MFPS Abstracts

May 6, 1998

Contents

Note: Comprox and Continuations are blocks of talks, but not "official special sessions" like the other two.

Invited Speakers

David Pym

Chantal Berline
Simple models for System F

We will present here two simple modelisations of Girard's and Reynolds' polymorphism (System F), one related to Scott's continuous semantics and the other one to Girard's coherent semantics. These modelizations are more simple than the previous existing ones and in particular do not require categorical tools. The models in both classes are rich in polymorphic maps (although there is an unexpected difference in this respect between the continuous case and its stable analogue). Many of these models can be read as two intertwinned "webbed" models of untyped lambda-calculus, the simplest of them being the Engeler-Plotkin model viewed as "self-intertwined". The class also contains other models, like the extensional model proposed very recently by S. Berardi and Barbanera, which was shown to be complete for F_η by Berardi and I, and from which the present classes arise in fact.

Samson Abramsky
Some New Uses for Domain Theory

We show how domain theory can be used to formalize a natural notion of concurrent games. Applications include modelling a general form of branching quantifiers and "independence-friendly" logic in the sense of Hintikka and Sandu; a full completness theorem for Multiplicative-Additive Linear logic; and full abstraction for PCF. "Classical" domain-theoretic notions such as stability, Kahn-Plotkin sequentiality and linearity acquire a new significance in this setting. In particular, using the ideas of Geometry of Interaction and Game Semantics, models of lambda-calculi can be built using only first-order domain theory.

John McLean
The Role of Semantics in Computer Security

Computer Security has long been seen as a a fertile ground for the application of formal methods. Security mistakes can cost a large amount of money and, at least in some cases, cost lives. However, since security properties differ from standard functional properties, it is not always clear how to apply standard formal techniques to verify system security. In particular, it is unclear whether a formalism correctly captures the property of interest and how much assurance is actually gained by formal proofs. One reason for this lack of certainty is that security formalisms tend to be entirely syntax based and, hence, lack the benefit of an independent validation metric. Formal semantics may be able to provide that independent metric.

Martin Hyland
Sequentiality

The notions of sequential algorithm and sequential function are usually discussed in terms of concrete mathematical models. In joint work with Cockett and Schalk, we take a more abstract approach; we describe (some) categories of 'sequentiality spaces' as subcategories of 'causality spaces'. We construct embeddings of various concrete models in causality spaces and hence 'test' for sequentiality; and we consider what further structure is needed to enable one to reconstruct more familiar models.

Security Session

Pasquale Malacaria and Chris Hankin
Security, Information Flow and Games

Security is an important issue in modern software systems. Due to the growing importance of network activity, personal data are sent around and a great effort is devoted to guarantee that these data do not fall in the wrong hands.

In the absence of higher order feautures, the basic notion of security is that once program variables are given a security clearance no user able to read values stored in variables with clearence s can gain any information about values stored in variables with a clearence higher than s. There are typically two ways to gain information about a variable:

- A direct way is through assignment: If we know the value of y in $x := E(y)$ then we can deduce the value of x (once we know what the expression E is). This situation is described in the literature by saying that there is a *direct flow* from y to x.
- Alternatively we can deduce partial information about x through predicates: For example in $\texttt{if}\ x > 0\ \texttt{then}\ y := 1\ \texttt{else}\ y := 2$. In this situation we can deduce information about x by knowing if y is 1 or 2. This is referred to in the literature by saying that there is an indirect flow from x to y.

A way to ensure then that security is achieved in a system is to demand that if there is a (direct/indirect) flow from x to y then the security of x is lower than the security of y.

Several approaches have been proposed to implement these ideas.

It is clear that, whatever approach one chooses, a minimal requirement for a security system should be that it should be proved to be correct. It would be otherwise difficult to trust such a system.

Many approaches which have clear correctness results are based on a type system security checker [7, 3] in which the security status of a program component is treated in a very similar way to type information in a typed language. In this setting a correctness result amounts to a soundness proof for the type system.

However the analogy between types and security is not completely convincing. A variable of type integer is expected to hold an integer value independently from the context it is in. This is to ensure the validity of operations over this variable and for memory allocation. However security is a much more contextual notion. Consider for example the following program:

$$P = x := y; y := z$$

where x is "low security" variable and z a"high security" variable. In [7] is not possible to certify the security of P, because y should be (because of z) a"high security" variable (and then we would breach security with $x := y$ or y should be (because of x) a "low security" variable (and then we would breach security with $y := z$. However it is rather obvious that if y hasn't been declared "high security" before the statement $x := y$ then there is no breach of security in P; the variable y has just become "high security" because of $y := z$ and whatever it has been in the past doesn't matter[1].

Higher order feautures introduce additional difficulties to the problem. In this setting one could indeed get information about secure data without using assignments. Consider the following example:

```
let lookup x l = if l=[] then false
                 else if x= hd(l) then true
                      else lookup x (tail l)
```

Here whoever uses this function can get information about the elements of the list. The natural requirement for a function f is hence that the security of the application of f to its arguments should be at least as high as the highest security of its arguments. So in the above example even if everybody could be authorised to use `lookup`, the result of `lookup a L` couldn't be read by someone not authorised to read `L`.

This approach to security is the one followed by [3] (and is based on similar ideas used by [5] in the framework of integrity). Both these approaches are however based on types and a rigidity similar to the one we have already noticed in the imperative case arises. Consider for example $(\lambda xy.x)AB$; here users are authorised to read the result if they have security clearance as high as the highest security clearance between A and B. It is clear however that it is only the security clearance of A that matters.

We believe hence that a more reasonable treatment of security should be in terms of the information flow of the program. Unlike types, information flow takes into account temporality so to make possible distinctions like the ones we were discussing above.

We will concentrate on Idealised Algol [6], a typed higher order functional language with imperative constants, i.e. an imperative extension of PCF. The algorithm we will use relies on the generalised flowchart of the program [4]. Generalised flowcharts are graphs derived from the game semantics [1] of a program and allow the derivation of information flow in a program. Classical flowcharts are a particular case of such graphs.

The game based approach to security has the following features:

- It is a particular application of a general framework for the static analysis of programs [4]. As a consequence it inherits many properties of this general framework:
    - Correctness of the security analysis based on games is a corollary of a much more general result about the relation of abstract games and operational semantics.

[1] Note that if P was embedded in a `while` loop then we would have a security breach because in a loop if A follows B then A also precedes B. However if P was embedded in a conditional there would be no security breach; this distinction between conditionals and while loops is not present in some information flow based analysis like [2]

  - The game approach works for a broad spectrum of programming languages with functional, imperative and class-oriented features.

- On the other hand, concurrency is at the moment yet outside the game theoretical approach. However there seems not to be a general agreement concerning what is an information flow for a concurrent language (see the discussion in [7])

A related issue to security is that of integrity. Here the point is to distinguish between data that can be trusted (as for example data received by a authorised client) and data that cannot be trusted [5]. It is natural to ask that an assignment $x := E(y)$ should be accepted if (at that particular point) y is trusted at least as x is and that we shouldn't have an untrusted variable as a guard for trusted ones. An information flow analyser is hence valid to certify the integrity of a program.

References

[1] Abramsky S., Jagadeesan R. and Malacaria P. Full abstraction for PCF (extended abstract). In *Proc. TACS'94*, LNCS 789, pp 1–15, Springer-Verlag, 1994.

[2] J. Banâtre, C. Bryce and D. Le Métayer, Compile-time Detection of Information Flow in Sequential Programs, in *Proc. of the European Symposium on Research in Computer Security*, Springer LNCS 875, 55–73, 1994.

[3] N. Heintze and J. G. Riecke, The SLam Calculus: Programming with Secrecy and Integrity, in *Proc. POPL'98*, ACM Press, 1998.

[4] P.Malacaria and C.Hankin, Generalised Flowcharts and Games, in Proceeding ICALP 98, LNCS.

[5] J. Palsberg and PØrbæk, Trust in the λ-calculus, in em Proc. of the 1995 Static Analysis Symposium, Springer LNCS 983, 1995.

[6] Reynolds J. C. Syntactic control of interference. In *Proc. POPL'78*, pp 39–46, ACM Press, 1978.

[7] D. Volpano, G. Smith and C. Irvine, *A Sound Type System for Secure Flow Analysis*, Journal of Computer Security, 4(3), 1996.

Steve Schneider
Proving security properties in CSP

Security protocols are tricky to construct correctly and verify, in part because of the complexity of the possible interactions between participants. The language of Communicating Sequential Processes is well-suited to the analysis of this kind of complexity, and the existing semantic models for CSP can provide a foundation for a security-oriented theory. This talk will be concerned with the use of CSP as a modelling tool to provide a basis for specification and for correctness proofs of security protocols.

Kim Wagner
About Cut-and-Choose

In many cryptographic protocols, in particular ones to do with anonymous electronic cash, a particular method called cut-and-choose is employed. The interesting aspect of this method is that it employs randomness and probability in a non-trivial way. This in turn makes reasoning about such protocols difficult, unless the reasoning is specific about probability. We try to identify core qualities of the cut-and-choose method eventually aiming to facilitate abstract reasoning about cut-and-choose and related notions.

Andy Gordon
Secure Mobile Ambients

Boundaries like houses, national frontiers, bank vaults, and wallets matter for the security of our physical activities. Similarly, boundaries like login prompts, directories, applet sandboxes, firewalls, and encryption matter for the security of our computational activities. We have formalised the notion of a secure boundary in a process calculus. This calculus, the ambient calculus, describes computations that may establish boundaries, may pass through them given the possession of suitable capabilities, and may tear them down, again given suitable capabilities. My talk will introduce the ambient calculus, and show how it can describe a variety of techniques for securing valuable computational resources. This is joint work with Luca Cardelli.

Catherine Meadows
The NRL Protocol Analyzer: An Overview

The NRL Protocol Analyzer is a prototype special-purpose verification tool, that has been developed for the analysis of cryptographic protocols that are used to authenticate principals and services and distribute keys in a network. In this paper we give an overview of the Analyzer, describing the theory behind it, how it works, and some of its achievements.

Bill Roscoe and R.S. Lazic
Using data independence in the automation of non-interference proofs

It has previously been shown how separability and noninterference properties of deterministic CSP processes can be characterised, and automatically checked, via the determinism of appropriately formulated abstractions of the process. The systems one wishes to check for these properties are frequently data independent in one or more types: for example a file system may well be data independent in both the types of files and file contents; it follows that we can hope to obtain more general automated verifications if we can find ways of checking them independently of which types are used. In this talk I will show how this can be done by extending the theory of data independence to encompass determinism checking and, in order to support the style of example typically found in this area, certain types of arrays (containing one data independent type and indexed by another), labellings such as predicate symbols, and constant symbols.

Landin Session

John Reynolds

Ian A. Mason and Carolyn Talcott
Feferman-Landin Logic

In a series of papers in the 1960s Landin [Lan64, Lan65, Lan66] proposed that programming languages should be consist of the lambda calculus augmented by primitives for manipulating data as well as aspects of a program's environment such as state, control, or communication. We will call such languages "Landinesque languages". In the 1970s and 1980s Feferman [Fef75, Fef79, Fef85] developed several theories for natural formalization of constructive mathematics. These theories treat functions as objects in a first-order setting. In the 1990s Mason, Talcott and their collaborators used Feferman's theories as a starting point for developing formalisms for reasoning about two specific Landinesque languages: IOCC [Tal92] and VTLoE [HMST95]. The Landinesque language of IOCC incorporated a control facility similar to Scheme's call/cc, while VTLOE incorporated ML-like reference cells.

Following Feferman's ideas, both formalisms are essentially first-order, two-layered theories. The lower layer is a theory of program equivalence. The upper layer is a theory of class membership with a general comprehension principle for defining classes. Many standard class constructions and data-types can be represented naturally including algebraic abstract data types, list-, record-, and function- type constructions. In addition classes can be constructed as minimal or maximal fixpoints, which in turn give rise to associated induction and co-induction principles.

In IOCC (Impredicative theory of Operations, Control, and Classes) programs using escape mechanisms can be derived from functional specifications and one can define and reason about streams and co-routines. Methods for constructing models of IOCC from semantic models of the underlying programming language were also given in [Tal92].

VTLoE (Variable Type Logic of Effects) extends the first layer of the two-layer theory to include program modalities to reason about effects. The key to being able to define satisfaction is the CIU theorem [MT91] — a form of context lemma giving an alternative characterization of operational equivalence. The language of VTLOE, in particular the program modalities, was developed to enable an elegant axiomatization of the reference primitives [Mas97].

More recently Talcott [Tal97] introduced a general framework for defining the semantics of Landinesque languages along with the notion of uniform semantics. A general form of the CIU theorem was shown to hold for languages with uniform semantics and general principles were developed for reasoning about program equivalence in languages with uniform semantics. For such languages we have the combined benefits of reduction calculi (modular axiomatisation), and operational equivalence (more equations). In addition we have a sound basis for defining satisfaction for VTL0E-style assertions.

In this talk we show how to combine all of these ideas in a logic for reasoning about Landinesque languages called Feferman-Landin logic (FLL for short). FLL generalizes VTLoE to Landinesque languages with uniform semantics. We define the syntax of the logic relative to a signature for primitive operations, for each signature a class of computational models and a notion of satisfaction. A basic set of logical axioms and proof rules will be given. It is future work to determine appropriate notions of completeness. We intend that the logic support both model based and proof based reasoning and plan to implement a proof systems that supports this mixed reasoning. In the talk we will give some examples of primitive operations and their axioms and illustrate the wide range of properties the can be expressed in this logic.

References

[Fef75] S. Feferman. A language and axioms for explicit mathematics. In *Algebra and Logic*, volume 450 of *Springer Lecture Notes in Mathematics*, pages 87–139. Springer Verlag, 1975.

[Fef79] S. Feferman. Constructive theories of functions and classes. In *Logic Colloquium '78*, pages 159–224. North-Holland, 1979.

[Fef85] S. Feferman. A theory of variable types. *Revista Colombiana de Matématicas*, 19:95–105, 1985.

[HMST95] F. Honsell, I. A. Mason, S. F. Smith, and C. L. Talcott. A Variable Typed Logic of Effects. *Information and Computation*, 119(1):55–90, 1995.

[Lan64] P. J. Landin. The mechanical evaluation of expressions. *Computer Journal*, 6:308–320, 1964.

[Lan65] P. J. Landin. A correspondence between Algol60 and Church's lambda notation. *Comm. ACM*, 8:89–101, 158–165, 1965.

[Lan66] P. J. Landin. The next 700 programming languages. *Comm. ACM*, 9:157–166, 1966.

[Mas97] I. A. Mason. A First Order Logic of Effects. *Theoretical Computer Science*, 185:277–318, 1997.

[MT91] I. A. Mason and C. L. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1:287–327, 1991.

[Tal92] C. L. Talcott. A theory for program and data type specification. *Theoretical Computer Science*, 104:129–159, 1992.

[Tal97] C. Talcott. Reasoning about functions with effects. In *Higher Order Operational Techniques in Semantics*. Cambridge University Press, 1997.

Olivier Danvy

The mechanical evaluation of higher-order expressions

We revisit Peter Landin's "mechanical evaluation of expressions" in the presence of higher-order abstract syntax.

Michel Mauny

Dana Scott
Logics of Types and Computation. Preliminary Report

Over the last year, I and my students have been developing a circle of ideas that came up in my 1996 graduate course on domain theory. The aim is to use type theory (and topos theory) via realizability to model a constructive logic which accomodates standard types (e.g. countably based topological spaces) and domain theory (as certain countably based T0-spaces), has extensive closure conditions (e.g. as a cartesian closed and locally cartesian closed category), and supports polymorphism (in the style of Girard and Reynolds or the calculus of constructions).

Similar approaches having some of these advantages have been suggested many times before, but we now think better results can be achieved using realizability over the graph model for the (untyped) lambda calculus. Moreover, in this proposed interpretation, there is a clear-cut definition of a type operator #A, meaning the formation of the computable elements of the type A (which includes computable functions as a type #(A - B)). This operator can also be applied to propositions providing a constructive version of an kind of S4 modal logic. In this way it is possible for example to work both with the whole set of real numbers along with the computable numbers and functions.

The major task of the project is to show that this logic is capable of naturally formulating the constructions and proofs required for the questions of semantics of programming languages – especially in justifying the existence and properties of recursively defined types. If this is successful, there should be significant conclusions for constructuve logic as well.

Continuation Session

Hayo Thielecke
A tutorial on Landin's 'generalization of jumps and labels' (33 years later)

We give a guided tour of Peter Landin's 1965 tech report "A generalization of jumps and labels" (which is soon to be reprinted), relating what may now seem archaic to modern concepts. Although continuations were not invented when Landin discovered and discussed his control operator **J**, a modern reader trying to understand control operators (the canonical one being `call/cc` in Scheme) still needs to understand many of the issues already discussed by Landin.

Although Landin did not have the general concept of continuation to guide him, he discovered a great deal of the pragmatics of `call/cc` more than ten years before this construct actually appeared in programming languages. He certainly knew better than some "folk theorems" about control that we still encounter more than 30 years later.

Paul Levy
Game Semantics as Continuation Passing

We show that the game semantics of Hyland, Ong and Nickau and related game semantics factorize via continuation-passing style (CPS) transforms. We provide a comparatively simple game semantics for a language $\Sigma\neg$-calculus that can be seen as the target of these transforms. This elucidates some key concepts of game semantics. We see that the role of *pointers between moves* is to indicate the time of receipt of the continuation being thrown to. *Arenas* are seen to be type normal forms in $\Sigma\neg$-calculus.

J. Laird
A Fully Abstract Semantics for higher-order control

We give a construction of a fully abstract semantics for the language (call-by-value) μPCF, which is PCF extended with the operations of naming and μ-abstraction of Parigot's $\lambda\mu$ calculus. This is equivalent to adding call-with-current-continuation to PCF, and we show that it can be modelled soundly in the Kleisli category of a continuations monad on Fam(C), where C is any Cartesian closed category. The fully abstract construction can be given at an axiomatic level, characterising the properties of the base category which are required for adequacy and completeness. Abramsky/Jagadeesan/Malacaria and Hyland/Ong games both provide concrete examples of such categories; in the latter case the semantics of μPCF can be considered as an extension of the Abramsky/McCusker model of call-by-value PCF, obtained by abandoning its 'bracketing condition'.

Carsten Führmann
Exponential categories and Semantics of Continuations

Our goal is to give denotational semantics for programming languages with higher order types and access to the control flow. Typical imperative languages allow modification of the control flow by mechanisms like goto's and block escapes, whereas functional languages use continuations, which are handled by operators like callcc and throw. We introduce exponential categories for modelling languages with higher order types and control operators, and we present and extended Kleisli construction by which expontial categories are equivalent to an earlier approach, namely tensor-not-categories (see [PT97]). Moreover, we give numerous substantially different characterisations of exponential categories, including one that involves self-adjoint functors, and one that describes exponential categories as certain full subcategories of cartesian closed categories; this multitude of representations suggests that exponential categories are elementary. From this and the equivalence to tensor-not-categories, we conclude that the resulting semantic framework has an appealing mathematical status.

[PT97] John Power and Hayo Thielecke, Environments, continuation semantics and indexed categories, in Proceedings TACS'97, LNCS1281, pages 391-414, Springer Verlag, 1997.

Comprox Session

Philipp Sunderhauf and Abbas Edalat
Computable Analysis using Continuous domains

We use basis ingredients of an effective theory of continuous domains as a foundation for computable analysis. Continuous domains provide a convenient framework to capture the notion of partial information.

We investigate computability on the real line via the interval domain, and on Banach spaces via the domain of formal balls. Our approach turns out to be equivalent to the well-established approach of (Pour-El & Richards '88).

Peter J. Potts
Efficient on-line computation of real functions using exact floating point

In this article, we develop the work of Vuillemin and Nielsen and Kornerup, and show that incrementality and efficiency can be simultaneously achieved in exact real arithmetic. We present a formal account of incremental digit representations born out of domain theory, which includes the redundant binary representation and continued fraction representation. The generalisation of both these representations leads to the notion of a general normal product constructed using Möbius transformations. We then examine a specialisation of general normal products called exact floating point with interesting mathematical properties. Real functions are captured by the composition of 2-dimensional Möbius transformations, leading to the notion of expression trees. Various reduction rules and a lazy form of information flow analysis is used to allow expression trees to be converted efficiently into the exact floating point representation. Algorithms for the basic arithmetic operations and the transcendental functions are presented using "redundant if" statements for range reduction and expression trees derived from the theory of continued fractions.

Reinhold Heckmann
Contractivity of Linear Fractional Transformations

One possible approach to exact real arithmetic is to use linear fractional transformations (LFT's) to represent real numbers and computations on real numbers. Recursive expressions built from LFT's play a prominent role in this approach. They are only convergent (i.e. denote a well-defined real number) if the involved LFT's are sufficiently contractive. In this paper, we define a notion of contractivity for LFT's. It is used for convergence theorems and for the analysis and improvement of algorithms for elementary functions.

Michal Konecny
Functions over Reals Represented by Streams of Moebius Transformations Computable by Finite Automatons

Recently Potts and Edalat have introduced a representation of real numbers by streams of digits out of a specific finite set of Moebius transformations. Using such a representation some real functions can be computed by an I/O automaton that continuously inputs digits of the arguments and outputs digits of the result. I've recently been tackling the question: Which real functions can be computed using only finite memory i.e. computable by a finite automaton ? As a first approximation I've been asking only: Which Moebius transformations are computable by a finite automaton ? The answers are dependent on the specific choice of transformations that serve as digits. I'd like to state theorems about computability of Moebius transformations themselves for the system of Potts and Edalat and give some partial results about the case of an arbitrary digit system. These aim at determining how good the present system is when compared to other ones, in particular whether there is a significantly better one.

M. Alvarez-Manilla, A. Edalat, N. Saheb-Djahromi
An extension result for continuous valuations

A valuation is a real-valued function defined on the lattice of open sets of a given topological space that is modular, monotone and strict. For domains, the problem of extending a (Scott) continuous valuation, i.e. one preserving directed sups, to a Borel measure appeared first in the context of probabilistic nondeterminism in the work of Saheb-Djahromi [SD80]; however, this proof of the extension given for ω-algebraic domains contained a gap. Norberg established the result for σ-finite valuations on ω-continuous domains and gave applications in random set theory and the proof of Daniell-Kolmogorov theorem. For ω-continous bounded complete domains the result followed from the work by Lawson [Law82] who showed that a continuous valuation defined on a distributive continuous lattice L has a unique extension to a regular Borel measure on L (in the Borel σ-algebra of its Lawson topology). Finally Jones and Plotkin [Jon89],[JP89] claimed the result for continuous domains, without presenting a correct proof.

Continous valuations have in recent years played a crucial role in the domain theoretic approach to classical measure theory which has led to a new generalization of the Riemann integral with applications in fractal geometry [Eda97].

Here we present a general result that includes the domain theoretic cases mentioned before. We show that if a σ-finite valuation on a directed complete partial order is the directed supremum of a family of simple valuations then it has a unique extension to a Borel measure (on the Borel σ-algebra of the Scott topology of the domain). As a corollary we have that every bounded and continuous valuation on a continuous domain can be extended uniquely to a Borel measure. This completes and concludes the work of extension of valuations to measures on domains since 1980.

[Eda97] A. Edalat. Domains for computation in mathemetics, physics and exact real arithmetic. Bulletin of Symbolic Logic, 1997. To appear.

[Jon89] C. Jones. Probabilistic Non-determinism. PhD thesis, University of Edinburgh, 1989.

[JP89] C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In Logic in Computer Science (LICS), pages 186-195. IEEE Computer Society Press, Silver Spring, MD, 1989.

[Law82] J.D. Lawson. Valuations in Continuous Lattices. In Rudolf-Eberhard Hoffman, editor, Continuous Lattices and Related Topics, Vol. 27 of Mathematik Arbeitspapiere. Universität Bremen, 1982.

[SD80] N. Saheb-Djahromi. CPO's of measures for nondeterminism. Theoretical Computer Science, 12:19-37, 1980.

Contributed Talks

Stephen Brookes
On the Kahn Principle and Fair Networks

The Kahn Principle for deterministic networks of asynchronous processes is well known. It asserts that each node can be viewed as computing a continuous function from input streams to output streams, and that these node functions can be characterized via the least fixed point of a functional whose definition reflects the mutually recursive patterns of interaction between the nodes. Non-deterministic networks arise naturally, bringing with them the need to tackle issues such as fairness which play no significant role in the deterministic setting. Several approaches have been proposed as generalizations of the Kahn Principle to non-deterministic networks, sometimes with constraints on the kind of non-determinism allowed. The prevailing tendency has been to try to produce a generalized Kahn Principle as close in spirit to the original: typically in these approaches a node is viewed as computing either a relation on streams (possibly equipped with some extra book-keeping structure) or as computing (one of) a set of continuous functions on streams. Also typically, such approaches tend to lack a completely satisfactory treatment of fairness. We argue that when considering non-deterministic networks, assumed to operate under a standard weak form of fairness, the computational intuition behind the original Kahn Principle no longer fits. This leads us to formulate a transition-trace semantics for non-deterministic networks, incorporating fairness directly. In essence, a trace set can be seen as a "stream-relation extended in time", and this concept seems to be a natural generalization of Kahn-style stream-functions. We discuss full abstraction issues, and connections with existing work. An advantage of our approach is that we have already shown how to use the underlying semantic model to interpret a shared-variable parallel language and a CSP-like language, so that we achieve a unifying framework for paradigms which have traditionally been modelled in mutually incompatible ways.

Michael A.Bukatin and Svetlana Yu. Shorina
Relaxed Metrics, Maximal Points, and Negative Information

In our earlier paper, we presented a method of obtaining partial and relaxed metrics from co-continuous and bi-continuous valuations for any continuous Scott domain A via formalism of $muInfo$-structures. Certain pathologies in the behavior of $I_x = \{y \in A \,|\, \{x, y\} \text{ is unbounded}\}$ precluded us from extending this method beyond bounded complete domains.

In this paper, we obtain meaningful Scott continuous relaxed metrics for continuous dcpo's by replacing I_x with $Neginfo(x) = Int(J_x)$, where $J_x = \{y \in A \,|\, x \in Int(I_y)\}$. This result can be understood in terms of interplay between *negation duality* and *Stone duality.*

This technique works especially well, when a continuous dcpo A satisfies the *Lawson condition* — the relative Scott and Lawson topologies on $Total(A)$ are equal. We obtain that the Lawson condition is equivalent to the property $\forall x \in Total(A).\, I_x = J_x$. This implies that when the Lawson condition holds, an induced metric on $Total(A)$ results.

Cristiano Calcagno
Two-level languages for program optimization

Two-level languages incorporate binding time information inside types, that is, whether a piece of code is completely known at compile-time, or needs some more inputs and can be evaluated only at run-time. We consider the use of 2-level languages in the framework of partial evaluation, and use a 2-level version of the simply typed lambda calculus with recursion. We give an operational semantics, an equational theory and a denotational semantics, that give an account of the distinction between compilation and execution phases. An adequacy theorem is given to relate the two semantics, showing in particular how they agree on non-termination at compile time. New models are considered based on other approaches to the semantics of 2-level languages in the literature.

Olivier Danvy and Kristoffer Rose
Type-Directed Partial Evaluation by Higher-Order Rewriting

Normalization by Evaluation (a.k.a. Reduction-Free Normalization) is an intriguing normalization method for the simply typed lambda-calculus, which has proven non-trivial to formalize [Berger and Schwichtenberg, LICS'91; Berger, TLCA'93; Altenkirch, Hofmann and Streicher, CTCS'95; Altenkirch, Hofmann and Streicher, LICS'96; Cubric, Dybjer and Scott, MSCS'97]. Inspired by our earlier work on Higher-Order Rewriting and Partial Evaluation [Danvy and Rose, RTA'98], we phrase the definitional equations of Normalization by Evaluation as a Combinatory Reduction System.

Type-Directed Partial Evaluation is a call-by-value variant of Normalization by Evaluation which has been implemented and used on non-trivial examples, but has not been formalized yet [Danvy, POPL'96]. We phrase it as a variant of the Combinatory Reduction System above, using explicit substitutions.

Alessandra Di Pierro and Herbert Wiklicky
An Operator Algebraic Semantics for Concurrent Constraint Programming

A distinctive feature of both Concurrent Constraint Programming (CCP) and its extension Probabilistic Concurrent Constraint Programming (PCCP), introduced by the authors in previous papers, is the clear separation between information order and the program execution. This makes it possible to define a denotational semantics avoiding the problem of solving domain equations.

In this work we investigate the use of mathematical structures which are unconventional in semantics. More precisely, we look at the free vector space on the underlying (countable) set of constraints, i.e. the set of (formal) linear combinations of constraints. This space can be given a topology in a natural way by introducing a norm. As a result we are lead to consider Banach spaces. In this setting we first define a semantics of PCCP: The domain of denotations will be given as an operator algebra, namely a set of linear operators on the Banach space. In the finite case we can represent these operators simply by matrices.

The semantics is then given via an inductively defined fixed point operator Ψ on this operator (matrix) algebra. The definition of Ψ encodes the structure of the underlying constraint system in an elegant way. The existence of a fixed point can be guaranteed by exploiting a well-known theorem from functional analysis. A concrete construction is also presented for which the resulting semantics is equivalent to a notion of observables corresponding to the exact results. Finally, we show that various domains used in conventional approaches towards the semantics of CCP (e.g. powerdomain, metric, etc.) are embedded in this framework.

Brian Dunphy, Peter O'Hearn and Uday Reddy
Linearity and logical relations

In this work, we address the formulation of logical relations for domain models of linear lambda calculi with recursion. It turns out that the action of the tensor product operator on relations is subtle and requires a careful analysis. We consider the definitions of logical relations for pointed cpo's with strict functions (for modelling relevant lambda calculus) and additive functions (for linear lambda calculus). We mention the results that have been obtained as well as some open problems.

Abbas Edalat
Bisimulation and Semi-pullbacks in Categories of Markov Processes

We say that a category has semi-pullbacks if for any pair of morphisms f:A-¿C and g:B-¿C there exists an object U and morphisms h:U-¿A and k:U-¿B such that h;f=k;g.

We give a canonical construction for semi-pullbacks in the category whose objects are families of Markov processes, with given transition probabilities, on Polish spaces, i.e. topologically complete separable metrizable spaces, and whose morphisms are transition-probability preserving surjective continuous maps. Our construction gives semi-pullbacks for various full subcategories, including that of Markov processes on locally compact second countable spaces and also in the larger category where an object is a family of Markov processes on an analytic space (continuous image of a Polish space into another Polish space) and a morphism is a transition-probability preserving surjective Borel map. It also applies to the corresponding categories of ultrametric spaces. Finally, our result also holds in the larger categories with Markov processes which are given by subprobability distributions, i.e. the total probability of transition from a state can be strictly less than one.

In order to properly state and deduce this result, we go beyond these categories, which are similar to the category considered by Giry, to a more foundational framework in probability theory and stochastic processes.

An immediate consequence of this result is that probabilistic bisimulation for labelled Markov processes on all these categories, defined in terms of spans of morphisms in the spirit of the work of Joyal, Nielsen and Winskel, is an equivalence relation; this generalizes the work of Larsen and Skou on discrete labelled probabilistic processes to continuous processes (joint work with Blute, Desharnais and Panangaden). Furthermore, one can obtain a logical characterization of probabilistic bisimulation (joint work with Desharnais and Panangaden).

Adam Eppendahl
Fixed Point Objects Corresponding to Freyd Algebras

A right adjoint R carries any Freyd LR-algebra for LR to a fixed point object. We note that R also carries any Freyd LTR-algebra to a fixed point object for T, where T is any endofunctor. We obtain an account of indexed fixed points by taking multiplication for T. This leads to good 2-categories of compact functors.

M.H. Escardo and R. Flagg
Semantic domains, injective spaces and monads

In applications of domain theory to denotational semantics, one starts by implicitly or explicitly embedding given topological spaces into appropriate domains endowed with the Scott topology. Given embeddings j:X-¿D and k:Y-¿E of this kind, one models continuous maps X-¿Y by Scott continuous maps D-¿E. It is thus natural to demand that the latter fully capture the former. Technically, one wants E to be injective over the embedding j:X-¿D.

Dana Scott (1972) characterized the continuous lattices endowed with the Scott topology precisely as the spaces which are injective over all embeddings. From another perspective, Alan Day (1975) and Oswald Wyler (1976) characterized the continuous lattices as the algebras of a certain filter monad on the category of topological spaces. Later, Scott characterized the continuous Scott domains as the injective spaces over dense embeddings, and Wyler characterized them as the algebras of the proper filter monad.

It turns out that Scott's results actually follow from Day's and Wyler's. One uses the fact that the filter monad is of the Kock-Zoberlein type. In any poset-enriched category with such a monad structure, the injective objects over a certain class of embeddings defined in terms of the monad structure are precisely the algebras (Escardo 1997). For the filter monad the embeddings are exactly the topological embeddings, and for the proper filter monad they are exactly the dense topological embeddings.

We apply this technique to obtain similar characterizations by injectivity of several other notions of semantic domain, including L-domains, continuous meet-semilattices, continuous join-semilattices, stably compact spaces and sober spaces.

Lisbeth Fajstrup, Eric Goubault and Martin Raussen
Directed Homotopy and Concurrency

In this talk we would like to introduce some recent concepts and results linking concurrency theory with algebraic topology. The origin of the work was the seminal papers by Vaughan Pratt[Pra91] and Rob van Glabbeek[vG91] where the standard automata-theoretic framework was extended to distinguish non-determinism from concurrency. This was done by introducing higher-dimensional transitions, whose dimension represents the number of processes that are acting concurrently. The geometry of the higher-dimensional transition system is very much related to interesting computer scientific properties, like deadlocks, serialization, mutual exclusion etc. as already hinted in papers by Jeremy Gunawardena [Gun94] and Eric Goubault [Gou95].

Since then, a collaboration took place between a computer scientist (Eric Goubault) and two mathematicians (Lisbeth Fajstrup and Martin Raussen) so that the mathematical foundations of this geometric concurrency theory are secured and with the objective to use the algebraic topological apparatus to solve more efficiently some concurrency problems. A paper and an implementation of a deadlock detection algorithm ([FGR96] and see http://www.dmi.ens.fr/~goubault) has been a first step. It appears now that using the geometric formulation, concurrency theory brings new problems to algebraic topology.

Since time introduces a direction, the geometric models come equipped with a partial order. Executions correspond to directed paths respecting this partial order. Directed paths which can be deformed into each other by a family of directed paths correspond to equivalent executions. While deformations of paths, i.e., homotopy theory is well-studied, the introduction of a partial order introduces new phenomena and problems. It appears that several notions from general relativity theory fit into the framework.

It is the objective of the talk to explain these problems and what we currently know about them.

References

[FGR96] L. Fajstrup, E. Goubault, and M. Raussen. Detecting deadlocks in concurrent systems. Technical report, Submitted, also available at http://www.dmi.ens.fr/ goubault, 1996.

[Gou95] E. Goubault. Schedulers as abstract interpretations of HDA. In *Proc. of PEPM'95*, La Jolla, June 1995. ACM Press, also available at http://www.ens.fr/~goubault.

[Gun94] J. Gunawardena. Homotopy and concurrency. In *Bulletin of the EATCS*, number 54, pages 184–193, October 1994.

[Pra91] V. Pratt. Modeling concurrency with geometry. In *Proc. of the 18th ACM Symposium on Principles of Programming Languages*. ACM Press, 1991.

[vG91] R. van Glabbeek. Bisimulation semantics for higher dimensional automata. Technical report, Stanford University, Manuscript available on the web as http://theory.stanford.edu/~rvg/hda, 1991.

Paul C Gilmore
An impredicative simple theory of types

By the theory TT is meant a weakening of Church's formulation of the simple theory of types in which all non-logical axioms have been dropped. The theory ITT described here is an impredicative version of TT. The types of ITT are exactly the types of TT. But the type membership of the individuals is extended by permitting a term of any type to be an individual provided only individual variables have a free occurrence in the term. The effect of this extension is to make the domain of the individual variables the equivalence classes of a typed lambda calculus.

This extension of TT is motivated by a nominalist view of universals in which a predicate of a universal is interpreted as a predicate of a name of the universal. For example, a nominalist interpretation of "Yellow is a colour" is "Yellow is a colour-word"; the sentence is understood to describe how the word 'Yellow' is used in English. This nominalist view requires a careful distinction between the use and mention of predicate names, especially when treating abstraction and quantification. Since computers are consumate nominalists, a nominalist interpretation of computer languages is not unnatural.

The semantics and proof theory of TT are described. The semantic proof of the consistency of the full simple theory of types is adapted for ITT. A blending of the lambda calculus definitions of TT with definitions in the style of set theory is permitted in ITT. As a consequence both wellfounded and non-wellfounded recursive predicates can receive direct definitions as terms in ITT, rather than as fixed points of a lambda calculus identity. This will be illustrated with several examples.

Hanne Gottliebsen
Converting Between Rewriting Systems and Natural Semantics

Rewriting in combinatory reduction systems involves searching, not only for a reduction rule to use, but also for a subterm to apply this rule to. Whereas this is not a difficult task it is time consuming. With natural semantics a term is always evaluated from the outside and in, so although searching must be carried out to determine which semantic rule to apply, there is no need to search through the subterms (other than what is necessary for determining the rule). Presented here is a method to convert combinatory reduction systems into natural semantics, such that evaluation in the natural semantics corresponds in some close way to reduction in the combinatory reduction system.

In this talk we shall see how one can construct natural semantics equivalent to some combinatory reduction system such that any reduction sequence corresponds to some proof tree built from the natural semantics and vice versa. The natural semantics is constructed by simple syntactic manipulations of the terms and rules of the reduction system. The proof of the close correspondence between the two is constructive, thus enabling us to build the proof tree corresponding to some concrete reduction sequence, or indeed build a reduction sequence corresponding to some proof tree.

An interesting subset of the combinatory reduction systems is the term rewrite systems. As well as treating term rewrite systems as simply a special case of combinatory reduction systems, we shall also present a stronger result. For term rewrite systems which are weakly normalising and without critical pairs we will construct natural semantics equivalent to the rewrite system such that each reduction to normal form correspond to some proof tree.

Matthew Hennessy and James Riely
Types and Capabilities for Mobile Agents

A language called Dpi is presented for describing "mobile agents". Both the syntax and reduction semantics is based on that of the picalculus in that communication is channel based, and these may be created dynamically and shared privately among agents. However in Dpi channels reside at explicit locations or sites and communication is local to sites. Moreover agents can migrate between sites to effect computations.

We describe a typing system for Dpi based on the novel notion of a location type. This associates with each location a set of *capabilities*, i.e. a set of permissions to perform certain behaviour. These capabilities include the ability to send or receive values, of specified types, on named channels.

We prove a Subject Reduction theorem and a Type Safety theorem. The latter is expressed formally in terms of the knowledge of an agent; well-typed agents can only perform actions which are consistent with their knowledge of their environment.

Samin Ishtiaq and David Pym
Kripke Resource models of the lL-calculus

We describe a Kripke resource semantics of the $\lambda\Lambda_\kappa$-calculus, a system of first-order linear dependent function types which uses a function κ to describe the degree of sharing of variables between functions and their arguments. The models, a set of Kripke functors, each indexed by an element of a "resource" monoid, can be seen as a dependent or indexed formulation of Pym's and O'Hearn's work on "bunches".

Peter Johnstone, John Power, Toru Tsujishita, Hiroshi Watanabe and James Worrell
On near-toposes of coalgebras

There has been considerable interest in the study of categories of (labelled) transition systems as categories of coalgebras for suitable endofunctors F on the category of sets. Typically, the functors F do not preserve pullbacks, but they come fairly close to it: for example, they tend to preserve pullbacks of monomorphisms. We investigate what can be said in general, under such hypotheses, about the inheritance of good properties from a category $\mathcal{E}$ by the category $\mathcal{E}_F$ of coalgebras for an endofunctor $F : \mathcal{E} \to \mathcal{E}$. In particular, we show that, provided 'final coalgebras exist' in a suitable sense, $\mathcal{E}_F$ inherits (finite) completeness, (finite) disjoint pullback-stable coproducts and a subobject classifier from $\mathcal{E}$ if F preserves pullbacks of monomorphisms. If F satisfies the stronger condition of preserving weak pullbacks, then $\mathcal{E}_F$ inherits regularity from $\mathcal{E}$; in particular, if $\mathcal{E}$ is a Grothendieck topos then $\mathcal{E}_F$ embeds as a reflective full subcategory of a topos. However, we also show that, if F admits a certain coproduct decomposition, then $\mathcal{E}_F$ cannot be a topos unless F actually preserves pullbacks. These results extend earlier work of Power and Watanabe, Worrell, and Carboni and Johnstone.

Jürgen Koslowski (joint work with J. Robin B. Cockett)
Interaction and Games via the generalized Chu-Construction

The proper generalization of the Chu-construction to the setting of closed bicategories takes arbitrary 1-cells as objects and so-called "Chu-cells" as 1-cells, resulting in another closed bicategory. Two important variants arise by considering cyclic Chu-cells (then the objects become endo-1-cells) or chains of Chu-cells.

Specializing this construction to (the suspension of) the cartesian closed category **set**, Chu-cells between sets A and B can be viewed as complete labelled "tri-partite" graphs: there are three sets of nodes F_0, F_1 and F_2 such that all edges between F_0 and F_1 exist and have labels in A, and all edges between F_1 and F_2 exist and have labels in B. If completeness is not desired, one can take the category of sets and partial functions with a suitable monoidal closed structure instead.

In the cyclic case (i.e., $F_0 = F_2$), these graphs may be interpreted as games with fixed states for two "players", alternately making moves from A and B, respectively. These subsume the familiar tree-based games. In the case of chains, the states can be thought of as "evolving" in time. The coproducts present in the base category can then be used to mimic constructions known from tree-based games.

Philipp Kutter
Montages - "Pop Semantics"

Over the last two decades, research in formal semantics followed the trend towards higher levels of specialization, at the expense of computational clarity and applicability in language development. Somehow it seems that the people involved in semantics research were not aware of the needs of potential users of semantics, i.e. language designers. Often semantics research focused mainly on logical frameworks for performing some kind of deductive reasoning. It is striking how most of the existing languages are specified informally and even when a formal specification is undertaken, "semanticists play the role of morticians in the programming language community"[Schmidt97], e.g. analyzing an already designed type-system.

Designing a programming language is a complex activity which takes from two to five years [Watt96], and which involves a broad range of professionals. These people are surely experts in designing languages and they already know the semantics of that domain: all is requested is to provide a notation to express semantics conveniently. A formalism specifically tailored for describing programming languages (possibly a subclass of them) and which could be taught in a reasonable time. In essence, the major goal must shift from the mathematical beauty to the pragmatics of language design provided that rigour and formality are assured.

In order to make the language designer reconsider their attitude to formal methods, we proposed a formalism, called Montages [KuPi97]. The main aim of such a mathematical framework is to adopt a "user-friendly" interface to the underlying mathematical machinery. By mean of visual descriptions the designer can map her/his intuitions to semantical definitions, which can be fed to a rapid prototyping tool, called Gem-Mex. Montages are based on context free grammars, first order logic, Gurevitch's abstract state machines, and a novel visual language for the specification of the mapping from concrete to abstract syntax and for static analysis. In the talk Montages are presented, new developments of the formalism and examples from the Java specification are shown and the tool Gem-Mex is demonstrated.

[Schmidt97]
D.A. Schmidt.
"On the Need for a Popular Formal Semantics"
ACM SIGPLAN Notices, vol.32, #1, p.115-116, 1997.

http://www.cis.ksu.edu/~schmidt/papers/boston3.ps.Z

[Watt96]
A.D. Watt.
"Why don't Programming Language Designers use Formal Methods?"
Proc. SEMISH96, p.1-16, University of Pernambuco, Recife, Brazil, 1996.
http://www.dcs.gla.ac.uk/~daw/publications/SEMISH.ps

[KuPi97]
P.W. Kutter, A. Pierantonio.
"Montages: Specifications of Realistic Programming Languages"
Journal of Universal Computer Science, Springer Verlag,
vol. 3, #5, p. 416-442, 1997.
http://www.iicm.edu/jucs_3_5/montages_specifications

Ranko Lazic and Bill Roscoe
A Semantic Study of Data Independence with Applications to Model Checking

In practice, processes are frequently parameterised by data types. For example, communication protocols can transmit values of any given type.

If P(X) is such a process, and A and B are two types, what is the relationship between the semantic values of P(A) and P(B)? We answer this question by extending the basic theory of logical relations to cover process types, so that relations between A and B can be lifted to relations between semantic domains of processes.

The resulting theory, together with symbolic operational semantics and factoring out of symmetry, can be applied to the problem of uniform verification of data independent processes (which are a very important subclass of type parameterised processes). In other words, it yields procedures which, on the basis of a single run of a model checker, can establish whether P(A) satisfies a given specification for all A.

Keye Martin
The computation of real analytic mappings

We introduce the domain $\mathbb{P}^\infty[a,b]$ — an ω-continuous domain which provides a model for the computation of real analytic mappings on an interval $[a,b]$. Imagine trying to describe the way we compute the exponential map in a domain-theoretic fashion. Ideally, we would like to write down an increasing sequence

$$1 \sqsubseteq 1+x \sqsubseteq 1+x+\frac{x^2}{2!} \sqsubseteq ... \sqsubseteq 1+x+...+\frac{x^n}{n!} \sqsubseteq ...$$

whose supremum is

$$\bigsqcup(1+x+...+\frac{x^n}{n!}) = e^x$$

Implicitly, the information order $\sqsubseteq$ must respect two equally important rules:

(i) Computing more terms in a series means more information.

(ii) The sequence of terms must converge uniformly to a continuous map.

The trend in Domain theory traditionally has been that algebraic domains deal with things like (i), and that continuous (non-algebraic) domains address issues like (ii). However, the only way to completely and satisfactorily describe this situation is to combine the continuous and algebraic intuitions mentioned above. This is the problem that $\mathbb{P}^\infty[a,b]$ solves. In the end, we obtain an ω-continuous domain (with a measurement) which has a countable basis specified by the set of polynomials with rational coefficients, and whose maximal elements can be identified with the absolutely convergent power series on the interval $[a,b]$. The model has a number of interesting properties and applications that we will discuss as time permits.

Michael Marz
A Fully Abstract Model for Sequential Computation

In their LICS97 paper, J. Riecke and A. Sandholm constructed a domain theoretic model for FPC, a call-by-value language containing recursive types and sum types. I will generalise their approach by presenting a cartesian closed category SD of domains that contains Riecke's and Sandholm's model as a subcategory.

Next, I shall present a universal language SFL (sequential functional language) containing recursive types and various kinds of product types, sum types and function types. This language can be used as a metalanguage for different sequential languages, in particular call-by-value and call-by-name ones.

Finally, I will show that the category SD provides a fully abstract model for SFL.

Mike Mislove
A Truly Concurrent Semantics Using Resource Traces

Trace theory originated in the work of Mazurkiewicz as a model for Petri nets. Starting with a finite set of actions on which is defined a reflexive, symmetric relation, called the dependency relation, a concatenation operation is defined on strings of actions so that two actions commute with one another precisely when they are independent - i.e., when they are not dependent. The resulting structure can be made into a monoid whose identity is the empty trace by taking the monoid of strings modulo the congruence relation generated by the complement of the dependency relation. However, the concatentation operation on this monoid is not monotone with respect to the prefix order. Recently Gastin and Teodesiu have provided a model for this concatentation operator using the notion of resource traces. This model supports concatenation of both finite and infinite traces, and the model is a prime algebraic domain in which the concatenation operation is Lawson continuous. In this talk I will present a semantics for a simple language supporting a weak sequential operator which replaces sequential composition, a restriction operator which restricts processes to a given set of resources, a parallel composition operator and recursion. The denotational model for the language is the resource traces model for concatenation devised by Gastin and Teodesiu, and the meaning of the weak sequential composition of process is the concatenation of the meanings of the processes as resource traces. I also will describe an operational semantics for this language which it is congruent to the denotational model. The language has some interesting features: it is deterministic, so the denotation of a program is a single resource trace, and it is fair. For example, if a and b are independent actions, then the meaning of the process (rec x . a*x)*(rec y. b*y) is the same as that of rec x . a*b*x, where "*" denotes weak sequential composition. This is joint work with Paul Gastin (LIAFA, Paris VII).

Peter Mosses
Action Semantics and Monad Transformers

Action-semantic descriptions (not to be confused with action systems, action calculi, etc.) were proposed by D.A.Watt and the speaker about 10 years ago, with the primary aim of improving the pragmatic aspects of (compositional) semantic descriptions. The crucial feature is that the combinators and primitives of action notation (in contrast to λ-notation) all generalize smoothly when further kinds of information processing capabilities are added to actions (e.g., adding imperative updates or concurrency to pure functional actions). The full action notation is defined by a fixed, small-step operational semantics (and a notion of testing equivalence). Although action semantics has been found to be well-suited for describing a wide range of programming languages, the inherent difficulty of extending action notation with new concepts is an evident drawback.

Monad transformers were proposed originally by E.Moggi with similar aims to action semantics, the lifting of operations through transformers corresponding roughly to the generalization of action combinators and primitives. This approach has since been applied by P.Hudak and S.Liang in their modular monadic semantics framework. The main difference from action semantics is that monads are based firmly in the realm of higher-order functions, rather than first-order transition relations. It is also relatively easy to add new monad transformers to a modular monadic semantics.

Last year, K.Wansbrough completed a detailed investigation of the relationship between action semantics and monad transformers, providing a modular monadic semantics for most of action notation and demonstrating how this may be extended, e.g., with continuations. However, this meant a definite break with the original operational semantics of action notation.

This talk considers the possibility of finding constructs that would do the same for small-step operational semantics as monad tranformers do for higher-order functional semantics.

After all, there are some well-known techniques for modelling a wide range of constructs in operational semantics, just as in denotational semantics; but operational semantics does not have inherently-good modularity. Could these techniques be regarded as instances of some more general construction? If so, one might be able to exploit it to give a modular *operational* semantics for action notation, and thus provide a hybrid framework that combines the extensibility of Wansbrough's modular monadic action semantics with the generality and straightforwardness of first-order operational semantics.

Cyrus Nourani

J. R. Otto
The LCC-FS Categories by Collapsing Resolutions

We define bicategories of resolutions reasonably related to type theory and logic programming, as well as categories of fractions and sheaf theory, collapsing by identifying 1-cells connected by cospans of 2-cells to the small locally cartesian closed categories with disjoint finite sums.

Uday Reddy
Parametricity and naturality in the semantics of Algol

The original semantics of Idealized Algol was given in a functor category, where the morphisms are natural transformations. When the role of parametricity in these morphisms was recognized, parametricity got added as an additional condition. The separate existence of parametricity and naturality as uniformity conditions is mildly disturbing because both of them are meant to capture the same intuition of uniformity. In this talk, I formulate a general categorical framework where parametricity subsumes naturality. I also describe models of Algol where such subsumption takes place. This paves the way for combining parametricity with non-interference reasoning as in Specification Logic.

Pedro Resende
On observing independence and causality

We generalize previous work on quantales and process semantics by showing that quantales can be seen as algebras of finite observations for true-concurrency. The basic idea is that true-concurrency arises whenever observers are able to detect independence or causality between (some) observations. An example of application to event structures is given, yielding several linear-time but non-interleaving equivalences that we compare to pomset equivalence. Algebraic properties of quantales with observations of independence and causality are studied, and so is a related notion of concurrent observation that generalizes pomsets and pomset ideals.

Giuseppe Rosolini and Paul Taylor
A Model of Synthetic Domain Theory based on the Effective Topos which satifies Stone Duality

In 1991 Hyland and Taylor proposed an axiomatisation of synthetic domain theory as a certain internally reflective subcategory of a topos, namely the smallest which contains the object Σ of computable predicates. Its objects, the "predomains", are called replete and satisfy Leibniz Principles on elements and types. Taylor later proposed a further Leibniz Principle, on categories, which might also be called a Stone Duality: the dual of the category of predomains (its category of "frames") should be *monadic*, ie form the whole of a category of algebras.

It is not difficult to find categories with the monadic condition if no ambient topos is required, for example the category of locally compact locales. To reconcile the two axiomatisations of synthetic domain theory we want to use the Yoneda embedding to construct a topos. This requires the category of predomains to be small and complete, so we must abandon topological models in favour of the work on Hyland, Robinson and Rosolini, who provided ain internal category of the effective topos which is weakly equivalent to a complete one. This turns out to be enough.

We do not know whether the effective topos itself already has the required property: our claim is simply that *some* topos does.

James S. Royer
A Type-Theoretic Characterization of the Type-2 Basic Feasible Functionals

The type-2 Basic Feasible Functionals (abbreviated, BFF_2) have proven to be the most robust of the various proposals for a "natural" type-2 analog of the polynomial-time computable functions. One problem with $BFF_2$is that all of characterizations of it seem rather ugly or ad hoc. In this talk I explain a characterization ofBFF_2 based on type-theoretic constraints. The characterization is inspired by type-theoretic characterizations of type-1 polynomial-time by Bellantoni and Cook and by Leivant. While the systems of Bellantoni and Cook and of Leviant are strictly predicative, it seems necessary to partially break predicativity under our general approach. Also, in contrast to the prior programming formalisms characterizing BFF_2, our formalism is very close in spirit to Kapron and Cook's machine-based characterization of BFF_2.

(This comes out of joint work with Robert Irwin and Bruce Kapron.)

Andrei Sabelfeld
Forwards and backwards analysis for functional programs

We present an approach to static analysis of functional programs. We describe two general frameworks of semantic property analysis for recursive programs and present a marking technique for solving a rather powerful subset of these problems. Illustrative examples that are captured by this technique are *strictness analysis* and *needed/unneeded analysis*. However, some static property analyses of recursive programs cannot be formulated as either forwards or backwards analysis problems. As an instance, we consider the *formal parameter dependence analysis.*

M. P. Schellekens
Weightable Directed Spaces

Partial metrics, or the equivalent weightable quasi-metrics, have been introduced by Matthews as part of the study of denotational semantics of data flow networks. It is remarked in work by Bukatin and Shorina that "the existence of deep connections between partial metrics and valuations is well known in Domain Theory". Examples of such connections have been discussed by O'Neil, Bukatin, Scott and Shorina, where in each case partial metrics are generated from positive valuations. The interest in (e)valuations in connection to Domain Theory derives from work by Edalat, Heckmann, Jones and Plotkin. We analyze the precise connections between partial metrics and valuations.

Since domain theoretic applications typically involve semilattices rather than lattices, the question arises whether the fruitful notion of a valuation can be generalized to this context. We extend our prior work on weightable directed spaces and focus on the class of quasi-uniform semilattices in this context. Quasi-uniform semilattices arise naturally in Domain Theory and include in particular Smyth's totally bounded Scott domains, Matthews's Baire partial metric spaces as well as the complexity spaces.

Partial metric spaces and positive "semivaluations" are shown to be dual notions in this context. This involves the solution of an open problem from Kuenzi's survey paper "Nonsymmetric Topology" for the class of quasi-uniform semilattices. Each of the above mentioned examples is shown to correspond dually to a (semi)valuation space. We conclude that, in the context of Domain Theory, partial metrics correspond to generalized (e)valuations.

Wolfram Schulte
A Programmer Friendly Modular Definition of the Semantics of Java

We propose in this paper a definition of the semantics of Java programs which can be used as a basis for the standardization of the language and of its implementation on the Java Virtual Machine. The definition provides a system and machine independent view of the language as it is seen by the Java programmer. It takes care to *directly* reflect the description in the Java language reference manual so that the basic design decisions can be checked by standardizers and implementors against a mathematical model.

Our definition is the basis for a related definition we give in a sequel to this talk for the implementation of Java on the Java Virtual Machine.

joint work with Egon Börger Università di Pisa, Dipartimento di Informatica, I-56125 Pisa, Italy boerger@di.unipi.it

Peter Selinger
An induction principle for traced monoidal categories

In 1996, Joyal, Street, and Verity have introduced the notion of an abstract trace on a symmetric monoidal category. Abstract traces arise in many different contexts in mathematics, and they capture intuitions that are as varied as Markov traces, braid closure, and existential quantification. In computer science, the prevalent intuition of trace is that of a "feedback operator" - a way of introducing loops in compositional networks such as dataflow networks or flowcharts. But it seems that some examples of traced monoidal categories fit the "feedback" intuition more than others. Are there special properties that "feedback-like" trace operators satisfy?

In this talk, I will propose such a property. Based on the intuitive idea that looping, as an iterative notion, should have an induction principle to go along with it, I will introduce an Induction Axiom for traced monoidal categories. The leading examples suggest that this axiom indeed captures the iterative content: it holds in many standard traced monoidal categories that carry a a "feedback" intuition, such as CPO, (REL,+), or various categories given by labeled transition systems - but it does not hold in categories such as (REL,x) where trace is essentially given by existential quantification. The induction axiom gives rise to an interesting proof principle, and it has some non-trivial consequences. For instance, it implies that determinism (in the sense of Jeffrey (1998)) is preserved under trace.

Dieter Spreen
Solving Domain Equations in the Category of Effectively Given Topological Spaces

The solvability of domain equations is an important requirement for a category of spaces used in giving mathematical meaning to programming language constructs.As is is well known several categories of cpo's as well as of metric spaces share this property. In this talk it is shown that the same holds for the category of effectively given topological spaces.

David Wragg, Sophia Drossopoulou and Susan Eisenbach
Java Binary Compatibility is Almost Correct

The Java language description is unusual in that it defines the effect of interleaving separate compilation and source code modifications. In Java, certain source code modifications, such as adding a method to a class, are defined as binary compatible. The Java language description does not require the re-compilation of programs importing classes or interfaces which were modified in binary compatible ways, and it claims that successful linking and execution of the altered program is guaranteed. In this paper we show that Java binary compatibility does not actually guarantee successful linking and execution. We then suggest a framework in which we formalize the requirement of safe linking and execution without re-compilation and we propose a more modest definition of binary compatibility. We prove for a substantial subset of Java, that our definition guarantees safe linking and execution.

Nobuko Yoshida
Recent results on concurrent combinators for mobile processes

In this talk, we discuss resent results on concurrent combinators for asynchronous pi-calculus. The asynchronous pi-calculus is a succinct "minimum meaningful" fragment of pi-calculus, which can express versatile computational structures, including all computable functions, lambda-calculus, Lafont's typed Interaction Nets, concurrent objects and languages.

A system of concurrent combinators of pi-calculus is small proper subset of this pi-calculus which can nevertheless represent the whole world of the original calculus up to weak bisimilarity. In this talk I introduce 7 basic atoms and show that they have a capability to represent any pi-prefix over arbitrary terms in a behaviourally correct way with a new quick proof, just like lambda-prefix can be decomposed into SKI-combinators. Then I prove what would correspond to "combinatory completeness" in the universe of dyadic interactions. Finally I also mention how we can use concurrent combinators to analyse expressive powers of various variants of pi-calculi.

Zhe Yang
Encoding Types in ML

A Damas-Milner type system such as ML's seems to prohibit type-indexed values, i.e., functions that map a family of types to a family of values. Such functions generally perform case analysis on the input types and return values of possibly different types. The goal of our work is to demonstrate how to program with type-indexed values within a Damas-Milner type system.

Our approach is to recursively interpret the input type as the value at this type index. This solution is type-safe, in the sense that the ML type system statically prevents any mismatch between the input type and function arguments depending on this type.

Such specific type interpretations, however, prevent us from composing different functions sharing the same type, but interpreted differently. We then focus on finding a universal type encoding that can be shared by different function definitions. We propose and compare several solutions: one requires first-class polymorphism and thus fails for ML; another employs higher-order functors of SML/NJ, which is elegant but clumsy; the last one, through injection/projection, seems to be more practical.

We also demonstrate the usefulness of such type-indexed functions through examples including type-directed partial evaluation, 'polytypic' programming, printf-like formatting, and subtype coercions. We discuss the tradeoffs between our approach and some other solutions based on more expressive typing disciplines.

Part of this work was done during a visit to BRICS at the University of Aarhus and DIKU at the University of Copenhagen in the fall of 1997.