in applying the proof assistant HOL.

- a metalanguage for domain theory in HOL (Sten Agerholm, Part of a PhD project)
- a basic theory of computability in HOL (Ole Hougaard, student project)
- lattice theory in HOL (Esben Dalsgaard, student project)

The work on domain theory and computability borrows heavily from the book "Introduction to formal semantics" by Winskel to be published by MIT Press [123]. The technique it presents for proving adequacy of a denotational semantics with respect to an operational semantics, via information systems, has also been demonstrated to work for polymorphism [119].

Urban Engberg, as part of his PhD work, has been mechanising verification in Lamport's TLA (Temporal Logic of Actions). He is presently constructing an interactive theorem proving system, improving on his earlier work described in [26].

In additions there has been work of Torben Braüner, advised by Douglas Gurr and Glynn Winskel, on repairing a defect in Abramsky's term language for proofs in "A computational interpretation of linear logic". This work is an essential preliminary to applications of linear logic, for example to a "resource-conscious" domain theory. It is expected to tie-up with models treated in the handbook chapter "Models for concurrency" by Winskel and Nielsen [126], which provides helpful leads to describing and reasoning about concurrent computation.

In March Aarhus hosted a very successful workshop in Categorical Logic in Computer Science [122].

This summer saw the visits of Tom Melham and Juanito Camilleri, both helpful because of their expertise in HOL; Juanito also spent two days experimenting with the "Priority Workbench" implemented by Claus Torp Jensen, in the analysis of "toy" operating systems. Kim Skak Larsen has begun work in the implementation of Henrik Reif Andersens' model-checking algorithms. It is hoped that DART will fund a continuation of Kim Skak Larsen's work, presently supported by funds under Mogens Nielsen and Glynn Winskel. AnnGrete Tan will begin as Introduktionsstipendiat in November—her research is likely to be in theorem proving and constructive logic. Sergei Soloviev, associate professor in Mathematics at the University of St. Petersburg, will come to Aarhus in January '93 employed by the ESPRIT Basic Research Action CLICSII. Douglas Gurr has left Aarhus to take a position in London.

3.5 SBPM (DART 6.5): Semantics Based Program Manipulation (by N.D. Jones)

Area SBPM has been in a very dynamic state since DART began in March 1991, and noteworthy accomplishments have been made towards the goals outlined in the original proposal. This part of DART is still very much on track with numerous results already obtained and described in the "survey"

part of this report, and with many interesting tasks remaining to be done and new leads to be followed. Some are described later in this section.

Staff changes

Mads Tofte was appointed as lektor (associate professor), and Fritz Henglein was appointed as adjunkt (assistant professor).

John Hannan (University of Pennsylvania) was employed as postdoctoral guest for a half year, and David Sands (Imperial College, London) will replace him starting in November. A guest, Kees van Schaik, has worked on dynamic typing with Fritz Henglein, with support from Holland. Much useful work has been done by DART-employed programmers Lars Ole Andersen and Peter Holst Andersen.

Education

Four M.Sc. theses were written (Lars Ole Andersen, Jesper Jørgensen, Kristoffer Rose, and Bjarne Steensgaard/Morten Marquard). Lars Ole Andersen's received an award from the Dansk Datalogisk Selskab as the best Danish M.S. thesis of the year in Computer Science.

Three Ph.D. theses have been defended in 1991: Hans Dybkjær in May, Peter Sestoft in November, and Carsten Gomard in December. Three students are currrently working on Ph.D. degrees at DIKU (Lars Ole Andersen, Jesper Jørgensen, and Kristoffer Rose), and three students being advised by DART members and with Danish funding are working in foreign countries on Ph.D. degrees.

External recognition

Papers have been presented at conferences (most with publically available proceedings) including *ACM Principles of Programming Languages, Bordeaux Workshop on Program Analysis, Dagstuhl Workshop on Functional programming, Compiler Technology, and Parallelism, Functional Programming and Computer Architecture, Glasgow Workshop on Functional Programming, Lisp and Functional Programming, Logic in Computer Science, Logic from Computer Science* (not the same!), and *Partial Evaluation and Semantics Based Program Manipulation.*

DART people in area SBPM were on the program committees of the first two, and as well selected papers for *ICALP 92*, and *Programming Language*

Implementation and Logic Programming.

In addition, several journal and book articles have appeared, in the *ACM Transactions on Programming Languages and Systems, Journal of Functional Programming, Structured Programming, Mathematical Structures in Computer Science, Science of Computer Programming*, and *Theoretical Computer Science.*

A special issue of *Science of Computer Programming* was guest edited by Neil Jones, and a book on partial evaluation by him, Carsten Gomard, and Peter Sestoft is nearing completion, to be published by Prentice Hall next year.

Future work

The goals enunciated in the original DART proposal are still valid; rapid progress has been made, but much remains to be done. Partial evaluation has become widely recognized worldwide, and the publication of our book in 1993 should give it even more momentum. Crucial factors for a wider use of partial evaluation include:

- Stand-alone systems for realistic and widely used programming languages. There is still much to do with the *C partid evaluator* to turn it into a generally usable tool, but the possibility of this in principle has now been firmly established. The language ML is without question the most widely used typed functional language in the world, with large user communities in the UK, the US, and other countries. We have recently decided to begin a pilot project on *partial evaluation of ML*, and feel well equipped in both theory and practice on the basis of our members' expertise in areas SBPM and OST.

- More highly automated, reliable, efficient and predictable partial evaluation systems. This requires a better understanding of complexity issues and efficient algorithms. A major task is to predict what the results of program specialization will be before it is done; for this our work in abstract interpretation, speedup analysis, and general algorithm design will be quite useful.

- Wider applications. By far the greatest part of our work until now has centered around programming languages. We are now beginning

to investigate applications of *partial evaluation in scientific computing*, and have preliminary contacts with the Danish Hydraulics Institute and other groups.

- Theoretical issues. Recent applied successes in partial evaluation and related areas have opened up a variety of interesting new theoretical questions with considerable long term practical relevance. One example is how to identify those characteristics of a computational task that makes it particularly well suited to partial evaluation which in turn leads to many new questions to ask in applied algorithm design, analysis, and implementation.

A special problem

Anders Bondorf was employed full time on Esprit BRA (Basic Research Action) project Semantique I for three years, but its continuation Semantique II is as a working group rather than as a project. In consequence Semantique II has a much smaller budget, without funds for full-time researchers during its planned three years.

This situation is in no way a sign of problems with the academic quality of Semantique I (which received excellent evaluations), but rather that this year's Esprit BRA program suffered from bad planning and political problems. The BRA program's final budget was around half of that which was initially projected, with the result that many successful projects were terminated, and many others were changed to working groups with no salary funds.

Anders Bondorf's work is essential to SBPM as he is the head architect of the Similix partial evaluator and has published numerous articles on it. This system has been copied by nearly two hundred researchers many places in the world, and is under continuous development. Its recognition by other places has been discussed in the "Survey" section, and is also evident from the bibliographical items by himself, Jørgensen, Mossin, and Palsberg at Aarhus, all centered around Similix.

We argued vigorously for some salary to aid transition from Semantique I to Semantique II. The result is that Semantique II will pay 6 months salary, specifically targeted for Anders Bondorf. I have been told that we were the *only* working group to receive any salary money at all. In addition Bondorf

will receive 6 months "vikar" salary from DIKU, so he will not need to leave our research group in the immediate future, but the situation after next Spring is still unsettled.

In conclusion, Anders Bondorf's work is quite important for the whole DART project, and he has been one of the group's most productive members with respect to constructing pra ti ally useful (and used) programs and to transferring theoretical ideas into computational practice. However, continuation of this work in 1993 and later will require additional funds.

3.6 OST (DART 6.6): Operational Semantics, Types and Language Implementation (by M. Tofte)

The project proposal defined three areas of activities: the semantics of higher-order functors in ML, type inference and storage allocation, and the ML Kit. Progress in these areas has been as follows:

Higher-order Functors:

We have defined the static semantics of higher-order functors in a skeletal language and proved a central theorem which says that so-called principal signatures exist. An algorithm which finds principal signatures has been developed; it has been implemented in the ML Kit by Lars Birkedal. The theoretical results were presented at POPL '92 and a full journal version of this paper has been submitted for publication. There has been close collaboration with David MacQueen of Bell Labs, New Jersey (two visits by him to Denmark, one by Mads Tofte to Bell Labs). Bell Labs has produced the leading ML implementation, Standard ML of New Jersey; the next version of New Jersey ML will have higher-order functors built in. Since ML of New Jersey is the most widely used ML implementation, our work has therefore been put into practice and will become widely available. However, we are not entirely satisfied with the semantic theory of functor application in the presence of higher-order functors (this was not addressed in the above mentioned papers) and we feel that further consolidation in the theoretical work is desirable.

Type Inferference and Storage Allocation:

Within the past six months, we have had an exciting breakthrough in this

area. We have found that it is indeed possible to use a refined notion of type inference, we call it *region inference*, to partition the values a program produces into regions, which can be allocated and de-allocated in a stack-like manner. The exciting thing is that the scheme works for languages with higher-order functions despite the fact that it is widely believed that stanc allocation is incompatible with higher-order functions. So far we have concentrated on developing a mathematically robust theory of region inference. We have proved that region inference is sound and we have an algorithm which infers regions, that are "best possible", in a sense which can be made precise. The practical implications of this work are very important, we feel, as it appears that it will be possible to implement functional languages without garbage collection and with much more modest memory requirements than has been the case till now. However, there still remains much work in extending the theoretical work to cover a full language (which will be Standard ML) and trying out the scheme in a real implementation (which will probably be the ML Kit). This work has been done in collaboration with Jean-Pierre Talpin, Ecole des Mines, Paris.

The ML Kit:

During the past year the Kit has been developed further to an extent that it now implements virtually all of Standard ML. Also, we have written about 60 pages of documentation; the entire documentation is intended to be no more than 100 pages long. A student, Lars Birkedal, has been employed as a half-time programmer, funded by DART; his job is to fill out the remaining holes in the Kit code and to help with the documentation effort. This work has been done in collaboration with Nick Rothwell of Laboratory for Foundations of Computer Science, University of Edinburgh.

4 Reports on meetings

In this section we report on meetings since the previous workshop. Most importantly this comprises an "Open Day" aimed at Danish Industry and other researchers outside the project. Additionally, we have held the first of a series of small meetings, each devoted to a single topic.

4.1 Open Day (by N.D. Jones)

On February 127 1992 an open day was held at DIKU in Copenhagen, concentrating on two of DART's main directions: partial evaluation, and modelling of parallel systems. The day's program was as follows:

1. **Partial Evaluation.** A method for automatic program generation with applications to, among others, compiling and compiler generation from a formal definition of a programming language.

   0915-1000 Neil D. Jones, DIKU: Overview lecture.

   1015-1100 T. Mogensen: Applications of partial evaluation outside programming languages; and A. Bondorf, DIKU: a description of Similix, a locally developed partial evaluator.

   1115-1200 A. Bondorf, J. Jørgensen, DIKU: Demonstrations of the Similix system.

2. **1200-1315 Lunch.**

3. **Reactive Systems.** Languages and models for describing reactive systems, including process control systems, man-machine interface drivers, and communication protocols. Focus on languages for describing tightly coupled (ESTEREL) and loosely coupled parallel systems (CCS, OCCAM, and LOTOS), together with (graphic) programming, analysis and verification environments.

   1315-1400 K. G. Larsen, IESD: Overview lecture.

   1415-1500 U. Engberg, DAIMI: The Esterel system.

   1515-1600 K. G. Larsen, U. Engberg: Demonstrations of the Esterel system.

4. **1600-1630 Discussion** of technical questions, and an evaluation of the day's events.

Participants included, in addition to DART members, six from industry: three from TFL (Teletechnical Research Laboratory), Ambrasoft, PPU Software, and IFAD (Institute for Applied Computer Science); the leader of

Denmark's newly established Basic Research Fund; and four University participants from outside DART (Aalborg, Aarhus, and Denmark's Technical University).

Many questions were asked and interest was shown in the demonstrationse The final discussion session was lively and extended well over its scheduled time. A few comments that were made:

Partial evaluation

- For industrial usage a partial evaluator must be able to handle a widely used language such as C or Pascal.
- The programs encountered in practice are large, but not structurally or algorithmically complex. Their simplicity could make them suitable for automatic optimization.
- The classical Burstall-Darlington program transformation technology is *not* practically useful, since it does not scale up to large programs. Partial evaluation appears to be better for scaling up.
- Major speedups are not essential — even a 20% speedup would be of great help, if it could be accomplished without too much work on the part of the program developer.

Interactive systems

- At least two participants said that scaling up is also a major problem in the development of interactive systems.
- One reply: *compositional* specifications allow the development of one component at a time, and so make it possible to decompose large specifications into small and more manageable subproblems.
- Another was that the HOL (Higher Order Logic) system has already given good results in developing some large interactive systems.

- The need for a *module language*, providing a way to specify the way one uses communications channels, was identified.

General

- It was mentioned that for really large projects, there is no reason not to write special-purpose compilers or other tools.
- It is often a problem to persuade customers to use specialized tools.

4.2 DART-meeting on Type Inference (by H.R. Nielson)

On May 25'th 1992 an informal meeting on Type Inference was held in Aarhus. It was organized by the DART groups SAT and OST but was attended also by members of other DART groups and by Ph.D.- and advanced MSc-students.

The meeting was arranged in order to have an informal setting in which to discuss our current work. Due to the great interest in type inference we ended up having quite a number of talks but, luckily, with a lot of interesting discussions.

The talks were grouped under three headings as can be seen from the program listed below:

Non-standard type inference

Hanne Riis Nielson: Program analyses in logical form
Fritz Henglein: Strictness analysis
Kirsten Lockner Solberg: Binding time analysis

Extensions of functional languages

Mads Tofte: Polymorphic references
Mads Tofte: Higher-order functors
Flemming Nielson: Higher-order processes

Algorithmic aspects of type inference

Michael Schwartzbach: Type inference for partial types
Fritz Henglein: On the complexity of closure analysis

The talks on *Non-standard type inference* discussed the logical (or "type theoretic") formulation of traditional program analyses such as strictness analysis and binding time analysis. Special emphasis was given to the classification of such specifications and their equivalence to alternative formulations (using abstract interpretation or other logical formulations). The talks on *Extensions of functional languages* concentrated on the traditional typing problem. The problem has recently found a satisfactory solution for higher-order functors (in the sense of Standard ML). A number of approaches have been suggested to typing polymorphic references and the discussions revealed a similarity between the problems studied here and those arising when assigning types to higher-order processes. Finally, the talks on *Algorithmic aspects of type inference* presented a polynomial time algorithm for solving constraints obtained from partial typing and the application of similar techniques to closure analysis was surveyed.

5 Publications from DART (from March 1991 to July 1992)

Below we list the publications from the project. The overall criterion has been that publication took place in the period from March 1991 to July 1992, but we have marked with an asterix those entries where almost all scientific work was performed before March 1991.

References

[1] L. Aceto and A. Ingólfsdóttir, "A theory of testing for ACP," in *Proceedings of CONCUR'91*, Lecture Notes in Computer Science, 1991.

[2] S. Agerholm, "Mechanizing program verification in HOL," in *Proceedings of the International HOL users Meeting, Davis, California*, 1991. Also available as Internal Report DAIMI IR-111, Computer Science Department, Aarhus University, 1992.

[3] T. Amtoft, "Properties of unfolding-based meta-level systems," in *Proceedings of the Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, SIGPLAN NOTICES vol. 26, no. 9, pp. 243–254, 1991.

[4] T. Amtoft and J. Larsson Träff, "Partial memorization for obtaining time behavior of a 2dpda," *Theoretical Computer Science*, vol. 98, pp. 347–356, 1992.

[5] H. R. Andersen, "Local computation of alternating fixed-points," Tech. Rep. 260, Computer Laboratory, University of Cambridge, 1992.

[6] H. R. Andersen, "Local computation of simulataneous fixed-points." Submitted for publication, 1992.

[7] H. R. Andersen, "Model checking and boolean graphs," in *Proceedings of ESOP'92*, vol. 582 of *Lecture Notes in Computer Sciences*, Springer-Verlag, 1992.

[8] H. R. Andersen and G, Winskel, "Compositional checking of satisfaction," in *Proceedings of CAV, Aalborg*, vol. 575 of *Lecture Notes in Computer Science*, Springer-Verlag, 1991. To appear in Formal Methods in Systems Design.

[9] H. R. Anderson and G. Winskel, "Checking of satisfaction," 1992. Selected for a CAV Special Issue of Formal Methods in System Design.

[10] L. Andersen, "Self-applicable C program specialization," in *Proceeding of PEPM'92: Partial Evaluation and Semantics-Based Program Manipulation*, pp. 54–61, 1992. Available as Technical Report from Yale University.

[11] L. Andersen and C. Gomard, "Speedup analysis in partial evaluation: Preliminary results," in *Proc. of Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'92)*, pp. 1–7, 1992. Available as Technical Report from Yale University.

[12] L. O. Andersen, "C program specialization," Master's thesis, DIKU, University of Copenhagen, Denmark, 1991. Student Project 91-12-17, 128 pages.

[13] L. O. Andersen, "Partial evaluation of C and automatic compiler generation (extended abstract)," in *Proc. of International Workshop on Compiler Construction*, 1992. (to appear).

[14] A. Bondorf, "Compiling laziness by partial evaluation," in *Functional Programming, Glasgow 1990. Workshops in Computing* (S. L. P. Jones, G. Hutton, and C. K. Holst, eds.), pp. 9–22, Springer-Verlag, 1991.*

[15] A. Bondorf, "Similix manual, system version 3.0," Tech. Rep. 91/9, DIKU, University of Copenhagen, Denmark, 1991.

[16] A. Bondorf, "Similix manual, system version 4.0." Included in Similix distribution, 1991.

[17] A. Bondorf, "Improving binding times without explicit CPS-conversion," in *1992 ACM Conference on Lisp and Functional Programming. San Francisco, California,* pp. 1–10, 1992.

[18] A. Børjesson, "Distinguishing properties and model cheching in TAV." In preparation, 1992.

[19] A. Børjesson and K. G. Larsen, "Equation solving using TAV." In preparation 1992.

[20] A. Børjesson K. G. Larsen and A. Skou, "Generality in design and compositional verification using TAV," In *Proceedings of FORTE'92*, 1992. To appear.

[21] J. A. Camilleri and G. Winskel, "CCS with priority choice", In *Proceedings of LICS'91*, 1991. To appear in Information and Computation.

[22] K. Cerans, "Decidability of bisimulation equivalences for parallel timer processes", in *Proceedings of CAV'92*, Lecture Notes in Computer Science, 1992.

[23] S. Christensen, H. Hüttel, and C. Stirling, "Bisimulation equivalence is decidable for all context-free processes," Tech. Rep. ECS-LFCS-92, Department of Computer Sciences University of Edinburgh, 1992. To appear at CONCUR '92.

[24] H. Dybkjær, *Category Theory, Types, and Programming Languages.* PhD thesis, DIKU University of Copenhagen, Denmark, 1991. vi+146pp. Available as DIKU report 91/11.*

[25] H. Dybkjær and A. Melton, "Comparing Hagino's categorical programming language and typed lambda calculi," *Theoretical Computer Science*, 1992. Accepted for publication by Theoretical Computer Science.*

[26] U. Engberg, P. Grønning, and L. Lamport, "Mechanical verification of concurrent systems with TLA." To appear in the Proceedings of the Fourth International Workshop on Computer-Aided Verification, 1992.

[27] A. Gammelgaard, "Constructing simulations chunk by chunk," Internal Report DAIMI IR-106, Computer Science Department, Aarhus University, 1991.*

[28] A. Gammelgaard, "Reuse of invariants in proofs of implementation," DAIMI PB-360, Computer Science Department, Aarhus University, 1991.*

[29] A. Gammelgaard, H. H. Løvengreen, C. Ø. Rump, and J. F. Søgaard-Andersen, "Base system verification." Submitted for publication, 1992.*

[30] K. Glindtvad and H. R. Nielson, "Correctness preserving transformations on a multipass OCCAM compiler," DAIMI PB-368, Computer Science Department, Aarhus University, 1991.

[31] J. C. Godskesen and K. G. Larsen, "Real time calculi and expansion theorems," in *Proceedings of FST-TCS'92*, Lecture Notes in Computer Science, 1992. To appear.

[32] C. K. Gomard, *Program Analysis Matters.* PhD thesis, DIKU, University of Copenhagen, Denmark, 1991. DIKU report 91/17.

[33] C. K. Gomard, "A self-applicable partial evaluator for the lambda calculus: Correctness and pragmatics," *TOPLAS*, vol. 14, no. 2, pp. 147–172, 1992.*

[34] C. K. Gomard and N. D. Jones, "A partial evaluator for the untyped lambda calculus," *Journal of Functional Programming*, vol. 1, pp. 21–69, 1991.*

[35] C. K. Gomard and P, Sestoft, "Evaluation order analysis for lazy data structures," in *Functiunal Programming, Glasgow Workshop 1991* (Heldal, Holst, and Wadler, eds.), pp. 112–127, Springer-Verlag, 1991.

[36] C. K. Gomard and P. Sestoft, "Globalization and live variables," in *Proceedings of the Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, SIGPLAN NOTICES vol. 26, no. 9, pp. 166–177, ACM Press, 1991.

[37] C. K. Gomard and P. Sestoft, "Path analysis for lazy data structures," in *Fourth International Symposium on Programming Language Implementation and Logic Programming, PLILP 92*, Springer-Verlag, 1992. (to appear, ultimo August).

[38] J. F. Groote and H. Hüttel, "Undecidable equivalences for basic process algebra," Tech. Rep. ECS-LFCS-91-169, Department of Computer Science, University of Edinburgh, 1991.

[39] K. Grue, "Map theory," Theoretical Computer Science, vol. 102, pp. 1–133, 1991.

[40] D. Gurr and C. Brown, "Relations and non-commutative linear logic," DAIMI PB-372, Computer Science Department, Aarhus University, 1991.

[41] D. Gurr and C. Brown, "A representation therorem for quantales." To appear in Jounal of Pure & Applied Algebra, 1992.

[42] J. Hannan, "Making abstract machines less abstract," in *Proceedings of the 5th ACM Conference on Functional Programming and Computer Architecture* (J. Hughes, ed.), vol. 523 of *Lecture Notes in Computer Science*, pp. 618–635, Springer-Verlag, 1991.

[43] J. Hannan, "Staging transformations for abstract machines," in *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulation* (P. Hudak and N. Jones, eds.), pp. 130–141, ACM Press, 1991.

[44] J. Hannan, "Implementing λ-calculus reduction strategies in extended logic programming languages," in *Proceedings of the Second Workshop*

on Extensions to Logic Programming (L. Hallnäs, ed.), Lecture Notes in Computer Science, Springer-Verlag, 1992. To appear.

[45] J. Hannan and D. Miller, "From operational semantics to abstract machines," *Mathematical Structures in Computer Science*, 1992. To appear.*

[46] J. Hannan and F. Pfenning, "Compiler verification in LF," in *Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science* (A. Scedrov, ed.), IEEE Computer Society Press, 1992.

[47] F. Henglein, "Efficient type inference for higher-order binding-time analysis," in *FPCA* (J. Hughes, ed.), vol. 523 of *Lecture Notes in Computer Science*, pp. 448–472, 5th ACM Conference, Cambridge, MA, USA, Springer-Verlag, 1991.

[48] F. Henglein, "Type inference with polymorphic recursion," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1991.*

[49] F. Henglein, "Dynamic typing," in *Proc. European Symp. on Programming (ESOP), Rennes, France* (B. Krieg-Brückner, ed.), vol. 582 of *Lecture Notes in Computer Science*, pp. 233–253, Springer-Verlag, 1992.

[50] F. Henglein, "Global tagging optimization by type inference," in *Proc. 1992 ACM Conf. on LISP and Functional Programming (LFP), San Francisco, California*, ACM Press, 1992.

[51] U. Holmer, K. G. Larsen, and W. Yi, "Decidability of bisimulation equivalence between regular timed processes," in *Proceedings of CAV'91*, vol. 575 of *Lecture Notes in Computer Science*, 1992.

[52] C. K. Holst and C. K. Gomard, "Partial evaluation is fuller laziness," in *Proceedings of Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, SIGPLAN NOTICES vol. 26, no. 9, pp. 223–233, ACM Press, 1991.

[53] P. Hudak and N. Jones, eds., *Proceedings of the Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM), New Haven, Connecticut.* Sponsored by the ACM Special Interest Group SIGPLAN, in cooperation with IFIP, ACM Press, 1991.

[54] H. Hüttel, "Decidability, behavioural equivalences and infinite transition graphs," ECS-LFCS-91-191, Department of Computer Science, University of Edinburgh, 1992.

[55] H. Hüttel, "Silence is golden: Branching bisimilarity is decidable for context-free processes," in *Proceedings of CAV91*, vol. 575 of *Lecture Notes in Computer Science*, Springer-Verlag, 1992. The full version is available as Tech. Rep. ECS-LFCS-91-173, Department of Computer Science, University of Edinburgh.

[56] H. Hüttel and C. Stirling, "Actions speak louder than words: Proving bisimilarity for context-free processes," in *Proceedings of 6th Annual Symposium on Logic in Computer Science (LICS 91)*, pp. 376–386, IEEE Computer Society Press, 1991.

[57] A. Ingolfsdottir and B. Steffen, "Characteristic formulae," *Information and Computation*, 1992. To appear.

[58] A. Ingolfsdottir and B. Thornsen, "Semantics models for CCS with values," In *Proceedings of the Workshop on Concurrency, Båstad, Sweden*, 1991.

[59] C. T. Jensen, "The concurrency workbench with priorities," in *Proceedings of CAV'91, Aalborg, Denmark*, vol. 575 of *Lecture Notes in Computer Science*, Springer-Verlag, 1992.

[60] N. D. Jones, "Computer implementation and applications of Kleene's s-m-n and recursive theorems," in *Lecture Notes in Mathematics, Logic From Computer Science* (Y. Moschovakis, ed.), pp. 243–263, Springer-Verlag, 1991.*

[61] N. D. Jones, "Efficient algebraic operations on programs," in *Preliminary Proceedings, University of Iowa* (T. Rus, ed.), 1991. Accepted for publication by Theoretical Computer Science, 1992.

[62] N. D. Jones, ed., *Selected Papers of ESOP'90. Science of Computer Programming. Volume 17, numbers 1-3, pages 1-271*, Elsevier, 1991.*

[63] N. D. Jones, "Static semantics, types and binding time analysis," in *Images of Programming* (D. Bjørner and V. Kotov, eds.), North-Holland,

1991. Further appeared in Theoretical Computer Science 90 (1991), pages 95–118.

[64] B. Jonsson and K. G. Larsen, "On the complexity of equation solving in process algebra," in *Proceedings of TAPSOFT'91*, vol. 493 of *Lecture Notes in Computer Science*, Springer-Verlag, 1991.

[65] B. Jonsson and K. G. Larsen, "Specification and refinement of probabilistic processes," in *Proceedings of LICS'91*, 1991.

[66] J. Jørgensen, "Compiler generation by partial evaluation," Master's thesis, DIKU, University of Copenhagen, Denmark, 1991.

[67] J. Jørgensen, "Generating a compiler for a lazy language by partial evaluation," in *Nineteenth Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. Albuquerque, New Mexico*, pp. 258–268, 1992.

[68] D. Kozen, J. Palsberg, and M. I. Schwartzbach, "Efficient inference of partial types," in *Proc. FOCS'92, 33rd IEEE Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania*, 1992. Also available as Tech. Rep. DAIMI PB-394, Computer Science Department, Aarhus University.

[69] D. Kozen, J. Palsberg, and M. I. Schwartzbach, "Efficient recursive subtyping," DAIMI PB-405, Computer Science Department, Aarhus University, 1992.

[70] P. Krishnan, "Distributed CCS," in *Proc. CONCUR-91*, vol. 527 of *Lecture Notes in Computer Science*, pp. 393–407, Springer-Verlag, 1991.

[71] P. Krishnan, "A model for real-time systems," in *Proc. MFCS'91*, vol. 520 of *Lecture Notes in Computer Science*, Springer-Verlag, 1991.

[72] P. Krishnan, "Real-time action," in *Proc. Euromicro Workshop on Real-Time Systems*, 1991.

[73] P. Krishnan, "A semantics for multiprocessor systems," in *ESOP'92, Proc. European Symposium on Programming, Rennes*, vol. 582 of *Lecture Notes in Computer Science*, Springer-Verlag, 1992.

[74] P. Krishnan and P. D. Mosses, "Specifying asynchronous transfer of control," in *RTFT'92, Proc. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems, Delft*, vol. 571 of *Lecture Notes in Computer Science*, Springer-Verlag, 1992.

[75] K. G. Larsen, "Proof systems for satisfiability in Hennessy-Milner logic with recursion," *Theoretical Computer Science*, vol. 72, 1990.

[76] K. G. Larsen, "The expressive power of implicit specifications," in *Proceedings of ICALP'91*, vol. 510 of *Lecture Notes in Computer Science*, Springer-Verlag, 1991.

[77] K. G. Larsen, "Efficient local correctness checking," In *Proceedings of CAV'92*. To appear in *Lecture Notes in Computer Science.*, 1992.

[78] K. G. Larsen and A. Skou, "Bisimulation through probabilistic testing," *Information and Computation*, vol. 94, no. 1, 1991.

[79] K. G. Larsen and A. Skou, "Compositional verification of probabilistic processes," In *Proceedings of CONCUR'92*. To appear in *Lecture Notes in Computer Science.*, 1992.

[80] K. G. Larsen and L. Xinxin, "Compositionality through an operational semantics of contexts," *Journal of Logic and Computation*, vol. 1, no. 6, pp. 761–795, 1991.*

[81] K. S. Larsen, E. M. Schmidt, and M. I. Schwartzbach, "A new formalism for relational algebra," *Information Processing Letters*, vol. 41, no. 3, 1992.

[82] K. Marriot, H. Søndergaard, and N. Jones, "Denotational abstract intrepretation of logic programs," *ACM Transactions on Programming Languages and Systems*, 1991. To appear.

[83] T. Æ. Mogensen, "Efficient self-interpretation in lambda calculus," *Journal of Functional Programming*, 1992. To appear.

[84] T. Æ. Mogensen, "Self-applicable partial evaluation for pure lambda calculus," in *ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation*, 1992.

[85] T. Æ. Mogensen and A. Bondorf, "Logimix: a self-applicable partial evaluator for Prolog," in *LOPSTR 92*, 1992.

[86] P. D. Mosses, "An introduction to action semantics," DAIMI PB-370, Computer Science Department, Aarhus University, 1991. Accepted for the Proceedings of the 1991 Marktoberdorf Summer School, Springer-Verlag (Series F).

[87] P. D. Mosses, "A practical introduction to denotational semantics," in *Formal Description of Programming Concepts* (E. J. Neuhold and M. Paul, eds.), IFIP State-of-the-Art Report, pp. 1–49, Springer-Verlag, 1991.

[88] P. D. Mosses, *Action Semantics*. No. 26 in Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1992.

[89] C. Mossin, "Similix binding time debugger manual, system version 4.0." Included in Similix distribution, 1991.

[90] M. A. Musicante, "The Sun RPC language semantics," in *Proc. 18th Latin-American Conference for Informatics*, 1992.

[91] A. Mycroft and M. Rosendahl, "Minimal function graphs are not instrumented," in *WSA'92 Bordeaux 1992*, Bigre, Irisa, Rennes, France, 1992.

[92] F. Nielson and H. R. Nielson, "Forced transformations of OCCAM programs," *Information and Software Technology*, vol. 34, no. 2, 1992.*

[93] F. Nielson and H. R. Nielson, "The tensor product in Wadler's analysis of lists," in *Proceedings of ESOP'92*, vol. 582 of *Lecture Notes in Computer Science*, Springer-Verlag, 1992.

[94] F. Nielson and H. R. Nielson, *Two-Level Functional Languages*, vol. 34 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1992.

[95] F. Nielson, (ed.), "Design, analysis and reasoning about tools: Abstracts from the first workshop," DAIMI PB-367, Computer Science Department, Aarhus University, 1991.

[96] H. R. Nielson and F. Nielson, "Using transformations in the implementation of higher-order functions," *Journal of Functional Programming*, vol. 1, no. 4, pp. 459–494, 1991.*

[97] H. R. Nielson and F. Nielson, "Bounded fixed point iteration," in *Proceedings of POPL'92*, ACM Press, 1992. The full version appeared as DAIMI PB-359, Aarhus University, 1991.

[98] H. R. Nielson and F. Nielson, "Finiteness conditions for fixed point iteration," in *Proceedings of Lisp and Functional Programming*, 1992. The full version appeared as DAIMI PB-384, Aarhus University.

[99] H. R. Nielson and F. Nielson, *Semantics with Applications: A Formal Introduction*. Wiley, 1992.

[100] F. Nielson (ed.), "Design, analysis and reasoning about tools: Abstracts from the second workshop," DAIMI PB-417, Computer Science Department, Aarhus University, 1992.

[101] N. Oxhøj, J. Palsberg, and M. I. Schwartzbach, "Making type inference practical," in *Proc. ECOOP '92, Sixth European Conference on Object-Oriented Programming, Utrecht, The Netherlands*, vol. 615 of *Lecture Notes in Computer Science*, pp. 329–349, Springer-Verlag, 1992.

[102] J. Palsberg, "An automatically generated and provably correct compiler for a subset of Ada," in *ICCL'92, Proc. Fourth IEEE Int. Conf. on Computer Languages, Oakland*, pp. 117–126, IEEE, 1992.

[103] J. Palsberg, *Provably Correct Compiler Generation*. PhD thesis, Aarhus University, 1992.

[104] J. Palsberg, "A provably correct compiler generator," in *ESOP'92, Proc. European Symposium on Programming, Rennes*, vol. 582 of *Lecture Notes in Computer Science*, pp. 418–434, Springer-Verlag, 1992.

[105] J . Palsberg and M. I. Schwartzbach, "Object-oriented type inference," in *Proc. OOPSLA'91, ACM SIGPLAN Sixth Annual Conference on Object-Oriented Programming Systems, Languages and Applications, Phoenix, Arizona*, pp. 146–161, 1991.

[106] J. Palsberg and M. I. Schwartzbach, "Static typing for object-oriented programming," DAIMI PB-355, Computer Science Department, Aarhus University, 1991.

[107] J. Palsberg and M. I. Schwartzbach, "Types, inheritance and assignments," 1991. Workshop held at ECOOP'91 in Geneva, Switzerland, July 1991. The collection of position papers is available from Computer Science Department, Aarhus University as PB-357.

[108] J. Palsberg and M. I. Schwartzbach, "What is type-safe code reuse?," in *Proc. ECOOP '91, Fifth European Conference on Object-Oriented Programming, Geneva, Switzerland*, vol. 512 of *Lecture Notes in Computer Science*, pp. 325–341, Springer-Verlag, 1991.

[109] J. Palsberg and M. I. Schwartzbach, "Safety analysis versus type inference," DAIMI PB-389, Computer Science Department, Aarhus University, 1992.

[110] J. Palsberg and M. I. Schwartzbach, "Safety analysis versus type inference for partial types," *Information Processing Letters*, vol. 43, pp. 175–180, 1992. Also available as Tech. Rep. DAIMI PB-404, Computer Science Department, Aarhus University.

[111] J. Palsberg and M. I. Schwartzbach, "Three discussions on object-oriented typing," *ACM SIGPLAN OOPS Messenger*, vol. 3, no. 2, pp. 31–38, 1992.

[112] K. H. Rose, "Graph-based operational semantics for lazy functional languages," in *Sema Graph '91 Symposium on the Semantics and Pragmatics of Generalized Graph Rewriting* (M. J. Plasmeijer and M. R. Sleep, eds.), (Nijmegen, Holland), pp. 203–225, Katholieke Universiteit Nijmegen, 1991. (available as Nijmegen Tech. Report 91-25).*

[113] K. H. Rose, "Explicit recursive binding," in *CTRS '92—3rd International Workshop on Conditional Term Rewriting Systems* (M. Rusinowitch and J.-L. Rémy, eds.), Lecture Notes in Computer Science, (Pont-a-Mousson, France), pp. 28–32, Springer-Verlag, 1992. To appear.

[114] K. H. Rose, "Gos-graph operational semantics," M.Sc.-thesis 92-1-9, DIKU, University of Copenhagen, Denmark, 1992. (56pp).

[115] M. Rosendahl, "Strictness analysis for attribute grammars," in *PLILP'92*, Lecture Notes in Computer Science, Springer-Verlag, 1992.

[116] M. I. Schwartzbach, "Type inference with inequalities," in *Proc. TAPSOFT'91*, vol. 493 of *Lecture Notes in Computer Science*, Springer-Verlag, 1991.

[117] P. Sestoft, *Analysis and efficient implementation of functional programs*. PhD thesis, DIKU University of Copenhagen, Denmark, 1991.

[118] H. Søndergaard and P. Sestoft, "Non-determinism in functional languages," *Computer Journal*, 1990 ? Accepted - Melbourne Technical Report 88/18.*

[119] B. B. Sørensen and C. Clausen, "Adequacy results for a lazy functional language with recursive and polymorhic types," Internal Report DAIMI IR-113, Computer Science Department, Aarhus University, 1992. Submitted to Theoretical Computer Science.

[120] M. Tofte, "Principal signatures for higher-order program modules," in *The 19th Annual ACM Symposium on Principles of Programming Languages, Albuquerque, New Mexico*, pp. 189–199, 1992.

[121] G. Winskel, "On 1ocal model checking the modal λ-calculus," 1991. In an ICALP'89, special issue of Theoretical Computer Science.

[122] G. Winskel, ed., *CLICS Workshop-Parts I and II. Proceedings of the Workshop on Categorical Logic in Computer Science*, 1992. Also available as Tech. Rep. DAIMI PB-397 I and II, Computer Science Departement, Aarhus University.

[123] G. Winskel, *The formal semantics of programming languages*. To be published by MIT Press, 1992.

[124] G. Winskel and J. Camilleri, "CCS with a priority choice," in *Proceedings of LICS*, 1991.

[125] G. Winskel and K. Larsen, "Using informations systems to solve recursive domain equations," *Information and Computation*, vol. 91, no. 2, 1991.*

[126] G. Winskel and M. Nielsen, "Models for concurrency." To appear as a chapter in the Handbook of Logic and the Foundations of Computer Science, Oxford University Press.

[127] L. Xinxin, *Specification and Decomposition in Concurrency*. PhD thesis, Department of Mathematics and Computer Science, Aalborg University, Denmark., 1992.

[128] W. Yi and K. G. Larsen, "Testing probabilistic and nondeterministic processes," *Proceedings of PSTV'92*, 1992.