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

Flemming Nielson
(editor)

October 1991

1 Introduction

The first DART workshop took place on September 16th and September 17th at Aarhus University. It attracted some 26 attendees and 16 halfhour talks were given. This booklet gives

- a brief introduction,
- abstracts of all talks, and
- a detailed description of the goals of the DART project.

The DART project concerns the "Design, Analysis and Reasoning about Tools". It is funded by the Danish Research Councils until early 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

A more detailed description of these areas may be found in Section 3 together with names of contact persons.

From the diversity of the headings it should be clear that despite the focus on semantics, the DART project comprises several research areas with their own concepts, methods and tools. To facilitate the interaction within the projects and the transfer of ideas between different areas, the majority of the talks were asked to relate to one of the following two themes:

- fixed points
- types

Both themes touched upon more than one area and often the talks managed to stimulate discussion between areas that otherwise had only little interaction.

Acknowledgements

Finally, thanks to Susanne Brøndberg for arranging the workshop and to Tove Legaard for arranging the payment of our bills.

2 Abstracts of Talks given

Mechanizing Program Verification in HOL

Sten Agerholm
Aarhus University

Proofs of program correctness are usually large and complex. This advocates mechanical assistance for managing the complexity and details of proofs. In this talk we present a program verifier based on the HOL system which is an interactive general-purpose proof-assistant system for conducting proofs in higher order logic. We describe a formalization of the weakest precondition predicate transformer semantics of a small programing languages a verification condition generator for total correctness specifications, and a number of simplification tools for proving subparts of verification conditions, automatically. Examples are consiered in order to illustrate the application and evaluate the usability of the program verifier.

Variables Determining control Flow

Torben Amtoft
Aarhus University

The purpose of partial evaluation (PE) is to do some computations at PE-time which otherwise would have to be done repeatedly at run time. The standard view on PE is as follows: some parts (variables) of the input are known *a priori* while others are not; from the known parts one attempts to do as much computation as possible. In this talk an alternative view is presented: there is no a priori binding time division, instead variables are classified with respect to how much computation is possible using their values. More precisely, one takes a look at the algorithmic structure of the program and for each loop in the flow chart an analysis is made to detect which variables must be known in order for this loop to be eliminatable. The result of the analysis can give helpful clues on how the program in question should be specialized. The analysis can be considered as a backward binding time analysis, and is as such in some sense resemblant of strictness analysis.

Several examples will be given to illustrate the topics. It will be discussed whether it is possible to put the concept on more solid ground, and related approaches will be mentioned.

Soundness and Consistency of The UNITY Logic with Induction

Flemming Andersen
Teleteknisk Forsknings-Laboratorium

It has been discussed whether the UNITY logic, as presented in the book by Chandy and Misra: Parallel Program Design, A Foundation, is consistent. One of the questions concerning the logic has been the use of two informally presented induction principles. The two principles are used to prove theorems of the progress relation "leadsto" in the logic.

Working with the construction of minims fixed points using the Tarski method, it was discovered that the implication theorem, satisfied by the construction of the fixed point, may be used for deriving induction principles for the minimal fixed point of boolean recursive functionals.

Assuming the monotonic functional maps the boolean function to be defined into a disjunction of boolean functions, it is possible to prove that a set of introduction theorems for the fixed point is automatically given. Further the implication theorem may be used for proving two induction theorems for the fixed point satisfied.

In this talk these results are presented and used for defining "leadsto" as a minimal fixed point solution. It is shown that the "leadsto" relation may be defined by two different monotony functionals, resulting in the same minimal fixed point. Hence, four induction theorems are given for the found "leadsto" definition. Two of these represent the induction principles as described in the UNITY book. But it turns out that one of the used induction theorems is slightly different from the formally found.

Model Checking and Boolean Graphs

Henrik Reif Andersen
Aarhus University

This paper describes a method for translating a satisfaction problem of the modal μ-calculus into a problem of finding a certain marking of a boolean graph. By giving algorithms to solve the graph problem, we present a global model checking algorithm for the modal μ-calculus of alternation depth one, which has time-complexity $|A||T|$, where $|A|$ is the size of assertion and $|T|$ is the size of the model (a labeled transition system). This algorithm extends to an algorithm for the full modal μ-calculus which runs in time $(|A||T|)^{ad}$, where ad is the alternation depth, improving on earlier presented algorithms. Moreover, a local algorithm is presented for alternation depth one, which runs in time $|A||T| \log(|A||T|)$, improving on the earlier known algorithms that are all at least exponential.

Correctness Preserving Transformations on a Multipass Occam Compiler

Karin Glindtvad
Aarhus University

The verification of a compiler may be a substantial task. However, by introducing correctness preserving program transformations some automated assistance becomes available. The idea is to specify an initial multipass compilers to verify it in the usual way and then, while preserving the overall correctness result, to transform it into a more efficient single pass compiler. This transformation process may be performed using the fold/unfold framework of Burstall and Darlington and automation is provided by the Flagship Programming Environment. I will illustrate the transformation process on a compiler for a small subset of Occam.

Path Analysis for Lazy Data Structures

Carsten K. Gomard
University of Copenhagen

We describe a method to statistically infer evaluation order information for data structures in a typed lazy first order functional language. Our goal is to determine to which extent and in what order the variables and data structures in the program are evaluated. This information subsumes strictness analysis but can be used to optimize the implementation of suspensions (or "thunks") in such languages.

The evaluation order of the free variables in an expression is described by *variable paths* in the style of Bloss and Hudak.

Evaluation order for a data structure is described by a *context*, or *evaluation order type*, which is the type of the data structure together with path sets describing the order of evaluation of its components. For recursively defined types such as `natlist ::= Nil | Cons nat natlist`, only uniform descriptions are allowed: the recursive components (those of type `natlist`) must all have the same description.

Given a context in form of an evaluation order type for the result of an expression, we find the order in which the variables are evaluated. Also from the context of the expression we find contexts for the free variables. Similarly, from the evaluation order type of a function application we get evaluation order information about its parameters.

This work has goes related to those of Bloss and Hudak's "path analysis". It extends their resets because the backwards approach facilitates analysis of data structures. Moreover, evaluation order types seem to be natural tools for describing and reasoning with data structure strictness. In particular, one can characterize head strictness also in the absence of tail strictness.

In the paper we discuss the description of evaluation order and evaluation order types, describe the analysis method, demonstrate it on several examples, and discuss its application to optimization of suspensions.

Mechanical Verification of Translations Between Operational Semantics

John Hannan
University of Copenhagen

We address the problem of mechanically verifying the equivalence of two different operational semantics. We consider operations semantics defined as sets of inference rules axiomatizing a relation between programs p and values v. A computation in such a semantics is described by a proof of a proposition $(p \Downarrow v)$. Equivalence between two semantics axiomatizing the same such propositions can then be demonstrated by providing a bidirectional translation between proofs (of the same proposition) in the two semantics. Previously these proofs of equivalence were performed by hand. We demonstrate how they can be defined in the same logical framework in which the semantics are specified.

We use the metalanguage Elf, a logic programming language that incorporates logic definition facilities in the style of LF, to define operations semantics. Unlike other logic programming languages, such as Prolog and λProlog, Elf provides direct access to the proofs of queries or goals. These proofs are represented as terms in a dependent typed λ-calculus and can be directly manipulated by other Elf programs. We provide an example in which two operational semantics for reduction in the untyped λ-calculus can be proved equivalent via an Elf program relating the two semantics. We also comment on an extended example in which we relate an operational semantics using a higher-order abstract syntax and meta-level substitution to perform β-reduction, with a semantics using a first-order abstract syntax, environments and closures.

Proving Bisimilarity for Context-Free Processes

Hans Hüttel
Aalborg University Centre

Recently Baeten, Bergstra, and Klop have proved the remarkable result that

bisimulation equivalence is decidable for context-free grammars without useless or empty productions. Within process calculus theory these grammars correspond to normed processes defined by a finite family of guarded recursion equations in the signature of BPA (Basic Process Algebra). These processes can have infinitely many states (even after quotienting by bisimulation equivalence). Consequently the process calculus approach encompasses a much richer class of infinite-state systems that are open to automatic techniques normally associated with finite state systems than all those approaches based on trace or language equivalence.

However, the proof of decidability is not easy as it relies on isolating a possibly complex periodicity from the transition graphs of these processes. An alternative, more elegant proof utilizing rewrite techniques has been given by Caucal. But neither of the proofs reflect how one intuitively would show that two processes are bisimilar.

In this talk I present a result due to Colin Stirling and myself which gives a simpler and much more direct decidability proof using a tableau-based decision method involving goal-directed rules akin to the branching algorithms first investigated by Korenjak and Hopcroft. The decision procedure yields an upper bound on the depth of a tableau and provides the essential part of the bisimulation relation between two processes which underlies their equivalence, a self-bisimulation as in the work of Caucal. From the decision procedure we also get a sound and complete sequent-based equational theory for such processes.

Dynamic Typing

Fritz Hengkin
University of Copenhagen

The world of programming languages has historically been divided into statically (Pascal, Miranda (TM), ...) and dynamically (Scheme, Prolog, ...) typed languages, each camp being populated by advocates that are outspoken, at times ferociously so.

Statically typed programming languages require type declarations — explicit (Pascal) or implicit (Miranda) — that are checked at compile-time; dynam-

ically typed languages conduct type checking at run-time. Both camps have their proponents: statically typed languages are safer and mostly more efficient than dynamically typed ones, and they support software engineering principles such as modularization and information hiding better, it is argued by one camp. The other camp extols the flexibility and conciseness of dynamically typed languages, and their suitability for rapid prototyping and exploratory software development.

In this talk we present a way of combining both typing disciplines in a single frameworks called "dynamic typing", without compromising the benefits of either. We present an explicit typing discipline with a special type "dynamic" and operations for type tagging and type checking to, respectively, create and use values of type dynamic. Type information and tagging/checking operations may be completely or partially omitted, and a type inference algorithm is employed to infer the missing type information and type operations.

Apart from integrating seemingly contradictory typing and using characteristics in a single language framework, dynamic typing also bears more immediate technological promise. The type inference algorithm gives the same result as Miranda's type inferencer for statically type correct programs; for type incorrect ones it produces a typing in which the placement of type operations localizes the (static) type errors. In a LISP-like language the algorithm can be used to speed up the execution of programs by eliminating unnecessary tagging operations.

Typing Interpreters, Compilers and Partial Evaluators

Neil D. Jones
University of Copenhagen

A *symbolic version* of an operation on values is a corresponding operation on program texts. For example, symbolic composition of two programs p, q yields a program whose meaning is the (mathematical) composition of the meanings of p and q. Another example is symbolic specialization of a function to a known first argument value. This operation, given the first argument, transforms a two-input program into an equivalent one-input program. Computability of both of these symbolic operations has long been established in

recursive function theory; the latter is known as Kleene's "s-m-n" theorem, also known as *partial evaluation*.

In additon to computability we are concerned with *efficient* symbolic operations, in particular applications of the two just mentioned to compiling and compiler generation. Several examples of symbolic composition are given, culminating in nontrivial applications to compiler generation. Partial evaluation has recently become the subject of considerable interest. Reasons include simplicity, efficiency and the surprising fact that self-application can be used in practice to generate compilers, and a compiler generator as well.

This paper makes three contributions: First, it introduces a new notation to describe the types of symbolic operations, one that makes an explicit distinction between the types of program texts and the values they denote. This leads to natural definitions of what it means for an interpreter or compiler to be type correct — a tricky problem in a multilanguage context. Second, it uses the notation to a clear overview of several earner applications of symbolic computation. For example, it is seen that the new type notation can satisfactorily explain the types involved when generating a compiler by self-applying a partial evaluator. Finally, a number of problems for further research are stated along the way. The paper ends by suggesting Cartesian categorical combinators as a unifying framework in which to study symbolic operations.

Efficient Local Validity-Checking

Kim Guldstrand Larsen
Aalborg University

Automatic verification of parallel systems is a growing research area. In particular a number of tools and techniques for deciding various behavioural equivalences between processes and checking satisfiability of processes with respect to temporal logic formulae has been developed. Most of these techniques share a preprocessing phase in which the total state space of the processes involved is constructed based on which the equivalence and satisfiability checking is performed. However, the exponential growth of the state space in the number of parallel components has been identified as the main limiting factor for these techniques.

In contrast, the technique we offer will only explore (and construct) the part of the state space which is necessary to carry out the verification. We call this technique a *local checking* technique. Other researchers (Colin Stirling and Glynn Winskel) have previously considered similar local checking techniques but in all cases they have exponential worst-case time complexity. Our technique offers an *efficient* local checking technique, based on a dynamic programming technique in order to avoid unnecessary recomputations.

Our local checking technique is developed for a *Boolean Equation System Problem*. Here a number of propositional variables $x_1, \ldots, x_n$ are given together with a defining equation $x_i = D_i$ for each variable x_i, with the definition D_i being the formula tt, the formula ff or a conjunction or disjunction of propositional variables. The problem is to decide whether

$$x_1 \in V$$

where V is the maximum set of variables such that assigning tt to all variables in V and ff to all variables outside V validates all equations. In a local checking of the above we want to consider as few propositions variables as possible. The algorithm developed has quadratic time complexity in the number of propositions variables. Finally, it is shown how to translate a number of process valdating problems (e.g. bisimulation, equivalence, satisfiability wrt. ν-calculus) into Boolean Equation System Problems.

Efficient Self-Interpretation in Lambda Calculus

Torben Æ. Mogensen
University of Copenhagen

This note is in part a follow up to Henk Barendregt's Theoretical Pearl in Issue 2 of the Journal of Functional Programming ([Barendregt 1991]), but contains sufficient new ideas to be interesting in its own right.

We start by giving a compact representation of λ-terms and show how this leads to an exceedingly small and elegant self-interpreter. We then define the notion of a *self-reducer*, and show how this too can be written as a small λ-term. Lastly, the question "is there a λ-term that reduces to its own representation" is asked. We answer this affirmatively by constructing such a

term. All the constructions have been implemented on a computer, and experiment runs verify their correctness. Timings show that the self-interpreter and self-reducer are quite efficient, being about 35 and 50 times slower than erect execution using a call-by-need reduction strategy.

A Perspective on Types

Flemming Nielson
Aarhus University

We begin by briefly reviewing some of the many "non-standard" uses of types that may be found in the literature. We then go on to study two of these in greater detail.

One is the introduction of *binding times* into the notation for types (and expressions). There seems to be some consensus on the notation to use whereas there are many conflicting approaches to the definition of well-formedness. We demonstrate that indeed different notions of well-formedness are appropriate for different purposes but that nonetheless several of the techniques used have a lot in common. Based on the insights obtained we propose a general methodology for constructing a **B**-level language L where **B** is the "set" of binding times and L is some typed language.

The other is the introduction of *communication abilities* into the notation for types. As a price to be paid for a decidable type system we have to give up causality. The proposed system is distinctive in character (with respect to Mobile Processes and similar systems) but allows proving a result along the lines that "well-formed programs do not go wrong". Finally, we briefly mention the possibility of using a **B**-level version of the language as a metalanguage for Denotational Semantics, thus increasing the descriptive power of Denotational Semantics in the area of concurrency.

Bounded Fixed Point Iteration

Hanne Riis Nielson
Aarhus University

In the context of abstract interpretation for languages without higher-order features we study the number of times a functional need to be unfolded in order to give the least fixed point. For the cases of total or monotone functions we obtain an exponential bound and in the case of strict and additive (or distributive) functions we obtain a quadratic bound. These bounds are shown to be tight in that sufficiently long chains of functions can be shown to exist. Specializing the case of strict and additive functions to functionals of a form that would correspond to iterative programs we show that a linear bound is tight. This is related to the analyses studied in the literature (including strictness analysis).

The results are presented in H.R. Nielson, F. Nielson: Bounded Fixed Point Iteration, DAIMI PB-359, Aarhus University. An extended abstract will appear in the proceedings from POPL92.

Inheritance of Recursive Classes

Michael Schwartzbach
Aarhus University

The semantics of inheritance is commonly explained as a program transformation that expands class definitions. It is well-known that the simplest formulation of this yields unsatisfactory results when applied to inheritance of recursive classes: the recursive structure is not preserved in the subclass. Modifications such as the "like Current" construct have been suggested, but they do not generalize in any obvious way to mutually recursive classes. We present a new program transformation for inheritance that always preserves the recursive structure. It is based on a representation of classes as regular trees labeled by gapped, untyped source code. The algorithm analyzes the so-called program graph which has class definitions as nodes and Is-a and Has-a edges; it then reflects inheritance as operations on regular equations systems. A crucial property of this transformation is that it may introduce new classes in the expanded program.

The Compositional Checking of Satisfaction

Glynn Winskel
Aarhus University

This talk described the joint work of Andersen and Winskel. It presents a compositional method for deciding whether a process satisfies an assertion. Assertions are formulae in a modal ν-calculus, and processes are drawn from a very general process algebra inspired by CCS and CSP. Well-known operators from CCS, CSP, and other process algebras appear as derived operators. The method is compositions in the structure of processes and works purely on the syntax of processes. It consists of applying a sequence of reductions, each of which only take into account the top-level operator of the process. A reduction transforms a satisfaction problem for a composite process into equivalent satisfaction problems for the immediate subcomponents. Using process variables, systems with undefined subcomponents can be defined, and given an overall requirement to the system, necessary and sufficient conditions on these subcomponents can be found. Hence the process variables make it possible to specify and reason about what are often referred to as contexts, environments, and partial implementations. As reductions are algorithms that work on syntax, they can be considered as forming a bridge between traditional non-compositional model checking and compositional proof systems.

3 Description of the DART project

The acronym DART stands for "Design, Analysis and Reasoning about Tools". What follows is an edited excerpt from the application for funds. It is intended as an overall description of the project and was written in January of 1991.

3.1 Objective of Project

To conduct research about programs and systems using tools and techniques that have a strong semantic foundation. Most of these tools and techniques focus on aspects near the borderline between theory and practice and may involve theoretical as well as experimental components. As a consequence they have a clear potential for practice applications in all areas where computer based systems are (to be) used.

3.2 Introduction

The present project brings together leading Danish researchers in the exploitation and development of formal techniques for a number of issues related to programming languages. The programming languages include parallel and functional features; the language issues include semantics, analysis, verification and manipulation of programs.

Members of the group share the objective of basing such work on a formal semantics of programming languages. They complement one another in their study of functional versus parallel features, in the choice of synthetic versus analytical methodologies, and in balancing the development of new theories against the application of existing theories.

While these activities are mainly motivated by considerations of basic research they are strategic in nature, in that virtually all of the considerations are of potential concern for realistic development of software in a number of application areas. Based on experience from other countries, including the cooperation between academia and industry, it seems plausible that several of the techniques mastered by members of this group will need to be utilized

by Danish industry already in this decade, if it is to remain competitive.

To support the basic research nature of the project there will be a *steering committee* (entirely internal to the project); this has shown to be a valuable tool in the ESPRIT Basic Research Actions, which members of this project are involved in. In an attempt to maintain the strategic focus of the research there will be a contact committee where members of the steering committee will discuss the overall direction of the project with external representatives. To ensure overall coordination there will be a *project coordinator* (who will be a member of both committees).

3.3 Applicants Participating in the Project

The project will involve researchers at **DAIMI** (The Department of Computer Science at Aarhus University), **DIKU** (The Department of Computer Science at Copenhagen University) and **IESD** (The Institute of Electronic Systems at Aalborg University Centre) and may include researchers at other institutions when their specialty is considered important for the overall success of the project. Each of the applicants listed below will devote at least two thirds of their research time to activities listed in this proposal.

At **DAIMI** the group presently consists of:

Uffe Engberg, *assistant professor, Ph.D.*
Peter Mosses, *associate professor, Ph.D.*
Hanne Riis Nielson, *associate professor, Ph.D.*
Fleming Nielson, *associate professor, Ph.D., D.Sc.*
Michael Schwartzbach, *associate professor, Ph. D.*
Glynn Winskel, *professor, Ph.D.*

It is expected that several guests, Ph.D. students, and research assistants will be involved as well. Possible names include Anders Gammelgaard (RA), Jens Palsberg, Padmanabhan Krishnan (guest, Ph.D.), Henrik Andersen, Douglas Gurr (guest, Ph.D.), Carolyn Brown (guest, Ph.D.), Torben Amtoft.

At **DIKU** the group consists of:

Klaus Grue, *associate professor, Ph.D.*
Neil Jones, *professor, Ph.D.*

Torben Mogensen, *assistant professor, Ph.D.*
Mads Rosendahl, *assistant professor.*
Mads Tofte, *expected to become associate professor, Ph.D.*

It is expected that several guests, Ph.D. students, and research assistants will be involved as well. Possible names include Anders Bondorf (RA, Ph.D.), Ritz Henglein (RA, Ph.D.), John Hannan (guest, Ph.D.), Peter Sestoft, Carsten Gomard, Hans Dybkjær, Kristoffer Helm, Lars Ole Andersen, Jesper Jørgensen, Christian Mossin, Nils Andersen (associate professor)

At **IESD** the group consists of:

Kim Larsen, *associate professor, Ph.D.*

It is expected that also Arne Skou (associate professor), Anna Ingolfsdottir (assistant professor) and Hans Hüttel (assistant professor) will be involved.

3.4 Scientific Content

The bulk of the research is organized into a number of designated research areas:

SDT: Semantics as a Descriptive Tool

SAT: Semantics as an Analytical Tool

SOC: Semantics of Concurrency

SBD: Semantics Based Deduction

SBPM: Semantics Based Program Manipulation

OST: Operational Semantics, Types and Language Implementation

For each we list its title, likely participants (including a *contakt pepson*), its purpose and likely results to be achieved within one or two years. This distinguishes between the results we hope to achieve with only limited funding (e.g. perhaps some programming assistance, some travel money and minor amounts of equipment) and those results that need considerable additional funding (e.g. guests on extended stays, research assistants, full-time programmers, expensive equipment).

3.4.1 SDT: Semantics as a Descriptive Tool

The main activities in this area center around Action Semantics. This is an approach to semantics (developed mainly by Peter Mosses) that has its roots in Denotations Semantics but is designed so as to allow greater modularity of semantic descriptions. This is done by identifying four important facets of programming languages (concurrency, binding, functional, imperative) and by formulating a general set of combinators for expressing these.

During the coming years the main activities will be:

> To support the popularization of Action Semantics by producing a library of polished action semantic descriptions (including *Pascal, Modula-3, Standard ML, Beta, Occam-2*).
>
> To develop the theory behind Action Semantics: laws, proof techniques, etc.
>
> To develop an environment for editing, browsing, checking and interpreting action semantic descriptions. This may be based on the system *Mathematica*.

Likely participants are Peter Mosses (contact person), Jens Palsberg, Padmanabhan Krishnan.

If further funding and manpower is available, other aspects of Action Semantics may be studied: software specification, Action Semantics and language standardization, relationship to *Meta-IV* and *RAISE* etc.

3.4.2 SAT: Semantics as an Analytical Tool

The main activities of this area are concerned with the correct and efficient implementation of lazy functional languages like *Miranda* or *Haskell*. Important ingredients are the systematic construction of abstract interpretations, the development of (provably correct) implementations for sequential and parallel abstract machine architectures, and the application of program transformations to increase the efficiency of various implementation schemes.

During the coming years the main activity we be to complete the development of a two-level framework for describing and reasoning about code generation, abstract interpretation and their combination. The theoretical part of this work includes:

a correctness proof for a "simple" translation,

formulation and proof of correctness for abstract interpretations that may facilitate more efficient code to be generated,

combination of code generation and abstract interpretation and studies of the correctness problem associated with this.

To complement the theoretical work it is planned to construct and use various tools to experiments with different approaches. These may include the *HOL* system (see the description of Semantics Based Deduction) and the *FLAGSHIP* transformation system. More specialized tools may be written directly in a functional language.

Likely participants are Hanne Riis Nielson (*contact person*), Flemming Nielson, Jens Palsberg, Torben Amtoft.

If further funding and manpower is available, it is planned to combine ideas from the two-level approach with state-of-the-art techniques used in the implementation of *Haskell*.

3.4.3 SOC: Semantics of Concurrency

The main activities in this area are concerned with the development of a theoretical basis and supporting automatic tools for designing provably correct distributed/concurrent systems. In particular, theories and tools supporting 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.

During the coming years the main activity will be to study and develop a range of specification formalisms for concurrent systems and to investigate to what extent the specification formalisms support compositions verification. The theoretical part of this work includes:

study of the compositionality question for a modal μ-calculus,

study of the compositionality question for extensions of the modal μ-calculus allowing explicit representation of timing constraints and properties of probabilistic behaviour of a concurrent system,

study of the compositionality question for true concurrent specification formalisms.

To complement the theoretical work it is planned to design and (prototype) implement a range of automatic tools supporting the design methodology. These tools may be implemented directly in a functional language, in a logic programming language (similar to the implementation of the existing *TAV*-system) or in the *HOL* system (see the description of Semantics Based Deduction). Also, an evaluation of the design methodology (and of the associated tools) through practice experiments is planned.

Likely participants are Uffe Engberg, Kim Guldstrand Larsen (*contact person*), Glynn Winskel, Henrik Andersen, Arne Skou, Anna Ingolfsdottir and Hans Hüttel.

If further funding and manpower is available an efficient implementation of a programming environment integrating the various tools is planned.

3.4.4 SBD: Semantics Based Deduction

On October 1-2, 1990 the Computer Science Department of Aarhus University hosted the Third International *HOL*, Users Meeting, attended by around 40 people from as far afield as California, Vancouver and Canberra, Australia. *HOL* is a higher-order-logic theorem prover originally developed at Cambridge University. It is becoming widely accepted as one of the most important tools in the area of automated proof and is being applied by universities and industry in the verification of safety-critical hardware and software as well as in the automated proof of standard mathematics. The siting of the meeting at Aarhus was in part a recognition and encouragement of the promising work being done there by several M.Sc. students under the direction of Glynn Winskel. However, securing and directing this expertise requires careful planning and funding.

The goals of this activity require substantial funding and are

the establishment of a site of *HOL* expertise open to a variety of applications by universities and industry, in teaching, consultancy and research,

a broadening of this expertise to other related theorem provers especially Coquand's *Calculus of Constructions* and Paulson's *Isabelle*. Such type theories are non-trivial and warrant theoretical study,

the development of tools for automated proof. In particular it is planned to investigate automated support for transformations in functional programming languages (possibly based on recent ideas of Abramsky).

Although it is not yet the case that *HOL* or similar systems are being widely used by Danish industry (*HOL* is used by *Teleteknisk Forsknings-Laboratorium*, however), the growing importance of automated verification is well recognized outside Denmark. We have contact with researchers in the area at *AT & T Labs.* (New Jersey), *SRI* and *TopExpress* (Cambridge) and *DEX* (Palo Alto). These and links with other European universities through our ESPRIT Basic Research Actions will be an advantage in establishing machine verification as an area of expertise in Denmark, in anticipation of a fruitful interchange with Danish industry over the next decade.

Likely participants include Glynn Winskel (*contact person*), Douglas Gurr, and Sten Agerholm.

3.4.5 SBPM: Semantics Based Program Manipulation

This area has three highly intertwined threads:

Partial evaluation, an automatic technique for program optimization by specializing a program to partially known input data. Self-application of a partial evaluator can be used to generate program generators, a notable example being to transform a programming language interpreter into a compiler.

Abstract interpretation, concerns techniques for static program analyses; their goal is to extract information about a program's runtime behaviour without actually executing it. Central

concerns are safety: correctness of the analysis; computability by means of terminating algorithms; precision of the behavioural description.

Implementation of functional languages. Efficient implementation techniques for two closes of functional languages: strict: (*Standard ML FP, ...*) and lazy (*Miranda, Haskell, ...*).

Following are some specific research goes:

to extend partial evaluation to handle larger languages and to improve efficiency,

to find problem areas to which partial evaluation can profitably be applied,

to find static analyses and program transformations to obtain better results during partial evaluation,

to complete a distribution version of a system (*Similix*) for the partial evaluation of a higher-order subset of *Scheme*; currently pre-releases are being used at European and American universities,

to investigate design decisions for functional language implementation, e.g. tagged versus tagless data and heap versus stack based control,

to design and implement static program analyses that gather information useful in generating efficient code, e.g. storage usage, variable lifetimes, and linearity,

to develop highly parallel graph reduction machines, to experiment with simple versions of the proposed implementation techniques, and to obtain statistics relating their efficiency to programs in e.g. the programming language C,

to better understand backwards abstract interpretation of functional languages, and to relate forwards and backwards analyses,

techniques to combine abstract interpretations, for example using attribute grammars as a descriptive language to set up an abstract interpretation implementation framework,

to better relate abstract interpretation to types, e.g. to compare the former's fixpoint based techniques with the deductive and inferential techniques used for types,

a complexity theory about the speedups seen in partial evaluation (computerized prediction methods are essential to the fully automatic use of partial evaluation).

The participants are expected to include Neil Jones (*contact person*), Klaus Grue, Mads Tofte, Mads Rosendahl, and Torben Mogensen. In addition, some ESPRIT researchers (Anders Bondorf, Ritz Henglein) and M.Sc. and Ph.D. students (C. Gomasd, L.O. Andessen, J. Jørgensen, C. Mossin, Peter Sestoft).

If sufficient funding and manpower is available, we would do some of the following:

implement partial evaluators for full or almost full versions of commercially available and widely used languages (examples could be *Scheme*, *Prolog* or *Miranda*),

realize a parallel graph seduction machine. Presently at DIKU, 5 transputers implementing a graph seduction machine (as yet unoptimized) can deliver one tenth of the computational power as a *SPARC* station programmed in *C*,

develop a prototype system using attribute grammars to implement and apply abstract interpretation to a wide spectrum of programming languages.

3.4.6 OST: Operational Semantics, Types and Language Implementation

Operational Semantics is a simple and powerful mathematical framework for defining the semantics of psogsamming languages. The programming language *Standard ML* is defined using the method (see Milner, Tofte, Harper: The Definition of Standard ML, MIT Press 1990). Also, operational semantics allows precise analysis of the defined language. One can investigate the mathematics properties of the language and study problems that are essential

for a good implementation in a precise setting (see Milner, Tofte: Commentary on Standard ML, MIT Press 1990) Finally, it has become apparent that it is highly helpful to use the formal semantics for a manual translation into a set implementation. This direction is being explored by Mads Tofte together with Nick Rothwell and David Turner of Edinburgh University in the so-called *ML Kit*. The *ML Kit* is a highly modular piece of software where the individual pasts are so clearly separated that they can be put togethes in different ways for different purposes and combined with new modules depending on what kind of analysis or translation one wants the *Kit* to perform. At present the *Kit* can be assembled to form two different compilers and two different interpreters for a considerable subset of *ML*.

The work of the proposed project will be practical and theoretical. This is illustrated by three of the areas that will be addressed:

> (Theoretical) The semantics of higher-order functors in *ML*, including sharing, enrichment and contravariance in functor signature matching. Do principal signatures exist?
>
> (Theoretical and practical) Can one use (non-standard) type checking to determine good run-time storage allocation?
>
> (Practical) The further development, maintainance and documentation of the *ML Kit*.

Of these, the last point which is a case study in modular programming in *ML* as much as it is the development of a particular tool which we need, should clearly be of interest to software developers. In particular, Danish industry would probably have some interest in this, considering that our practice experience so far has convinced us completely that a type secure module system is of immense significance for the productivity of programmers, both when they work as individuals and when they work in teams. In the history of *ML*, there is already a tradition of fruitful cooperation between language designers and industrial partners who build *ML* compilers or systems based on *ML*. With increased knowledge about the semantics and implementation of the language available in Denmark, one word hope that this tradition could be established in Denmark as well.

The participants include Mads Tofte (*contact person*) and Neil Jones.

3.4.7 Special Topics

It has often been said that the best research is the unexpected research. With a view to this we want to keep the project open for research topics not singled out in the headings or descriptions above. These may arise in the course of the project whereas others (detailed below) can be foreseen from the outset. It will not, however, be our intent to use major funding in this area.

We foresee work on the study of (explicit and implicit) **type systems** for imperative programming languages, both traditional and object-oriented. The presence of assignments in combination with various type features (recursive types, multiple inheritance, parametric polymorphism, classes) requires different techniques from those applied to purely functional languages. We investigate a class of automata-based methods that are already proved useful in many contexts. This work is manly to be conducted by Michael Schwartzbach.

We also foresee work on **Map Theory**. The purpose of this activity is to combine set theory and λ-calculus into one theory which can handle all of classical mathematics as well as algorithmic and computes science seasoning. The constructive past of map theory constitutes a functional programming language. It would seem that map theory is suited to formalization of metamathematics and the handling of large proofs. To this end map theory will be compared to more established branches of mathematics such as category theory and type theory. This work is mainly to be conducted by Klaus Grue.

The *contact person* for research under this heading is the project coordinator.

3.5 Pattern of Cooperation

While it is true that the research areas have been defined so as to correspond to well-established patterns of cooperation among participants in this project, we would like to open up for additional cooperation.

Currently these is alseady some contact between the groups on *Semantics as a Descriptive Tool, Semantics as an Analytical Tool* and *Semantics Based Program Manipulation*; this takes place within a grant on *Formal Implementation, Transformation and Analysis of Programs* awarded by the Danish

Natural Science Research Council. (This grant has now terminated due to the commencement of the DART project.) Within the group on *Semantics of Concurrency* these are already many contacts between researchers at **DAIMI** and **IESD**. Finally, both **DAIMI** and **DIKU** run frequent research seminars involving, amongst others, all project members at the particular site.

During this project we will make additional efforts to increase the possibility of cooperation, not least between groups at different sites. In the area of *Semantics as a Descriptive Tool, Semantics as an Analytical Tool* and *Semantics Based Program Manipulation* cooperation is expected to be facilitated by the participation in the continuation of the *Semantique* ESPRIT Basic Research Action. Similarly, the cooperation between the sites within *Semantics of Concurrency* and *Semantics Based Deduction* (6.4) is expected to be facilitated by the participation in the continuation of the ESPRIT Basic Research Actions on Concurrency and the *CLICS* Basic Research Action. Finally, we intend to arrange annul meetings within the entire project where the results of the different research areas will be presented and discussed.

3.6 Managerial Structure

The project is intended to be a "rammeprogram" and will be managed by a *steering committee*, a *project coordinator* and a *contact committee*.

Members of the *steering committee* will be the senior researchers involved in the research areas of the project: Klaus Grue, Neil Jones, Kim Larsen, Peter Mosses, Hanne Riis Nielson, Flemming Nielson, Mads Tofte, Glynn Winskel. The charter of the steering committee is to overview the scientific progress, to discuss its relation to potential users, to influence the future evolution of the project, and to encourage and plan for cooperation between the sites.

For each research area, one member of the steering committee is designated as a *contact person*. It is the duty of this person to coordinate the responsibilities of the steering committee as concerns the particular research area.

The *project coordinator* is Flemming Nielson. The duties of the project coordinator is to monitor the project, not least the expenditure of the budget. Resources for this role have been included in the total budget. Responsibili-

ties for mini-budgets within the overall budget may be delegated to members of the steering committee.

The *contact committee* is responsible for securing links with outside users. The *internal* members of the contact committee are Neil Jones, Flemming Nielson and Glynn Winskel. The *external* members of the contact committee are Ove Færgemand (from TFL), Søren Prehn (from CRI), and Anders Ravn (from the Department of Computer Science at The Technical University of Denmark). It also has a formal role in monitoring the progress of the project.

3.7 Relevance for Danish Industry

Though oriented towards basic research, this project is strategic in nature. There are several areas of application in which the expertise and techniques to be furthered by this project are likely to be important in the future development of software and hardware in Denmark.

For example, formal methods are essential in the development of safetycritical systems and are increasingly recognized as worthwhile from an economic viewpoint. This attitude is becoming prevalent in the UK and USA where many companies regard verification or formal development of software as effort well-spent because of the time and money it saves in the long run through avoiding system or network crashes, or other disasters.

We believe that techniques possessed or to be developed by members of this group will be usefd for enhancing the techniques of software construction and validation in Denmark. Presently, *Teleteknisk Forsknings Laboratorium* (*TFL*) is working with such techniques originating from members of the group. (We are currently exploiting with *TFL* the possibility of joint work in Concurrency and on *HOL*.). To further the flow of ideas between members of this project and industry we intend arranging "open days" where our work and the goals of the project will be presented through tutorials, more advanced talks and discussions. The first such arrangement is likely to take place in 1992.