APPSEM Workshop Program

7. – 9. September 1999

August 30, 1999

Tuesday 7. Sept

Topic D: Verification methods

Eugenio Moggi (with F.Palumbo): Monadic Encapsulation of Effects: a Revised Approach talk related to HOOTS'99 paper with same title

Launchbury and Peyton-Jones came up with an ingenious idea for embedding regions of imperative programming in a pure functional language like Haskell. The key idea was based on a simple modification of Hyndley-Milner's type system.

We propose a more natural encapsulation construct exploiting higher-order kinds, which achieves the same encapsulation effect, but avoids the bogus type parameter of the original proposal.

The second contribution is a stronger (and hopefully bug-free) type safety result, namely encapsulation of strict state in higher-order lambda-calculus.

We formalize the intended implementation as a very simple big-step operational semantics on untyped terms, which captures interesting implementation details not captured by previously proposed reduction semantics.

We also outline how the definitions and results could be adapted to prove type safety for lazy state, and discuss the main differences between the scritc and lazy state.

Andy Gordon (with L. Cardelli): Anytime, anywhere: modal logics for mobile ambients

We describe a modal logic for mobile processes, based on the ambient calculus. The ambient calculus provides distinct locations where processes may reside. As a consequence, it makes sense to talk about properties that hold at particular locations, and it becomes natural to consider spatial modalities for formulas that hold at some location. In this talk, we describe techniques for establishing logical properties syntactically (via a proof system), semantically (via properties of the model), and "for free" (via type-checking). This is joint work with L. Cardelli.

Peter O'Hearn : Petri Net Semantics of Bunched Implications

Engberg and Winskel have given a model of linear logic using Petri nets; however, they noticed some discrepancies with linear logic, such as the validity of distribution of additive conjunction over disjunction. Here we reconsider their semantics, as a model of the logic BI of bunched implications. BI, like relevant logics, naturally admits distribution, but also includes intuitionistic implication. Completeness and incompleteness results for this semantics of BI will be given, and the suitability of BI as a logic of Petri Nets will be discussed.

Don Sanella (with Furio Honsell) : Pre-logical relations

We study a weakening of the notion of logical relations, called pre-logical relations, that has many of the features that make logical relations so useful as well as further algebraic properties including composability. The basic idea is simply to require the reverse implication in the definition of logical relations to hold only for pairs of functions that are expressible by the same lambda term. Pre-logical relations are the minimal weakening of logical relations that gives composability for extensional structures and simultaneously the most liberal definition that gives the Basic Lemma. The use of pre-logical relations in place of logical relations gives an improved version of Mitchell's representation independence theorem which characterizes observational equivalence for all signatures rather than just for first-order signatures. Pre-logical relations can be used in place of logical relations to give an account of data refinement where the fact that pre-logical relations compose explains why stepwise refinement is sound.

Topic H: Semantics-based optimisation

Lars Pareto : Recursion and Dynamic Datastructures in Bounded Space: Towards embedded ML programming

We present a functional language with a type system such that well typed programs run within stated space-bounds. The language is a strict, first-order variant of ML with constructs for explicit storage management. The type system is a variant of Tofte and Talpin's region inference system to which the notion of sized types, of Hughes, Pareto and Sabry, has been added.

Topic A: Semantics for object-oriented programming

Paul Blain Levy : Game Semantics for Call-By-Push-Value

Call-by-push-value (CBPV) is a fine-grain language paradigm that, we claim, provides the semantic primitives from which typed call-by-name (CBN) and typed call-by-value (CBV) are built. Evidence for this claim can be found in a wide range of semantics, from operational and machine semantics to denotational models such as domains, continuations, functor categories and games.

In this talk, we illustrate the advantages of using CBPV (rather than CBV or CBN) as our primary language of study by looking at game semantics.

We outline a game semantics for CBPV. As expected, this gives rise to the usual (Hyland-Ong-Nickau-McCusker) semantics for CBN and (Abramsky-McCusker-Honda-Yoshida) semantics for CBV. But because our language of study (CBPV) is so fine-grain, the semantics is closer to it—both technically and conceptually—than to CBV or CBN. We see that "asking a question" in the game semantics corresponds to "forcing a thunk" in the CBPV language, while "answering a question" in the game semantics corresponds to "producing to a consumer" in the CBPV language.

Didier Remy : Assembling Objects with Internal Concurrency Using Inheritance of Behavior and Refinement of Synchronization

We propose an object-oriented calculus with internal concurrency, class-based inheritance, views, and a new adjunction operator to build and refine definitions of concurrent programs, incrementally. This includes the refinement of behavior and some (but not all) refinement of synchronization as well.

Wednesday 8. Sept

Topic F: Games, sequentiality and abstract machines

Andrew Ker (with Hanno Nickau and Luke Ong) : Universal Game Models of Untyped Lambda Calculus

I will present the broad ideas of some recent work which uses Hyland/Ong games to construct firstly an extensional and universal (i.e. every element is definable) model of the untyped lambda-calculus modulo the maximal consistent sensible theory H^*, and a refinement which leads to another universal model, this time weakly extensional and which identifies term with the same Böhm tree.

Paul-Andre Mellies : Hypercoherence spaces as concurrent games

We present a game-theoretic interpretation of hypercoherence as a polarized variant of concurrent games [Abramsky-Mellies:lics99], thus explicating the computational (dynamic) content underlying a denotational (static) model of linear logic introduced by Thomas Ehrhard in [Ehrhard:mscs93].

Martin Wehr : Algebraic Analysis of Full Completeness

I have developed a method to derive logic and its intensional semantics from a syntactic parameter. So in fact I have a presentation for a family of logics. The linear logic, geometry of interaction and game semantics are instances of this method.

Abramsky has recently presented a semantic for linear logic via process realisability. I will present a semantics for the family of logics along the lines of Abramskys process realisability. This opens the door for a very general full completeness result. The talk will conclude with a full completeness conjecture for this family of logics.

Marina Lenisa (with Samson Abramsky): Fully Complete Models for ML Polymorphic Type

First we isolate conditions for a categorical model of system F to be *fully complete* for *ML polymorphic types*. Then we present an example of such a model, defined by realizability over a *Linear Combinatory Algebra*, arising from a Geometry of Interaction construction.

In particular, we carry out a *linear* analysis of the standard categorical notion of model for system F. Namely, we consider *adjoint models* of system

F, which arise as co-Kleisli indexed categories of *linear indexed categories*, and such that the canonical adjunction between the two categories is monoidal.

Following a pattern common in Full-Abstraction/Full-Completeness proofs of Game Semantics, we identify conditions on the adjoint model which guarantee the validity of a *Decomposition Theorem*. For morphisms which inhabit ML types, this theorem allows to recover the top level structure of the corresponding Böhm trees. The fine analysis of intuitionistic types in terms of linear types plays an essential role here.

In order to build a "concrete" fully complete model, we focus on realizability models, based on Partial Equivalence Relations (PERs) over a *Linear Combinatory Algebra* (LCA). In fact, PER categories over LCAs can be seen to be linear, and to form adjoint models with their co-Klesli categories.

Our concrete fully complete model is obtained via linear realizability on an LCA of *partial involutions*. This LCA arises as a special case of a general categorical construction, i.e. Abramsky's generalized version of Girard's Geometry of Interaction.

Samson Abramsky : Process Realizability

Realizability in its usual formulations, while useful and powerful, appears limited in its scope. In particular, it seems closely tied to Intuitionistic logic, on the logical side, and to functional computation, on the computational side. We propose a significant broadening of the scope of realizability: on the logical side, to Classical Linear Logic, and on the computational side to encompass concurrent processes. Examples include a realizability model for Classical Linear Logic based on the process calculus CCS. The guiding principle for this notion of process realizability is that of *duality*: a type is understood not just in terms of its canonical "values", but also in terms of the contexts for consuming these values; a formula is understood, not just in terms of its possible proofs, but also of its possible refutations; an interactive protocol is understood in terms of both "clients" and 'servers"; etc. The realizability models for Classical Linear Logic are *not* well-pointed - an essential feature from our point of view. They also offer an interesting and novel basis for rejecting the structural rules of contraction and weakening - so that Linear Logic itself can be seen as arising naturally from this realizability paradigm.

One example of this construction, based on "concurrent games", has led to full completeness results for Multiplicative-Additive Linear Logic, which will be presented at LiCS '99 in Trento. The example based on CCS suggests a number of possible applications to concurrency. It can be seen as a continuation of the work on interaction categories, which overcomes some of the problems which were encountered there. There are also possible applications to the computational interpretation of proofs in Classical Logic.

Topic G: Types and type inference in programming

Giorgio Ghelli : Typing Ambients using Ambient Groups

We present a type system for the calculus of mobile ambients, based on the classification of ambients in groups, and the ability to specify the actions that ambients in one group may exercise over ambients in a different group. The resulting type system allows communication, opening, and ambient crossing to be described at the abstraction level of groups, which is the 'right' abstraction level in some applications we present. This is joint work with Luca Cardelli and Andrew D. Gordon.

Francois Pottier : Three parts for a powerful type inference engine

I would like to discuss a type inference system built by combining several known technical devices: a framework for subtyping-constraint-based type inference, conditional constraints, and rows. Together, these three parts make up a powerful engine. I shall illustrate this claim by providing solutions to three delicate type inference problems: "accurate" pattern matchings, record concatenation, and "dynamic" messages. Although all three problems have been solved already, a significantly different system was proposed in each case. In addition to demonstrating the system's power, this result helps understand the above type inference problems, by showing that they may be solved using a single set of basic mechanisms.

Alan Mycroft : Compile-time pointer control for C/C++

IDL (Interface Definition Language) is a specification language which defines interfaces to remote objects in a language independent manner. The IDL specification (see `http://www.omg.org/`) also defines the mapping of IDL interfaces into various languages including C and C++; these have complicated rules for ownership of storages representing `in`, `out` and `inout` parameters, especially when structured data-types are passed.

Servers providing services specified by IDL may execute for a long time so it is important that space leaks do not occur. The standard C++ mapping provides a set of reference-counting primitives to facilitate programs to keep to the rules, but rely on the programmer behaving impeccably (e.g. not caching a pointer).

We discuss extensions to the C type system which allows these issues to be tracked at compile-time.

A special case of this issue occurs with C's '`scanf`' library routine. Consider the effect on verification and optimisation of code like:

```
void foo()
{   int n;
    scanf('%d', &n);
    for (i=0; i<n; i++) bar(i);
}
```

Here most compilers have to assume that a routine `scanf` having been passed an address (here of `n`) may have stored it away and that calls to `bar()` may therefore update `n` behind the scenes, even to the extent of causing the above code never to terminate! However, if we can declare `scanf` as having type

```
extern int scanf(char *nocache fmt, int *nocache argaddr)
```

then `scanf` is forbidden from from keeping a pointer to either of its arguments.

Luke Ong (with A. S. Murawski) :A Linear-time Algorithm for Verifying MLL Proof Nets via Lamarche's Essential Nets

We consider the following decision problems:

PROOFNET : Given a propositional multiplicative linear logic (MLL) proof structure, is it a proof net?

ESSNET : Given an essential net (of a propositional intuitionistic MLL formula), is it correct?

In this talk we present a linear-time algorithm for ESSNET. By showing that PROOFNET is linearly reducible to ESSNET, we obtain a linear-time algorithm for PROOFNET. Hence the question of whether a given linearly balanced MLL formula is provable can be settled by a linear-time algorithm.

Thurstday 9. Sept

Topic B: Program structuring

Barry Jay : Functorial Lambda-Caluclus

Functorial lambda-calculus is characterised by a semantics in which types are interpreted as functors between finite powers of a base category D rather than as objects of D. This motivates the introduction of novel types and type constructors, and the corresponding constants, which can be used to define generic functions such as equality, map, or fold, that work over all data types.

The principles will be illustrated using the functorial calculus that is implemented within the programming language FISh2 that is currently under development.

Hayo Thielecke : On the (continuation) semantics of ML exception

We have recently shown that exceptions and continuations cannot macro-express each other [ICALP'99]. Nonetheless, exceptions can of course be expressed in terms of continuations by passing a current handler continuation. In this talk, we derive such handler-passing from the exceptions monad, and use it to compare and contrast the exceptions constructs RAISE/HANDLE with the continuations constructs CALLCC/THROW.

Paul Taylor : Imperative interpretation of mathematics?

This is the seminar that I have been intending to present ever since I first heard that Hayo Thielecke was coming (three years ago?). Maybe I should say that it is a first attempt at this seminar.

Hayo in his thesis gave a categorical interpretation of some imperative programming constructs such as "callcc". I don't want to condone the use of such ways of writing source programs, but I find the techniques of evaluating functional programs by powerful imperative features intriguing, and would like to find out whether Hayo's translation can be used in reverse to evaluate morphisms of his category obtained from other mathematical constructions.

It was 1993 when I first proposed a certain categorical axiom, and did a construction that subsequently turned out to be very similar to Hayo's. However, this is a topic that yields up its secrets only slowly and reluncantly, so only in 1997 did I begin to make serious progress with my research programme.

What I have is a system of axioms, of which one model is the category of locally compact locales (and one might observe that a large part of the corpus of mathematics goes on in this category). I DO NOT intend to talk about the constructions I have from a mathematical point of view, but I will state some of these constructions as examples of morphisms in Hayo's category. With my axioms, as with Hayo's and the theory of locales, the "public" logical, topological and programming notions are represented "privately" in the OPPOSITE category.

Topic C : Integration of functional languages and proof assistants

Andrew Pitts : Symbolic Computation Modulo Alpha Conversion

This talk will describe work in progress (with Jamie Gabbay) on the design of a programming language for recursively defined functions on user-declared inductive datatypes involving binding constructors. The language has a novel type constructor for abstractions. Recursive definitions can use a form of pattern-matching on bound names of abstractions, while the type system ensures that only those definitions respecting alpha-conversion of bound names typecheck. The language design is 'semantically driven', in as much as it arises out of contemplating the model of variable-binding in terms of Freankel-Mostowski sets given by Gabbay and Pitts in Proc. LICS '99, pp 214–224.

Topic E : Automatic program transformation

Jens Peter Secher : On Perfect Supercompilation (This talk has already been held at PSI99, Novosibirsk, Russia)

Turchin's supercompiler [3] is a program transformer for functional programs which performs optimizations beyond *partial evaluation* and *deforestation* [4]. *Positive supercompilation* [2] is a variant of Turchin's supercompiler which was introduced in an attempt to study and explain the essentials of Turchin's supercompiler, how it achieves its effects, and its relation to other transformers.

For the sake of simplicity, the positive supercompiler was designed to maintain *positive information* only; that is, when the transformer reaches a conditional **if** $x{=}x'$ **then** t **else** t', the information that $x = x'$ is assumed to hold is taken into account when transforming t (by performing the substitution $x \mapsto x'$ on t). In contrast, the *negative information* that $x \neq x'$

must hold is discarded when transforming t' (since no substitution can represent this information!). In this paper we present an algorithm which we call *perfect supercompilation*. The perfect supercompiler arises by extending the positive supercompiler to take negative information into account.

A main contribution of the extension is to develop techniques which manipulate constraints of a rather general form. Similar techniques have not been presented in the literature, except for the paper by Glück and Klimov [1] which, however, handles constraints of a simpler form; for instance, our algorithm for normalising constraints has no counterpart in their technique. As another main contribution we generalise a technique for ensuring that positive supercompilation always terminates to the perfect supercompiler and prove that, indeed, perfect supercompilation terminates on all programs. No version of Turchin's supercompiler maintaining negative information has been presented which in general is guaranteed to terminate.

References

[1] R. Glück and A.V. Klimov. Occam's razor in metacomputation: the notion of a perfect process tree. In P. Cousot, et. al., editors, *Workshop on Static Analysis*, volume 724 of *LNCS*, pages 112–123. Springer-Verlag, 1993.

[2] R. Glück and M.H. Srensen. A roadmap to metacomputation by supercompilation. In O. Danvy, et. al., editors. *Partial Evaluation*, volume 1110 of *LNCS*. Springer-Verlag, 1996, pages 137–160.

[3] V.F. Turchin. The concept of a supercompiler. *ACM Transactions on Programming Languages and Systems*, 8(3):292–325, 1986.

[4] P.L. Wadler. Deforestation: Transforming programs to eliminate intermediate trees. *Theoretical Computer Science*, 73:231–248, 1990.

Topic I : Domain theory and real number computation

Abbas Edalat : Foundation of a Computable Topology for Solid Modelling

Computational geometry and solid modelling are based, on the one hand, on classical notions of topology and, on the other hand, on the Real RAM (Random Access Machine) model of computation.

These foundations, which are classically consistent, are both unsatisfactory in any feasible computational framework.

In fact, the basic predicates and operations in topology, such as the classical membership predicate of a proper subset of the Euclidean space or the binary operation of intersection of, say, compact subsets, are in general discontinuous and, therefore, non-computable. Furthermore, the Real RAM model is unrealistic since it assumes that comparison of real numbers is decidable in finite time.

Consequenctly, classically correct algorithms in computational geometry and solid modelling, when implemented for actual computation, turn into unreliable programs which can crash.

We present a domain, equipped with its Scott topology, to model the space of solid objects. In this model, all the basic predicates and operations, such as the membership predicate and the intersectipon operator, are indeed continuous. This framework is equipped with a well-defined and realistic notion of computability which reflects the observable properties of real solids. It admits regular and non-regular sets, and supports a design methodology for reliable (robust) algorithms. Moreover, the model is able to capture the uncertainties of input data in actual CAD situations.

Reinhold Heckmann : Complexity of Real Arithmetic by 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. We study the space and time complexity of this approach. Both kinds of complexities are connected with the growth of the entries of the involved LFT's during a computation. The time complexity is also affected by the number of argument digits needed to produce a certain number of result digits. We present upper and lower bounds for many simple functions and operations, and for exponential and square root.

Hideki Tsuiki : Gray Code Representation of Exact Real Numbers

We propose a new representation of exact real numbers using Gray code. It is known that a representation of real numbers becomes redundant if we want to construct, with the type 2 machine, the same computability structure as the Cauchy, the signed binary, and other admissible representations. In this talk, we propose a new representation based on Gray code expansion, which is injective (i.e. not redundant) and with a modified Type 2 machine induces the same computable structure on REAL as the Cauchy representations. This means that REAL with its standard computable structure can be embedded into the space of infinite words by considering a different sort of machine. We also give algorithms for some basic operations with respect to this representation. We also introduce a variant of Real PCF based on this representation.

Alex Simpson : A categorical notion of model for PCF

We give elementary categorical structure that axiomatizes a notion of model for PCF. Datatypes are determined by universal properties. The fixed-point operator is uniquely characterised and satisfies all true first-order recursion equations. There is a free term model allowing a functorial semantics for PCF. There is also a fully abstract term model whose existence means that denotational proofs of program equalities can be directly reinterpreted as purely syntactic operational proofs.

Achim Jung : A relational category of topological spaces

I will present a study of a category of topological spaces (stably locally compact) where the arrows are certain closed relations. In previous work a purely logical description of this category was given.

Specifically, the new results concern adjoint pairs of maps and bilimits of directed diagrams. The technique generalises both standard domain theoretic and purely topological constructions. (This joint work with Mathias Kegelmann and Andrew Moshier.)