ISSN 0105-8517

Proceedings of the

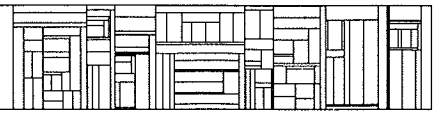
CLICS Workshop – Part I

Aarhus University, 23-27 March 1992

Glynn Winskel, ed.

DAIMI PB – 397-I
May 1992

COMPUTER SCIENCE DEPARTMENT
AARHUS UNIVERSITY
Ny Munkegade, Building 540
DK-8000 Aarhus C, Denmark



Foreword

This is a collection of papers, notes and copies of transparencies representing the talks of the CLICS[1] Workshop at the Computer Science Department, Aarhus University the 23 - 27 March 1992.

Glynn Winskel

Contents

- Programme.
- Abstracts of talks.
- Copies of transparencies and relevant papers.
- Address list.

[1]CLICS is an Esprit Basic Research Action on Categorical Logic in Computer Science.

CLICS WORKSHOP
Aarhus University, Denmark

PROGRAMME

Monday:

		page
09.00-10.00	J. Y. Girard. On the geometry of interactions[1].	
10.05-10.50	S. Abramsky. New foundations for the geometry of interactions.	1 – 42
	Coffee	
11.20-12.05	P. Wadler. There's no substitute for linear logic.	43 – 70
12.15-13.00	G. Bierman. Type systems, linearity and functional languages.	71 – 92
	Lunch	
14.15-15.00	V. Pratt. Event spaces.	93 – 114
15.05-15.50	C. Barry Jay. Tail Recursion Through Universal Invariants.	115 – 138
	Tea	
16.20-17.05	G. Longo. A new challenge: The categorical semantics of ad hoc polymorphism.	139 – 160
17.15-18.00	R. Tennent. Semantics of local variables.	161 – 174
18.00-18.20	A. Joyal. Conway games.	175 – 182

Tuesday:

09.00-10.00	T. Coquand. Interaction sequences.	183 – 212
10.05-10.50	R. Crole. Recursive types via fixpoint objects.	213 – 224
	Coffee	
11.20-12.05	E. Moggi. Evaluation logic, an overview.	225 – 242
12.15-13.00	P. Dybjer. A construction of a Frege structure in predicative type theory.	243 – 260
	Lunch	
14.15-15.00	S. Vickers. Topical categories of domains.	261 – 274
15.05-15.50	P. T. Johnstone. Variations of the bagdomain theme.	275 – 282
	Tea	
16.20-17.05	T. Streicher. Towards an axiomatization of domain theory in type theory.	283 – 303
17.15-18.00	A. Schalk. Power locales.	304 – 304
18.00-18.45	P. Taylor. Tarski's theorem for cpo's, constructively.	305 – 322

[1]There is no contribution for this talk.

Wednesday:

		page
09.00-10.00	P. Freyd. Towards axiomatic domain theory.	323 – 328
10.05-10.50	W. Phoa. Synthetic domains and operational semantics.	329 – 340
	Coffee	
11.20-12.05	M. Fourman. Set-theoretic semantics for ML.	341 – 358
12.15- 13.00	R. Amadio. On adequacy of per-models.	359 – 380
	Lunch	
	Free afternoon.	

Thursday:

09.00-10.00	M. Felleisen. Observable sequentiality and full abstraction.	381 – 412
10.05-10.50	S. Brookes. Computational comonads and intensional semantics.	413 – 458
	Coffee	
11.20-12.05	A. Pitts. Coinduction for recursively defined domains.	459 – 480
12.15-13.00	P. Scott. Remarks on the Π calculus and linear logic.	481 – 514
	Lunch	
14.15-15.00	Y. Lafont. Genetic abstract machine.	515 - 532
15.05-15.50	E. Ritter. Categorical abstract machines for higher order typed λ-calculi.	533 – 545
	Tea	
16.20-17.05	B. Jacobs. Affine and material semantics.	546 – 546
17.15-18.00	V. de Paiva. Full intuitionistic linear logic.	547 – 570

Friday:

09.00-10.00	G. Plotkin. On compactness and CPO-enriched categories.	571 – 584
10.05-10.50	E. Robinson. A categorical account of parametricity and type abstraction.	585 – 598
	Coffee	
11.20-12.05	P. Francois Lamarche. Sequentiality, Games and Linear Logic.	599 – 608
12.15-13.00	J. Power. A logical framework based on categories with structure.	609 – 612

List of Participants 613 – 616

AUTHOR: Samson Abramsky, Radha Jagadeesan

TITLE: **New foundations for the Geometry of Interaction**

Abstract

Girard has proposed the programme of Geometry of Interaction, which aims to find a new middle ground between denotational and operational semantics, capturing the essential features of computational dynamics in a syntax-free fashion using denotational tools. He has implemented this programme using the formalism of C^*-algebras.

In this paper, we present a new formal embodiment of Girard's programme, with the following salient features:

1. Our formalisation is based on Domain Theory, rather than C^*-algebras. We replace Girard's "execution formula" by a least fixpoint, essentially a generalisation of Kahn's semantics for feedback in dataflow networks. Coupled with the use of the other distinctive construct of Domain theory, the lifting monad, this enables us to interpret the whole of Classical Linear Logic, and to prove soundness in full generality, thus addressing the technical limitations of Girard's formalisation.
2. We give a concrete computational interpretation of the Geometry of Interaction in terms of dataflow networks.
3. We show how the interpretation arises from the construction of a categorical model of Linear Logic, and identify exactly what structure is required of the ambient category in order to carry out the interpretation. This makes the definition of the interpretation much more transparent.

AUTHOR: Roberto Amadio

TITLE: **On the Adequacy of Per Models**

Abstract

We consider a fixed point extension of the second order lambda calculus equpped with a call by value evaluation mechanism. We interpret the language in a partial cartesian closed category of directed complete partial equivalence relations over a domain theoretic model of a type-free, call-by-value, lambda calculus. Our main result is that the notions of syntactic and semantic convergence coincide.

AUTHOR: Gavin Bierman

TITLE: **Type Systems, Linearity and Functional Languages**

Abstract

In their early paper, Girard and Lafont showed that functional languages based on linear logic could be implemented in an efficient way. For example, Lafont's machine requires no garbage collector. However, previous experience with so-called linear functional languages has not been good: the program structure often becomes lost in annotations to placate the type checker. Ideally we would like to write our programs in traditional functional languages and let the compiler produce the appropriate annotations. We gained inspiration from Girard's translation of intuitionistic types to intuitionistic linear logic types and have been investigating using type systems to annotate lambda terms.

We have found problems with using linear logic types and have thus divided the of-course modality to provide finer sharing information. We have found that such a type system can infer more linearity than one based on linear logic alone.

My talk will discuss using linear logic to annotate lambda terms and reasons why perhaps using just linear logic is not that helpful. I will discuss my type system and highlight some differences between it and ones based on linear logic. I shall also point out some future work.

AUTHOR: Stephen Brookes

TITLE: **Computational Comonads and Intensional Semantics**

AUTHOR: Thierry Coquand

TITLE: **Interaction Sequences: Another Proof of Cut-Eliminations**

Abstract

We present a game-theoretic analysis of classical provability, where each formula defines a game, and a proof for this formula defines a winning strategy for this game. A cut is then seen as an "internal communication" between two players, and the result of cut-elimination is replaced by the fact that "internal chatters" end eventually. We apply this to give a (may be new) proof of cut-elimination for classical infinitary propositional logic.

AUTHOR: Roy Crole

TITLE: **Recursive Types via Fixpoint Objects**

Abstract

We introduce a new dependently typed equational logic, FIXmu which contains both a type universe and a fixpoint object, building upon core material which was introduced in Crole-Pitts. Within the logic FIXmu fixpoints are definable at the level of terms, inducing fixpoints at the level of types by a coding through the type universe. Thus we are able to solve recursive domain equations with the aid of the fixpoint type. The logic has a clean categorical semantics, being modelled by a particular variety of category-with-attributes. We note a mild generalisation of information systems, from which we prove a representation theorem for Scott predomains, and use this to describe a concrete model of our logic. FIXmu may be viewed as a programming metalogic into which programming languages with recursive types may be translated and reasoned about. We demonstrate this by proving computational adequacy results of FIXmu for a PCF-style language with recursive types.

(Crole-Pitts is the paper "New Foundations for Fixpoint Computations" published in the 1990 LICS proceedings, Authors Roy L. Crole and Andrew M. Pitts).

AUTHOR: Peter Dybjer

TITLE: **Constructing a Frege Structure in Predicative Type Theory**

Abstract

This talk describes the formalization and construction of Aczel's Frege structures in Martin-Löf's predicative type theory. Frege structures are models of the Logical Theory of Constructions, which is a consistent version of the theory in Frege's Grundgesätze. It can also be viewed as a consistent version of illative combinatory logic.

A central part of the construction concerns the simultaneous definition of the notions of proposition and truth. This kind of definition has been called half-positive, and has been explained by Aczel in classical set theory. In type theory Aczel's explanation can be refined: the presence of explicit proof objects makes it possible to analyse it as a simultaneous inductive definition of the notion of proposition and and a recursive definition of the notion of truth.

Another part is the construction of a model of the untyped lambda calculus. This construction is another example of the use of a general formulation of inductive and recursive definitions in type theory.

AUTHOR: Matthias Felleisen

TITLE: **Observable Sequentiality and Full Abstraction**

Abstract

One of the major challenges in denotational semantics is the construction of fully abstract models for *sequential* programming languages. For the past fifteen years, research on this problem has focused on developing models fpr PCF, and idealized functional programming language based on the typed lambda calculus. Unlike most practical languages, PCF has no facilities for *observing* and *exploiting* the evaluation order of arguments in procedures. Since we believe that such facilities are crucial for understanding the nature of sequential computation, this paper focuses on a sequential extension of PCF (called SPCF) that includes two classes of control operators: error generators and escape handlers. These new control operators enable us to construct a fully abstract model for SPCF that interprets higher types as sets of *error-sensitive* functions instead of *continuous* functions. The error-sensitive functions form a Scott domain that is isomorphic to a domain of decision trees. We believe that the same construction will yield fully abstract models for functional languages with different control operators for observing the order of evaluation.

AUTHOR: Michael Fourman

TITLE: **Set-Theoretic Semantics for ML**

Abstract

In this talk we describe joint work with Wesley Phoa. These are the initial results of a project, recently begun, which seeks to define a categorical semantics for Standard ML—in particular, one for which naive set-theoretic reasoning is sound.

Standard ML has both an expressive type structure and a powerful and flexible modules system; giving a clean and unified denotational semantics for it is a challenging problem. This paper outlines an approach, based on naive set-theoretic and categorical intuitions, to modelling a purely functional, explicitly typed variant of ML—'Pure ML': We take a *set-theoretic* view of algebraic datatypes - - - —i.e. those (possibly recursive) ML datatypes which contain no arrow types in their definitions. Our goal is to explain how a *domain-theoretic* semantics, which is necessary in the more general case, can be compatible with simple set-theoretic intuitions about these particular datatypes. *Synthetic domain theory* allows us to make sense of fixed points and recursive data types in a set-theoretic setting.

Our treatment of polymorphism and modules is novel. The idea that one needs polymorphic types to model the polymorphic type constructors of ML is misleading. Instead, we formalise the notion that a polymorphic type constructor is a construction on an *indeterminate* type, in a precise categorical sense. Our approach relies on results from *topos theory*, the categorical formulation of constructive set theory. the theory of *classifying toposes* and the notion of a generic structure, allow us to interpret type variables explicitly, and provide a straightforward way of interpreting signatures, functors and sharing specifications.

AUTHOR: Bart Jacobs

TITLE: **Affine and Material Semantics**

Abstract

By a semantic analysis it can be shown that the standard embedding of intuitionistic logic into (intuitionistic) linear logic — which makes essential use of Girard's ! — can be factored in two different ways: either via *affine logic* (in which one standardly has weakening but contraction only using an explicit operation $!^+$) or via¿*material logic* in which one has contraction and an operation $\bot$ for weakening). Operationally, one can think of

$>!A$ as A, as often as you like;
$!^+A$ as A, at *least* once;
$\bot A$ as A, at *most* once.

This yields two insights:

- Girard's ! is not a primitive operation, but can be described as composite $\bot!^+$ or as $!^+\bot$.
- Linear functions $!^+A \to B$ use their input at least once: this suggests eager evaluation. Similarly $\bot A \to B$ suggests lazy evaluation, because an element $a \in A$ may be discarded.

All this has a clean categorical semantics using notions from monad theory, introduced some 25 years ago by Anders Kock and Jon Beck.

AUTHOR: Barry Jay

TITLE: **Tail Recursion Through Universal Invariants**

Abstract

Tail-recursive constructions suggest a new style of semantics for datatypes, which allows a direct match between specifications and tail-recursive programs. The semantics is expressed in terms of the formal properties of loops, their fixpoints, invariants and convergence. Convergent models of the natural numbers and lists are examined in detail, and, under very mild conditions, are shown to be equivalent to the corresponding initial algebra models.

AUTHOR: Peter Johnstone

TITLE: **Variations on the Bagdomain Theme**

Abstract

The recent introduction of the bagdomain monad on the 2-category of toposes [1] was prompted by the work of S. Vickers [2] on domain-theoretic modelling of databases, for which something more general than the powerdomain is required. However, bagdomains come in an extraordinarily wide variety of flavours (much wider than powerdomains), and it seems likely that several of these besides the lower bagdomain studied in [1] will find applications in theoretical computer science, for example in the semantics of nondeterminism. The object of this talk will be to convey something of the range of possibilities available, and to describe how they relate to one another and to the various powerdomain monads.

[1] P.T. Johnstone, Partial Products, Bagdomains and Hyperlocal Toposes, to appear in Proceedings of the 1991 Durham Symposium on Applications of Categories in Computer Science.

[2] S.J. Vickers, Geometric Theories and Databases, ibid.

AUTHOR: Yves Lafont

TITLE: **Genetic Abstract Machines**

AUTHOR: Giuseppe Longo

TITLE: **A new challenge: the categorical semantics of ad hoc polymorphism**

AUTHOR: Eugenio Moggi

TITLE: **Evaluation Logic, an overview**

Abstract

Evaluation Logic aims to be a metalanguage for program logics in the same way as the computational lambda-calculus aims to be a metalanguage for programming languages. In this talk we review the categorical semantics proposed so far, give some examples of its expressiveness and indicate some problematic issues.

AUTHOR: Valeria de Paiva

TITLE: **Full intuitionistic linear logic**

AUTHOR: Wesley Phoa

TITLE: **Synthetic domains and operational semantics**

AUTHOR: Andrew M. Pitts

TITLE: **Co-induction for recursively defined domains**

Abstract

Recent work of Freyd on recursive types is based upon the characterization of solutions to functorial domain equations as objects that are simultaneously inital algebras and final coalbegras for functorial type contructors. Motivated by this work I will present two apparently new proof principles for reasoning about predomains recursively defined using arbitrary (ie. possibly not functorial) combinations of the sum, product, lifting and partial function space constructors. The first (induction) principle generalizes structural induction for inductive types and entails Scott's fixpoint induction. The second (co-induction) principle seems more interesting: it characterizes the information ordering on a recursive type in terms of a notion of "simulation" in the manner of Park and Milner. Abramsky's "internal full abstraction" result for the canonical model of his lazy lambda calculus is a particular instance of this principle.

AUTHOR: John Power

TITLE: **A logical framework based on categories with structure**

Abstract

We outline the relationship of a broad class of logics and type theories with categories with algebraic or, more generally, essentially algebraic structure, such as cartesian closed categories, toposes, and monoidal closed categories. We then move to the specific case given by logic programming. The general situation allows us to define a notion of map between logic programs, which we use to define a notion of an extension of a logic program. This acts as a basis upon which one can begin a structural analysis of logic programs, and in particular we prove a precise theorem giving local conditions under which all computations of a program lift to its extension.

AUTHOR: Vaughan Pratt

TITLE: **Event Spaces**

Abstract

An event space is a poset with top and arbitrary nonempty joins, whose maps preserve these. These form the first example know to this author of a nondegenerate algebraic model of full linear logic, coherence spaces being nondegenerate but not algebrfaic and complete semilattices being algebraic but degenerate. As such they affirmatively answer Johnstone's 1978 question as to whether there exist other self-dual algebraic categories besides complete semilattices. An important application is to concurrency, where event spaces and their dual state spaces strictly extend event structures and their dual families of configuration.

AUTHOR: Eike Ritter

TITLE: **Categorical Abstract Machines for Higher-Order Typed λ-Calculi**

Abstract

This talk will present a way of explaining categorical abstract machines via Ehrhard's notion of D-category, which is a particular kind of indexed category. This notion makes it possible to represent concepts that are important for the design of abstract machines, like environment or closure, directly in categorical terms. We give a uniform treatment of the Cousineau-Curien CAM and Krivine's machine, and extend them to machines for the Calculus of Constructions. These machines are constructed in three steps. Firstly, an equational presentation of the categorical structure yields categorical combinators. Secondly, one derives inference rules

describing the reduction to canonical combinators. There are several possible inference systems, e.g. one for eager reduction and another one for the lazy reduction. Finally, these inference systems yield directly abstract machines.

If we apply this framework to the constant D-categories, which are the appropriate structure to model the typed lambda-calculus, the eager machine is essentially the CAM, and the lazy machine specializes to Krivine's machine. Because the categorical structure corresponding to the Calculus of Constructions is a D-category with additional properties, we get generalizations of the CAM and Krivine's machine as well. It is also possible to derive an algorithm for the type checking of these combinators, which uses the above reduction machines.

AUTHOR: Edmund Robinson

TITLE: **A categorical account of parametricity and type abstraction**

Abstract

John Reynlds has stressed the link between parametricity and type abstraction. The aim of this work is to give a simple semantic account of the link in terms of linguistic features and extensions of the category of types. At the moment the work is at a preliminary stage.

AUTHOR: Andrea Schalk

TITLE: **Power Locales**

Abstract

As the category of locally compact locales is equivalent to the category of locally compact sober spaces we can view the monads given by the various power locale constructions as monads over topological spaces.

This setting makes it possible for us to determine the categories of algebras for some of these monads.

AUTHOR: Philip J. Scott

TITLE: **Remarks on Pi Calculus and Linear Logic**

Abstract

This work, joint with GianLuigi Bellin, examines connections of a version of Milner's Pi-calculus ("Synchronous Pi Calculus") with respect to its faithfulness in reflecting proof-theoretic properties of Multiplicative and Additive linear logic. This follows similar lines to the Abramsky program of "process interpretations" of linear logic.

If time permits, other recent work on connecting linear logic with concurrent computation will be mentioned; in particular, with categorical models of concurrent constraint programming.

AUTHOR: Thomas Streicher

TITLE: **Towards an Axiomatization of Domain Theory in Type Theory**

Abstract

We will present a first attempt of axiomatizing domain theory using the LEGO implementation of the Extended Calculus of Constructions.

Furthermore we will show that propositions expressing continuity such as Scotts Axiom are not extensionally realised.

AUTHOR: Paul Taylor

TITLE: **Tarski's theorem for cpo's, constructively**

Abstract

As regards giving a talk, I'd quite like to present my proof of Tarksi's theorem for cpos (not lattices) and monotone functions WITHOUT excluded middle. This involves a reformulation of the theory of ordinals & transfinite induction and is valid in any elementary topos. Some of the ideas came from synthetic domain theory, and the reason for foreswearing excluded middle is with a view to importing as much mainstream mathematics (including domain theory) as possible into toposes such as the effective topos.

AUTHOR: Bob Tennent

TITLE: **Semantics of Local Variables**

Abstract

A progress report on the use of categorical methods to address the problem of giving sufficiently abstract semantics to local variable declarations in Algol-like languages.

AUTHOR: Steven Vickers

TITLE: **Topical categories of domains**

Abstract

It is shown how many techniques of categorical domain theory can be expressed simply in the general context of topical (i.e. internal in the category Top/S of Grothendieck toposes with geometric morphisms) Cartesian closed categories, the toposes of objects and morphisms classifying information systems and approximable mappings respectively. These techniques are illustrated in an example in which the corresponding points form a category equivalent to that of SFP predomains and Scott continuous maps.

AUTHOR: Philip Wadler

TITLE: **There's no substitute for linear logic**

Abstract

Some problems with substitution and the cut rule in linear logic are pointed out, and some variations on linear logic proposed.

New Foundations for the Geometry of Interaction

Samson Abramsky and Radha Jagadeesan
Imperial College

(Paper to appear in LICS '92 anonymous - ftp: theory.doc.ic.ac.uk)

Traditionally, we have the dichotomies

Mathematical structure	Computational dynamics
denotational	operational
declarative	procedural
logical	computational

Much recent work in CS can be seen as trying to find middle ground between these dichotomies

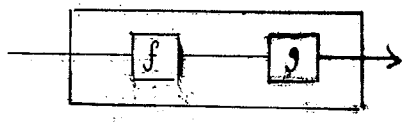
e.g. stability & sequentiality in Domain theor

constraint programming ...

initiated by Girard, to find a middle ground between denotational & operational semantics.

Denotational semantics (e.g. of functional or logic programming) loses the dynamics – typically in composition (Cut):

$g \circ f$



We get the input-output behaviour, but lose the internal structure of events which accomplishes it.

Operational semantics is too bound up with "surface structure" (syntax, variables, substitution) and does not expose the mathematical "deep structure" of the dynamics clearly to view.

(1) Syntax-free (e.g. no names, variables ...)

Denotational tools used to study dynamics.

(2) The dynamics is modelled as the flow of information (tokens) around a "fixed" network - cf. dataflow, Petri nets, sequential algorithms on concrete data structures.

(3) There is a normal form analogous to Kleene normal form in recursion theory - the entire dynamics of a system is represented by iterations of a single "operator". (Needed for (1)).

[(4) Purely local computation - no synchronization]

of GI, for the multiplicative fragment of CLL, with the following additional restriction: Axiom is only used for propositional atoms α:

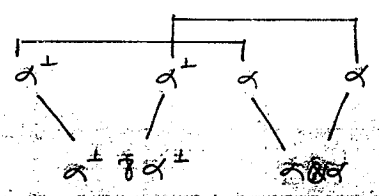
$$\vdash \alpha^\perp, \alpha$$

First idea: look at cut-free proofs, e.g. of $\forall\alpha.\ (\alpha \otimes \alpha) \multimap (\alpha \otimes \alpha)$.
(There are just two: id and twist).

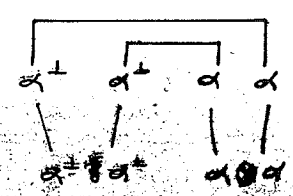
id:
$$\frac{x:\alpha^\perp,\ x:\alpha \qquad y:\alpha^\perp, y:\alpha}{x \parr y^{:\alpha^\perp \parr \alpha^\perp},\ x \otimes y : \alpha \otimes \alpha}\ (\text{Times, Par})$$

twist:
$$\frac{x:\alpha^\perp,\ x:\alpha \qquad y:\alpha^\perp, y:\alpha}{x \parr y^{:\alpha^\perp \parr \alpha^\perp},\ y \otimes x : \alpha \otimes \alpha}\ (\text{Times, Par})$$

id:



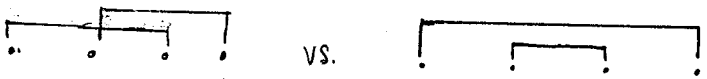
twist:



(1) The proof expression is a set of trees (terms) one for each formula in the sequent, with the leaves linked up in pairs by axiom links.

(2) The structure of the trees (in this fragment) is uniquely determined by the formulas in the sequent.
(This is where the restriction on axioms is used).

Hence a complete invariant to differentiate all distinct cut-free proofs of a given sequent is given by the information as to how the leaves are joined up:



vs.

in our example.

- the set of atoms, A
- a permutation π on A, the product of the transpositions corresponding to the axiom links.

So we have a representation of cut-free proofs in terms of "information flow" between tokens.

NB:
(We are using functions to represent this flow, but without input/output bias, as the flow is bidirectional).

Now comes the key idea: we will extend this representation to model the dynamics of cut-elimination in terms of information flow, by iterating a permutation.

Generally, given

$$t_1 \perp u_1, \ldots, t_k \perp u_k \; ; \; \widetilde{t}$$

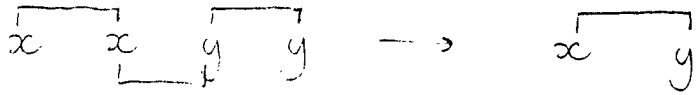
we can apply the rule

$$(*) \quad t \otimes u \perp t' \parr u' \longrightarrow t \perp t', \; u \perp u'$$

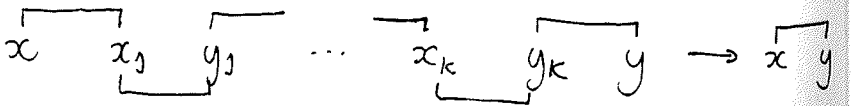
repeatedly until we get an expression of the form

$$x_1 \perp y_1, \ldots, x_n \perp y_n \; ; \; \widetilde{t}$$

In a sense, the purpose of $(*)$ is just to match up corresponding axiom links; the "real" information flow is accomplished by the axiom reduction



or more generally,



For all ~~the~~ proofs ~~of~~ ~~type~~, this can be done just in terms of permutations on finite sets

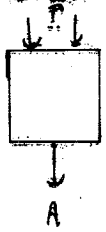
Extending these ideas to full CLL is much harder, and requires more sophisticated tools.

In particular, the shape of the trees, and hence how the leaves have to be matched up, is no longer determined by the types, and must be computed dynamically

The different shapes that can arise correspond to the different computation paths through the program.

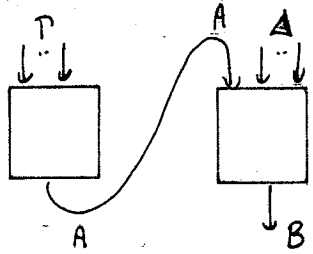
(3) Functional interpretation of IL, ILL:

$\Gamma \vdash A$ ⇝

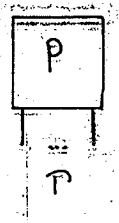


Axiom

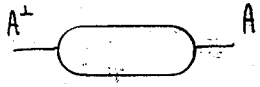
Cut



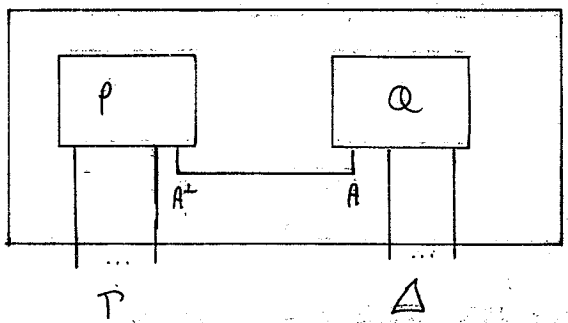
$\vdash \Gamma$ ⇝



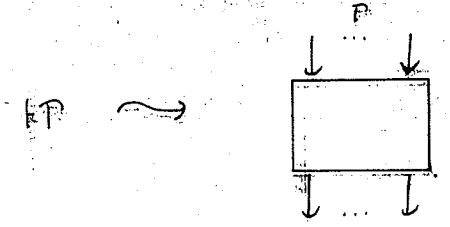
Axiom = buffer:



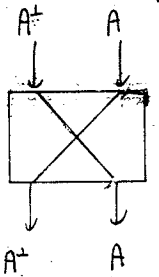
Cut = $(P \mid Q) \setminus x$



Functions modelling bidirectional, symmetric flow of information

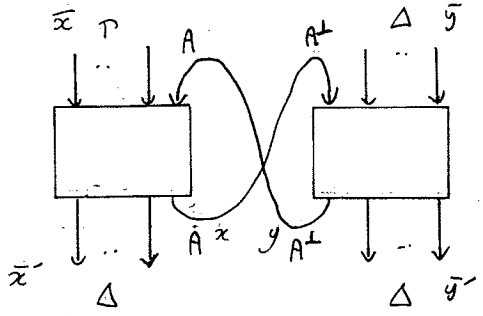


$\vdash \Gamma$ ⇝



Axiom

(NB: "forces" $A^{\perp} = A$)



Cut

The key idea of the Geometry of Interaction is to model this by generalizing Kahn's model of feedback in dataflow networks, using Domain theory.

If $f: D^n \times D \longrightarrow D^n \times D$

$g: D \times D^m \longrightarrow D \times D^m$

(D a domain, f, g continuous)

then $f \cdot g : D^n \times D^m \longrightarrow D^n \times D^m$

is defined by:

$$f \cdot g\,(\vec{x}, \vec{y}) = \begin{cases} \pi\,(f(\vec{x}, x_0),\ g(y_0, \vec{y})) \text{ where} \\ (x_0, y_0) = Y(\lambda(x,y).\ \sigma \circ (f \times g)(\vec{x}, x, y, \vec{y})) \end{cases}$$

where $\sigma(\vec{x}, x, y, \vec{y}) = (y, x)$

$\pi(\vec{x}, x, y, \vec{y}) = (\vec{x}, \vec{y})$.

This fixpoint gives the least solutions of

$$(\vec{x}', x) = f(\vec{x}, y)$$

$$(y, \vec{y}') = g(x, \vec{y}).$$

(1) GI models constructed from suitable categories of domains.

(2) Type-free version : "universal" domain + retracts. Simple examples

(3) Normal form.

Results. · We get an interpretation of all of CLL.

· Soundness holds in full generality

· Further results on "fine structure"

· Dataflow interpretation.

We start from a suitable category of predomains closed under:

· finite products and coproducts and distributivity:

$$dist_{A,B,C} : A \times (B + C) \xrightarrow{\sim} A \times B + A \times C$$

· Lifting monad $((\cdot)_\perp, up, \mu)$

Tensorial strength $t_{A,B} : A \times B_\perp \longrightarrow (A \times B)_\perp$

· solutions of

$$\tau A = (A + 1 + \tau A \times \tau A)_\perp$$

Domains are the algebras of the lift monad:

$$\alpha_D : D_\perp \longrightarrow D$$

[redacted]

- Objects ≡ domains in $\mathbb{C}$
- $f : A \rightsquigarrow B$ is $f : A \times B \longrightarrow A \times B$ in $\mathbb{C}$
- $I_A : A \rightsquigarrow A$ is $A \times A \xrightarrow{\text{symm}} A \times A$
- If $f : A \rightsquigarrow B$, $g : B \rightsquigarrow C$,

  $g \cdot f : A \rightsquigarrow C$ is

  $g \cdot f\,(a, c) = \pi \circ Y(\lambda(x, y).\ \sigma \circ (f \times g)(a, x, y, b))$

  $\pi(a, x, y, b) = (a, b)$

  $\sigma(a, x, y, b) = (a, y, x, b)$.

Duality $A^\perp = A$

If $f : A \rightsquigarrow B$, $f^\perp$ is $B \times A \xrightarrow{\text{symm}} A \times B \xrightarrow{f} A \times B$

$\xrightarrow{\text{symm}} B \times A$.

Clearly $(\ )^{\perp\perp} = \mathrm{Id}$

[redacted]

$I = \perp = 1$

Get monoidal structure from product in $\mathbb{C}$

assoc : $(A \times B) \times C \cong A \times (B \times C)$

symm : $A \times B \cong B \times A$

unit : $A \times 1 \cong A$

via $f : A \cong B \longmapsto A \times B \xrightarrow{f \times f^{-1}} B \times A \xrightarrow{\text{symm}} A \times B$

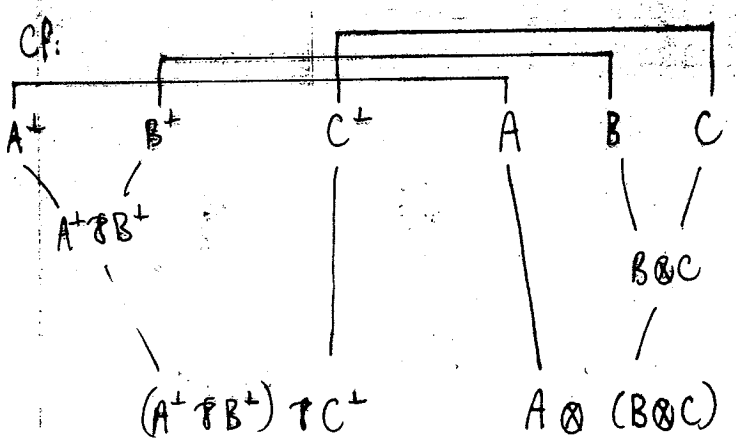
(NB: $\hat{f}^\perp = \hat{f}^{-1} = \widehat{f^{-1}}$).

[redacted]

$\widehat{assoc}\ ((a,b),c),(u,(v,w))) = ((a,(b,c)),(u,v),w)$

$\widehat{assoc} : (A \otimes B) \otimes C \rightsquigarrow A \otimes (B \otimes C)$

CP:



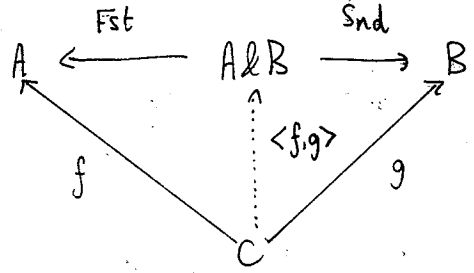
[redacted]

$A \& B \equiv A \oplus B \equiv (A+B)_\perp$

Define

$\ell : A \lhd (A+B)_\perp : \ell^*$ $\qquad$ $r : B \lhd (A+B)_\perp : r^*$

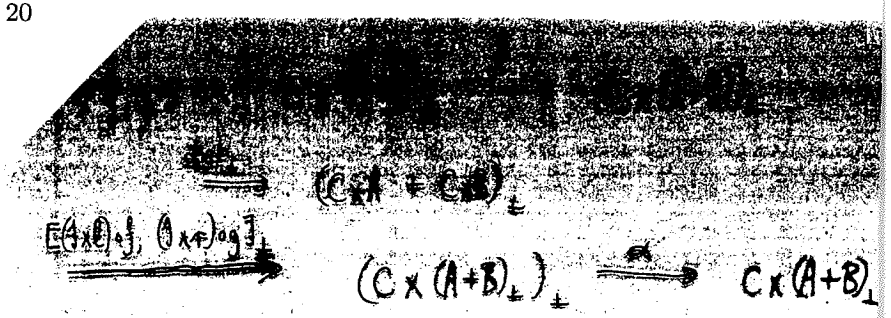
$\ell = inl ; up$ $\qquad$ $\ell^* = [1, \perp]_\perp$) ✓



$Fst = (A+B)_\perp \times A \xrightarrow{\ell^* \times 1} A \times A \xrightarrow{I_A} A \times A \xrightarrow{\ell \times 1} (A+B)_\perp \times$

so $Fst(inl(x), z) = (inl(x), x)$

$Fst(inr(y), z) = Fst(\perp, z) = (inl(\perp), \perp)$



So $\langle f,g\rangle (x, \mathrm{in}_l(y)) = (x', \mathrm{in}_l(y'))$, where $(x',y') = f(x,y)$

$\langle f,g\rangle (x, \mathrm{in}_r(y)) = (x', \mathrm{in}_r(y'))$ where $(x',y') = g(x,y)$

$\langle f,g\rangle (x,\bot) = (\bot,\bot)$.

This is almost categorical product:

$\mathrm{Fst} \cdot \langle f,g\rangle = f \qquad \mathrm{Snd} \cdot \langle f,g\rangle = g$

$\langle \mathrm{Fst}\cdot h, \mathrm{Snd}\cdot h\rangle \sqsubseteq h$

$y \neq \bot \Rightarrow \langle \mathrm{Fst}\cdot h, \mathrm{Snd}\cdot h\rangle (x,y) = h(x,y)$.

(4) From typed to type-free

Domain D with retracts

$m: D^2 \triangleleft D : m^*$

$u: \mathbb{1} \triangleleft D : u^*$

$a: (D+D)_\bot \triangleleft D : a^*$

$e: \tau D \triangleleft D : e^*$

Interpret all types as D, use retracts to internalize constructions from $\mathcal{GI}(\mathbb{C})$.

Signature Σ

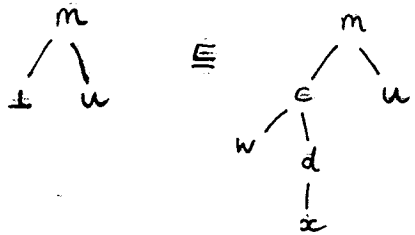
$\Sigma_0 = \{u, w\}$

$\Sigma_1 = \{\ell, r, d\}$

$\Sigma_2 = \{m, c\}$

$W(\Sigma, X)$ Free Σ-algebra on X

$W_\perp(\Sigma, X)$ Free ordered Σ-algebra



$W_\perp^\infty(\Sigma, X)$ Free continuous Σ-algebra

$= \mathbb{E}al(W_\perp(\Sigma, X))$

$(\perp, \perp)$ otherwise.

Push all cuts up to a single simultaneous fixpoint at the top level.

If π is a proof of $A_1, \ldots, A_n$ with m cuts between $B_1, B_1^\perp, \ldots, B_m, B_m^\perp$ the object interpreting π will be a function

$$f : D^{2m+n} \longrightarrow D^{2m+n}$$

Cut will be interpreted by juxtaposition

If $f : D^{2m'+n'+1} \longrightarrow D^{2m'+n'+1}$

$g : D^{2m''+n''+1} \longrightarrow D^{2m''+n''+1}$

let $m = m' + m'' + 1$

$n = n' + n''$.

$$f \cdot g : D^{2m+n} \longrightarrow D^{2m+n}$$

$f \cdot g = \sigma \circ (f, g) \circ \sigma$

where $\sigma(\vec{u}, \vec{x}, x, \vec{v}, \vec{y}, y) = (\vec{u}, \vec{v}, x, y, \vec{x}, \vec{y})$

$\vec{u}, \vec{x}, x$: Γ A — $\vec{v}, \vec{y}, y$: Δ $A^\perp$ — $\vec{x}, \vec{y}$: Γ Δ

Now given $f : D^{2m+n} \longrightarrow D^{2m+n}$

the permutation $\sigma : D^{2m} \longrightarrow D^{2m}$

$$\sigma(x_1, y_1, \ldots, x_m, y_m) = (y_1, x_1, \ldots, y_m, x_m)$$

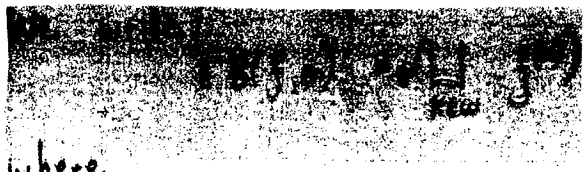
represents the flow of information through the cuts — the "message exchange" function.

We can define

$$FB(f, \sigma) : D^n \longrightarrow D^n$$

$$FB(f, \sigma)(\vec{x}) = \pi \circ Y(\lambda \vec{u}.\, (\sigma \times 1) \circ f(\vec{u}, \vec{x}))$$

where $\pi(\vec{u}, \vec{x}) = \vec{x}$.



where $f^{(k)} : D^n \longrightarrow D^{2m+n}$

$$f^{(0)} = \bot$$

$$f^{(k+1)} = (\sigma \times 1) \circ f \circ \langle \pi' \circ f^{(k)}, 1 \rangle$$

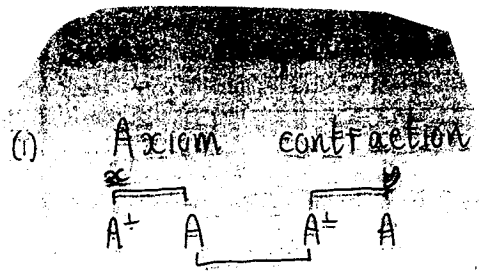
where $\pi'(\tilde{u}, \tilde{x}) = \tilde{u}$.

The idea is that if $\sigma \circ f$ represents a proof π in C.L.L., $FB(f, \sigma)$ will represent any cut-free proof obtained by cut-elimination from π. The dynamics of the cut-elimination is modelled by the sequence of iterations to the fixpoint

$$f^{(0)} \quad f^{(0)} \quad \cdots \quad f^{(k)} \quad \cdots$$

Strong normalization is modelled by the **finite convergence property**:

$$f^{(k)} = f^{(k+1)} \quad \text{for some } k.$$



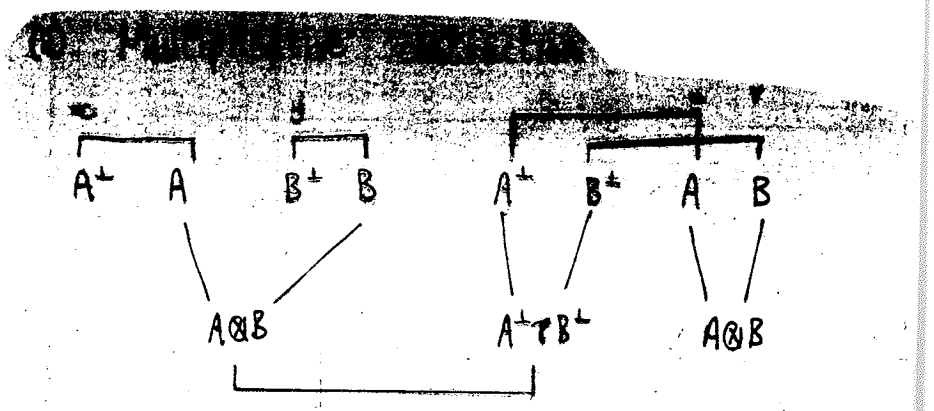
Fixpoint computation	Iteration
$(\bot, \bot, x, y)$	0
$(y, x, \bot, \bot)$	1
(y, x, y, x)	2

so $(x, y) \mapsto (y, x)$

In more detail

$$\sigma \circ f\,(\bot, \bot, x, y) = (y, x, \bot, \bot)$$
$$\sigma \circ f\,(y, x, x, y) = (y, x, y, x)$$

Note that a "fresh copy" of the parameters (x, y) is used at each iteration. This is in fact the key difference between Girard's execution formula and our feedback formula.



Iteration

0 $(\perp, \perp, x, y, m(u,v))$

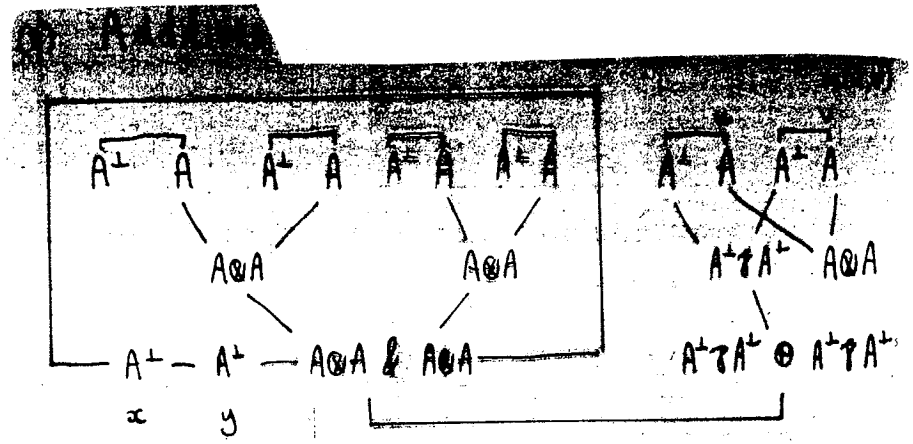
1 $(m(\perp,\perp), m(\perp,\perp), \perp, \perp, m(\perp,\perp))$

2 $(m(u,v), m(x,y), \perp, \perp, m(\perp,\perp))$

3 $(m(u,v), m(x,y), u, v, m(x,y))$

So $(x, y, m(u,v)) \mapsto (u, v, m(x,y))$.

GI-31



Iteration

0 $(\perp, \perp, x, y, m(u,v))$

1 $(\ell(m(\perp,\perp)), \perp, \perp, \perp, m(\perp,\perp))$

2 $(\ell(m(\perp,\perp)), \ell(m(x,y)), \perp, \perp, m(x,y))$

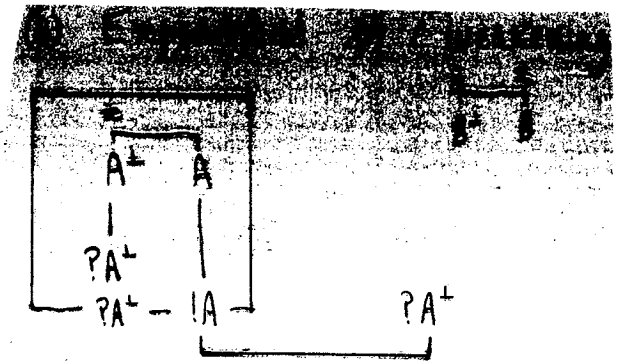
3 $(\ell(m(u,v)), \ell(m(x,y)), \perp, \perp, m(x,y))$

4 $(\ell(m(u,v)), \ell(m(x,y)), u, v, m(x,y))$

So $(x, y, m(u,v)) \mapsto (u, v, m(x,y))$.

Note how synchronization occurs in steps 1 and 2.

GI-32



Iteration

0 $(\bot, \bot, d(x), y, z)$

1 $(w, \bot, d(\bot), z, y)$

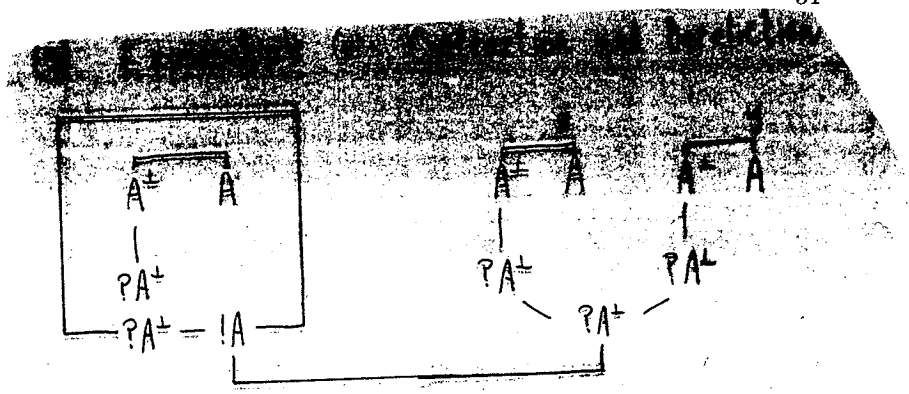
2 (w, w, w, z, y)

3 (w, w, w, z, y)

so $(d(x), y, z) \longmapsto (w, z, y)$

This example illustrates how synchronization affects the context of an Of Course. This is a typical case where Girard's interpretation does not fit the proof theory.

GI-33



Iteration

0 $(\bot, \bot, c(d(u), d(v)), x, y)$

1 $(c(d(\bot), d(\bot)), \bot, c(d(\bot), d(\bot)), \bot, \bot)$

2 $(c(d(\bot), d(\bot)), c(d(u), d(v)), c(d(\bot), d(\bot)), \bot, \bot)$

3 $(c(d(x), d(y)), c(d(u), d(v)), c(d(\bot), d(\bot)), u, v)$

4 $(c(d(x), d(y)), c(d(w), d(v)), c(d(x), d(y)), u, v)$

so $(c(d(w), d(v)), x, y) \longmapsto (c(d(x), d(y)), u, v)$

GI-34

If π is a proof in CLL$_!$, let (f_π, σ_π) be its interpretation.

Theorem

(1) (Soundness) If $\pi \vdash \Gamma$, Γ does not contain &, then if $\pi \longrightarrow\!\!\!\!\!\rightarrow \pi'$,

$$FB(f_\pi, \sigma_\pi) = FB(f_{\pi'}, \sigma_{\pi'})$$

(2) (Convergence)

$$\exists k.\ f_\pi^{(k)} = f_\pi^{(k+1)} = FB(f_\pi, \sigma_\pi)$$

Reason for restriction on (1):

the additive commutative reduction not valid:

$$\&(P,Q)\cdot R \longrightarrow \&(P\cdot R, Q\cdot R)$$

But this is valid at all arguments at principal door of the form $l(x)$ or $r(y)$.

So we want to capture the idea of the right "shape" of arguments for a given proof.

Given $f: D^\sim \to D^\sim$

define

$$dom(f) = Int\ \{d \in D^\sim \mid f^2 d = d\}$$

$$E(f) = f \restriction dom(f) : D^\sim \to D^\sim$$

Theorem For $D \cdot W^*(\Sigma, \emptyset)$:

any proof π (no restrictions),

$\pi \twoheadrightarrow \pi'$:

$$E(FB(f_\pi, \sigma_\pi)) = E(FB(f_{\pi'}, \sigma_{\pi'}))$$

This is proved via a striking characterization of the interpretations of cut-free proofs which also yields connections with proof nets, coherence space semantics.

Step Assign to π a set of linear term tuples in $W(\Sigma, X)^\wedge$.

$$\mathcal{S}(I) = \{ x, x \mid x \in X \}$$

$$\mathcal{S}(\otimes(P)) = \{ \bar{t}, m(t,u) \mid \bar{t}, t, u \in \mathcal{S}(P) \}$$

$$\mathcal{S}(\&(P,Q)) = \{ \bar{t}, \ell(t) \mid \bar{t}, t \in \mathcal{S}(P) \} \cup \{ \bar{u}, r(u) \mid \bar{u}, u \in \mathcal{S}(Q) \}$$

$$\mathcal{S}(!(P)) = \{ \bar{t}, d(t) \mid \bar{t}, t \in \mathcal{S}(P) \} \cup \{ \bar{w}, w \} \cup \{ c(\bar{t}, \bar{u}), dt, u) \mid \bar{t}, t, \bar{u}, u \in \mathcal{S}(!(P)) \}$$

$$\mathcal{S}(P \cdot Q) = \{ s(\bar{t}, \bar{u}) \mid \exists t, u. \; \bar{t}, t \in \mathcal{S}(P), \; \bar{u}, u \in \mathcal{S}(Q), \; s = mgu(t,u) \}$$

Now given $t \in \mathcal{W}_\Sigma(\Sigma, D)$ with K $\perp$-leaves, define grafting operation:

$t[t_1, \ldots, t_K]$

$t \sqsubseteq u \iff \exists! t_1, \ldots, t_K.\ u = t[t_1, \ldots, t_K]$.

Let $\sigma \in S_K$.

$p_{t,\sigma} : D \rightharpoonup D$

$\mathrm{dom}\ p_{t,\sigma} = \uparrow t = \{u \mid t \sqsubseteq u\}$

$p_{t,\sigma}(u) = t[t_{\sigma(1)}, \ldots, t_{\sigma(k)}],\ u = t[t_1, \ldots, t_k]$

$p_{t,\sigma}$ is a partial automorphism; an involution if σ is.

This extends to sets.

Given family $\{(t_i, \sigma_i)\}_{i \in I}$

with $\uparrow t_i \cap \uparrow t_j = \varnothing \quad (i \neq j)$

Can glue these partial functions together

$$\sum_i p_{t_i,\sigma_i}$$

Example

$\mathcal{S}(\mathrm{twist}) = m(x,y),\ m(y,x)$

$\rightsquigarrow\ m(\perp_1, \perp_2),\ m(\perp_3, \perp_4),\ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}$

$p_{\mathrm{twist}}(m(a,b),\ m(c,d)) = m(d,c),\ m(b,a)$.

[redacted]

(Cf. Genericity or "communication without understanding")

$$E(FB(f_\pi, \sigma_\pi)) = \Sigma \{ p_{\bar{t}} \mid \bar{t} \in \mathcal{S}(\pi$$

[redacted]

- connection with coherence space semantics
- implementation with interaction nets; optimality
- proofs of results: Linear Realizability Algebras, abstract realizability lemma
- relation to Girard's Geometry of Interaction interpretation

GI programme using C*-algebras

- Does not interpret all of Linear Logic

  Restricted form of Soundness Theorem

- Would like a more synthetic account

- No concrete computational interpretation

$$P \begin{cases} \vdash A^\perp, A & \vdash B^\perp, B \\ \vdash A^\perp, B^\perp, B \otimes A \\ \vdash A^\perp \parr B^\perp, B \otimes A \end{cases} \quad Q \begin{cases} \vdash A^\perp, A & \vdash B^\perp, B \\ \vdash B^\perp, A^\perp, A \otimes B \\ \vdash B^\perp \parr A^\perp, A \otimes B \end{cases}$$

$$\vdash A^\perp \parr B^\perp, A \otimes B$$

Proof expressions:

$$P = x \parr y,\ y \otimes x \qquad Q = y' \parr x',\ x' \otimes y'$$

$$P \cdot Q = y \otimes x \perp y' \parr x' \ ;\ x \parr y,\ x' \otimes y'$$

$$y \perp y',\ x \perp x' \ ;\ x \parr y,\ x' \otimes y'$$

$$x \parr y,\ x \otimes y \quad = id_{A \otimes B}$$

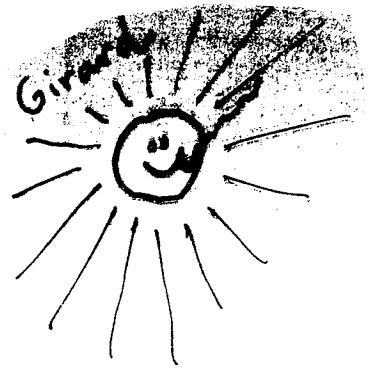
LRA term for P: $\parr^{x;y}_{z}\left(\otimes^{y;z}_{z}\left(\perp_{x;z}, I_{y;y}\right)\right)$

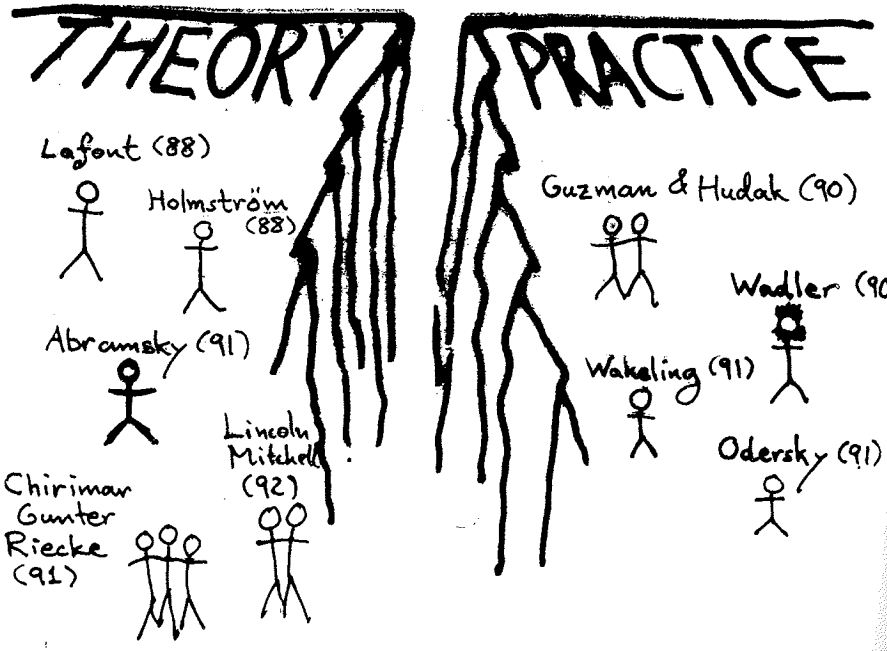
(Need: $P_z\, \perp_{x,y} = P[y/z]$, $\parr^{x;y}_{z}(P) \cdot_a Q = \parr^{x;y}_{z}(P \cdot_a Q)$ to do this calculation.)

There's no substitue for linear logic

Philip Wadler

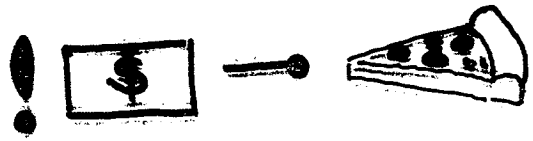
University of Glasgow





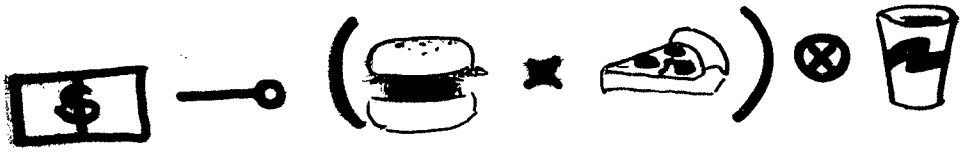
Linear types

Dereliction



Promotion

An isomorphism



Linear logic: Semantics

Monoidal closed category

$$\frac{\Gamma \otimes A \xrightarrow{b} B}{\Gamma \xrightarrow{curry(b)} (A \multimap B)} \qquad (A \multimap B) \otimes A \xrightarrow{apply} B$$

Comonad

$$\frac{!\Gamma \xrightarrow{a} A}{!\Gamma \xrightarrow{kleisli(a)} !A} \qquad !A \xrightarrow{counit} A$$

Girard category (Seely)

$$I \simeq !1 \qquad !A \otimes !B \simeq !(A \times B)$$

$$\begin{array}{ccccc} I & \xleftarrow{discard} & !A & \xrightarrow{duplicate} & !A \otimes !A \\ \wr\downarrow & & \downarrow & & \downarrow\wr \\ !1 & \xleftarrow[!terminal]{} & !A & \xrightarrow[!diagonal]{} & !(A \times A) \end{array}$$

A comonad refresher

Comonad determines a functor

$$\frac{\dfrac{A \xrightarrow{f} B}{!A \xrightarrow{counit;f} B}}{!A \xrightarrow{kleisli(counit;f)} !B}$$

Comonad equations

$$\begin{aligned} kleisli(counit) &= id \\ kleisli(a); counit &= a \\ kleisli(a); kleisli(b) &= kleisli(kleisli(a); b) \end{aligned}$$

Linear logic: Structural rules, $\multimap$ rules

Id

$$\frac{}{x : A \vdash x : A} \qquad \frac{}{A \xrightarrow{id} A}$$

Exchange

$$\frac{\Gamma, x : A, y : B, \Delta \vdash c : C}{\Gamma, y : B, x : A, \Delta \vdash c : C} \qquad \frac{\Gamma \otimes A \otimes B \otimes \Delta \xrightarrow{c} C}{\Gamma \otimes B \otimes A \otimes \Delta \simeq \Gamma \otimes A \otimes B \otimes \Delta \xrightarrow{c} C}$$

$\multimap$-Elimination

$$\frac{\Gamma \vdash f : (A \multimap B) \quad \Delta \vdash a : A}{\Gamma, \Delta \vdash (f\ a) : B} \qquad \frac{\Gamma \xrightarrow{f} (A \multimap B) \quad \Delta \xrightarrow{a} A}{\Gamma \otimes \Delta \xrightarrow{f \otimes a} (A \multimap B) \otimes A \xrightarrow{apply} B}$$

$\multimap$-Introduction

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash (\lambda x : A.\, b) : (A \multimap B)} \qquad \frac{\Gamma \otimes A \xrightarrow{b} B}{\Gamma \xrightarrow{curry(b)} (A \multimap B)}$$

Linear logic: ! rules

Weakening

$$\frac{\Delta \vdash e : !A \quad \Gamma \vdash b : B}{\Gamma, \Delta \vdash (\text{let } \_ = e \text{ in } b) : B} \qquad \frac{\Delta \xrightarrow{e} !A \quad \Gamma \xrightarrow{b} B}{\Gamma \otimes \Delta \xrightarrow{id \otimes e} \Gamma \otimes !A \xrightarrow{id \otimes discard} \Gamma \otimes I \simeq \Gamma \xrightarrow{b} B}$$

Contraction

$$\frac{\Delta \vdash e : !A \quad \Gamma, x : !A, y : !A \vdash b : B}{\Gamma, \Delta \vdash (\text{let } (x \otimes y) = e \text{ in } b) : B} \qquad \frac{\Delta \xrightarrow{c} !A \quad \Gamma \otimes !A \otimes !A \xrightarrow{b} B}{\Gamma \otimes \Delta \xrightarrow{id \otimes e} \Gamma \otimes !A \xrightarrow{id \otimes duplicate} \Gamma \otimes !A \otimes !A \xrightarrow{b} B}$$

Dereliction

$$\frac{\Delta \vdash e : !A \quad \Gamma, x : A \vdash b : B}{\Gamma, \Delta \vdash (\text{let } (!x) = e \text{ in } b) : B} \qquad \frac{\Delta \xrightarrow{e} !A \quad \Gamma \otimes A \xrightarrow{b} B}{\Gamma \otimes \Delta \xrightarrow{id \otimes e} \Gamma \otimes !A \xrightarrow{id \otimes counit} \Gamma \otimes A \xrightarrow{b} B}$$

Promotion

$$\frac{!\Gamma \vdash a : A}{!\Gamma \vdash (!a) : !A} \qquad \frac{!\Gamma \xrightarrow{a} A}{!\Gamma \xrightarrow{kleisli(a)} !A}$$

Substitution and the Cut rule

'Let' is the natural deduction equivalent of 'Cut':

$$\frac{\Delta \vdash a : A \qquad \Gamma, x : A \vdash b : B}{\Gamma, \Delta \vdash (\mathsf{let}\ x = a\ \mathsf{in}\ b) : B} \qquad \frac{\Delta \xrightarrow{a} A \qquad \Gamma \otimes A \xrightarrow{b} B}{\Gamma \otimes \Delta \xrightarrow{id \otimes a} \Gamma \otimes A \xrightarrow{b} B.}$$

Substitution lemma:

$$\Gamma \otimes \Delta \xrightarrow{[\![b[a/x]]\!]} B \;=\; \Gamma \otimes \Delta \xrightarrow{id \otimes [\![a]]\!]} A \otimes \Delta \xrightarrow{[\![b]]\!]} B,$$

Substitution does not preserve typing

Substitution is well-typed when:

$$\frac{\Delta \vdash a : A \qquad \Gamma, x : A \vdash b : B}{\Gamma, \Delta \vdash b[a/x] : B.}$$

This is **not** always the case:

$$\frac{\Delta \vdash a : !A \qquad !\Gamma, x : !A \vdash !b : !B}{!\Gamma, \Delta \vdash !b[a/x] : B.}$$

Substitution does not preserve semantics

Even when restricted to well-typed instances, the Substitution Lemma does not always hold.

Compare the following

$$y : !!A \vdash (!(\text{let } (!z) = y \text{ in } z)) : !!A \qquad !!A \xrightarrow{kleisli(counit)} !!A$$

$$y : !!A \vdash (\text{let } x = (\text{let } (!z) = y \text{ in } z) \text{ in } (!x)) : !!A \qquad !!A \xrightarrow{counit} !A \xrightarrow{kleisli(id)} !!A.$$

Necessary and sufficient conditions

From preceding example, if Substitution Lemma holds then necessarily

$$!A \underset{counit}{\overset{kleisli(id)}{\rightleftarrows}} !!A$$

Proposition. The identity $counit; kleisli(id) = id$ holds iff every arrow $f : !A \longrightarrow !B$ has the form $f = kleisli(k)$ for some $k : !A \longrightarrow B$.

Proposition. The Substitution Lemma holds (when all terms are well-typed) iff $counit; kleisli(id) = id$.

Read-only array

read : $(!\mathrm{ReadArr} \multimap X^*) \multimap \mathrm{Arr} \multimap X^*$

lookup : $!\mathrm{Ix} \multimap !\mathrm{ReadArr} \multimap !\mathrm{Val}$

* ReadArr cannot appear in X

Array update

alloc : Arr

update : Ix → Val → Arr → Arr

lookup : Ix → Arr → Val

Linear array update

block : $(\mathrm{Arr} \multimap X \otimes \mathrm{Arr}) \multimap X$

update : $!\mathrm{Ix} \multimap !\mathrm{Val} \multimap \mathrm{Arr} \multimap \mathrm{Arr}$

lookup : $!\mathrm{Ix} \multimap \mathrm{Arr} \multimap (!\mathrm{Val} \otimes \mathrm{Arr})$

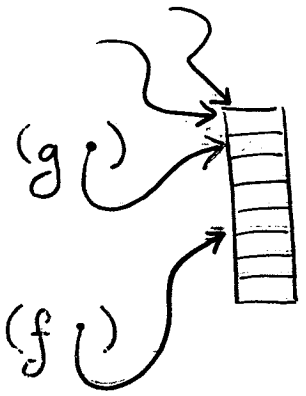
A problem

$f : !(Arr \multimap Arr)$

$f = !(\lambda x{:}Arr.\ update\ i_\circ\ v_\circ\ x)$

$g : !(!Arr \multimap !Arr)$

$g = !(\lambda y{:}!Arr.\ !(\underline{let}\ !x = y\ \underline{in}\ f\,x))$



Approaches to memory

			Update in place	No Copy	Full ILL
Lafont	Linear	Copy on Contract	✓	✗	✓
Lincoln-Mitchell	Two space	Copy on Derelict + Promote	✓	✗	✓
Chirimuuo-Gunter-Riecke	Non-linear	No copy	✗	✓	✓
Steadfast-Apt	Two space	No copy	✓	✓	✗

An apt subset of linear logic

Weakening and Contraction

Old syntax:

$$K = (\lambda x.\,(\lambda y.\,(\text{let } (\_) = y \text{ in } x))),$$
$$W = (\lambda f.\,(\lambda z.\,(\text{let } (x \otimes y) = z \text{ in } ((f\ x)\ y)))).$$

New syntax:

$$K = (\lambda x.\,(\lambda y.\,x)),$$
$$W = (\lambda f.\,(\lambda z.\,((f\ z)\ z))).$$

The usual properties of *discard* and *duplicate* guarantee **coherence**.

Apt linear logic: structural rules

Id

$$\frac{}{x : A \vdash x : A} \qquad \frac{}{A \xrightarrow{id} A}$$

Exchange

$$\frac{\Gamma, x : A, y : B, \Delta \vdash c : C}{\Gamma, y : B, x : A, \Delta \vdash c : C} \qquad \frac{\Gamma \otimes A \otimes B \otimes \Delta \xrightarrow{c} C}{\Gamma \otimes B \otimes A \otimes \Delta \simeq \Gamma \otimes A \otimes B \otimes \Delta \xrightarrow{c} C}$$

Weakening

$$\frac{\Gamma \vdash b : B}{\Gamma, x : !A \vdash : B} \qquad \frac{\Gamma \xrightarrow{b} B}{\Gamma \otimes !A \xrightarrow{id \otimes discard} \Gamma \otimes I \simeq \Gamma \xrightarrow{b} B}$$

Contraction

$$\frac{\Gamma, x : !A, y : !A \vdash b : B}{\Gamma, z : !A \vdash b[z/x, z/y] : B} \qquad \frac{\Gamma \otimes !A \otimes !A \xrightarrow{b} B}{\Gamma \otimes !A \xrightarrow{id \otimes duplicate} \Gamma \otimes !A \otimes !A \xrightarrow{b} B}$$

Apt linear logic: $\multimap$ rules

$\multimap$-Introduction

$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \text{ linear}}{\Gamma \vdash (\lambda x{:}A.\ b)^* : (A \multimap B)}$$

$$\frac{\Gamma \otimes A \xrightarrow{b} B}{\Gamma \xrightarrow{curry(b)} (A \multimap B)}$$

$\multimap$-Elimination

$$\frac{\Gamma \vdash f : (A \multimap B) \quad \Delta \vdash a : A}{\Gamma, \Delta \vdash (f\ a)^* : B}$$

$$\frac{\Gamma \xrightarrow{f} (A \multimap B) \quad \Delta \xrightarrow{a} A}{\Gamma \otimes \Delta \xrightarrow{f \otimes a} (A \multimap B) \otimes A \xrightarrow{apply} B}$$

!$\multimap$-Introduction

$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \text{ non-linear}}{\Gamma \vdash (\lambda x{:}A.\ b) : !(A \multimap B)}$$

$$\frac{\Gamma \otimes A \xrightarrow{b} B}{\Gamma \xrightarrow{kleisli(curry(b))} !(A \multimap B)}$$

!$\multimap$-Elimination

$$\frac{\Gamma \vdash f : !(A \multimap B) \quad \Delta \vdash a : A}{\Gamma, \Delta \vdash (f\ a) : B}$$

$$\frac{\Gamma \xrightarrow{f} (A \multimap B) \quad \Delta \xrightarrow{a} A}{\Gamma \otimes \Delta \xrightarrow{(f;counit) \otimes a} (A \multimap B) \otimes A \xrightarrow{apply} B}$$

The single pointer property

Proposition. In the apt system, a value of linear type has **exactly** one pointer to it.

Active and passive types

The following definitions (almost) were used by Reynolds in 1978:

$$\begin{array}{lll} \text{Active type} & A^{\mathbf{A}}, B^{\mathbf{A}} & ::= X \mid (A^{\mathbf{A}} \multimap B^{\mathbf{A}}) \mid (A^{\mathbf{P}} \multimap B^{\mathbf{A}}), \\ \text{Passive type} & A^{\mathbf{P}}, B^{\mathbf{P}} & ::= !X \mid !(A^{\mathbf{A}} \multimap B^{\mathbf{A}}) \mid !(A^{\mathbf{P}} \multimap B^{\mathbf{P}}). \end{array}$$

A type is *apt* if it is either active or passive.

A context is *apt* if all types in it are apt,
it is *active* if it is apt and linear, and
it is *passive* if it is apt and non-linear.

Substitution preserves well-typing

Proposition. Assume $\Gamma \vdash a : A$ where all types in Γ and a are apt. Then

(i) A is apt,
(ii) Γ is active iff A is active.

Corollary. Substitution is always well typed in the apt system. That is, the following is an admissable typing rule:

$$\frac{\Delta \vdash a : A \qquad \Gamma, x : A \vdash b : B}{\Gamma, \Delta \vdash b[a/x] : B.}$$

Substitution preserves semantics

Proposition. The semantics of a closed term of type $A^{\mathbf{P}} \multimap B^{\mathbf{P}}$ has the form $kleisli(k)$ for some $k : !A \longrightarrow B$.

Corollary. The Substitution Lemma holds in the apt system, so long as the semantics is constrained to allow only arrows of the form $kleisli(k)$ to interpret types of the form $A^{\mathbf{P}} \multimap B^{\mathbf{P}}$.

$$\frac{\Delta \vdash a : A \qquad \Gamma, x : A \vdash b : B}{\Gamma, \Delta \vdash b[a/x] : B} \qquad \frac{\Delta \xrightarrow{a} A \qquad \Gamma \otimes A \xrightarrow{b} B}{\Gamma \otimes \Delta \xrightarrow{id \otimes a} \Gamma \otimes A \xrightarrow{b} B}$$

Question 1.

For all $!\Gamma \xrightarrow{a} A$ and $!\Delta \xrightarrow{b} B$ does there exist a $!(\Gamma \times \Delta) \xrightarrow{c} A \times B$ such that

$$\begin{array}{ccc} !\Gamma \otimes !\Delta & \cong & !(\Gamma \times \Delta) \\ \downarrow{\scriptstyle kleisli(a) \otimes kleisli(b)} & & \downarrow{\scriptstyle kleisli(c)} \\ !A \otimes !B & \cong & !(A \times B) \end{array}$$

commutes ?

Question 2.

Does

$$\begin{array}{ccc} !A & \xrightarrow{duplicate} & !A \otimes !A \\ \downarrow{\scriptstyle kleisli(id)} & & \downarrow{\scriptstyle kleisli(id) \otimes kleisli(id)} \\ !!A & \xrightarrow[duplicate]{} & !!A \otimes !!A \end{array}$$

commute ?

Proposition (Wadler) The two questions are equivalent.

Proposition (Barr) The answers are 'yes'.

Type Systems, Linearity and Functional Languages

Gavin Bierman

University of Cambridge Computer Laboratory

gmb@cl.cam.ac.uk

My Interests

Lazy Functional Languages

...are a good thing

but

...are difficult to implement efficiently

Why?

- Laziness
- Garbage Collection
- Copying of data structures

Optimisations

Previous Approaches

- Transformation — Burstall & Darlington
- Abstract Interpretation — Mycroft
- Exotic Type Systems
  - Effects — Gifford & Lucassen
  - Single-Threading — Guzmán & Hudak
  - **Linear Logic** — Girard & Lafont

Plan of Talk:

1. Linear Functional Languages
2. Linear Logic as a type system
3. My Type system(s)
4. Some operational intuitions
5. Future Work

Logics and Term Calculi

- "Curry-Howard Correspondence"

Basic Idea: Relates

Propositions	and	Types
Proofs	and	Programs
Normalisation	and	Evaluation to HNF

In particular, it relates

Intuitionistic Logic to Lambda Calculus

(Intuitionistic) Linear Logic

- Logic of "resources"

CHIM gives a *Linear Term Calculus*

Can build on top a *Linear Functional Language*

Examples:

- LIVE - Lafont
- Lilac - Mackie
- Linear ML - Gunter

$$\text{Id}\ \frac{}{x:\sigma \vdash x:\sigma} \qquad \text{Cut}\ \frac{\Gamma \vdash t:A \qquad x:A,\Delta \vdash u:B}{\Gamma,\Delta \vdash u[t/x]:B}$$

$$\text{1-R}\ \frac{}{\vdash * : 1} \qquad \text{1-L}\ \frac{\Gamma \vdash t:A}{\Gamma, z:1 \vdash \mathsf{let}\ z\ \mathsf{be}\ *\ \mathsf{in}\ t:A}$$

$$\otimes\text{-R}\ \frac{\Gamma \vdash t:A \qquad \Delta \vdash u:B}{\Gamma,\Delta \vdash t \otimes u : A \otimes B} \qquad \otimes\text{-L}\ \frac{\Gamma, x:A, y:B \vdash t:C}{\Gamma, z:A\otimes B \vdash \mathsf{let}\ z\ \mathsf{be}\ x\otimes y\ \mathsf{in}\ t:C}$$

$$\multimap\text{-R}\ \frac{\Gamma, x:A \vdash t:B}{\Gamma \vdash \lambda x.t : A \multimap B} \qquad \multimap\text{-L}\ \frac{\Gamma \vdash t:A \qquad x:B,\Delta \vdash u:C}{\Gamma, f:A\multimap B, \Delta \vdash u[(ft)/x]:C}$$

$$\&\text{-R}\ \frac{\Gamma \vdash t:A \qquad \Gamma \vdash u:B}{\Gamma \vdash \langle t,u\rangle : A\&B}$$

$$\&\text{-L}\ \frac{\Gamma, x:A \vdash t:C}{\Gamma, z:A\&B \vdash \mathsf{let}\ z\ \mathsf{be}\ \langle x,\_\rangle\ \mathsf{in}\ t:C} \qquad \frac{\Gamma, y:B \vdash t:C}{\Gamma, z:A\&B \vdash \mathsf{let}\ z\ \mathsf{be}\ \langle \_,y\rangle\ \mathsf{in}\ t:C}$$

$$\oplus\text{-R}\ \frac{\Gamma \vdash t:A}{\Gamma \vdash \mathsf{inl}(t) : A\oplus B} \qquad \frac{\Gamma \vdash u:B}{\Gamma \vdash \mathsf{inr}(u) : A\oplus B}$$

$$\oplus\text{-L}\ \frac{\Gamma, x:A \vdash u:C \qquad \Gamma, y:B \vdash v:C}{\Gamma, z:A\oplus B \vdash \mathsf{case}\ z\ \mathsf{of}\ \mathsf{inl}(x) \Rightarrow u \mid \mathsf{inr}(r) \Rightarrow v : C}$$

$$!\text{-R}\ \frac{!\Gamma \vdash t:A}{!\Gamma \vdash !t : !A}$$

$$\text{Dereliction}\ \frac{\Gamma, x:A \vdash t:B}{\Gamma, z:!A \vdash \mathsf{let}\ z\ \mathsf{be}\ !x\ \mathsf{in}\ t:B}$$

$$\text{Contraction}\ \frac{\Gamma, x:!A, y:!A \vdash t:B}{\Gamma, z:!A \vdash \mathsf{let}\ z\ \mathsf{be}\ x@y\ \mathsf{in}\ t:B}$$

$$\text{Weakening}\ \frac{\Gamma \vdash t:B}{\Gamma, z:!A \vdash \mathsf{let}\ z\ \mathsf{be}\ \_\ \mathsf{in}\ t:B}$$

Advantages:

- Simple Strictness Analyser
- No garbage collector
- Linear data structures → no copying

Disadvantages:

1. Difficult to program in:

true $= \lambda x.\lambda y.$*let* y *be* $-$ *in* $x : \alpha \multimap !\beta \multimap \alpha$

false $= \lambda x.\lambda y.$*let* x *be* $-$ *in* $y : !\alpha \multimap \beta \multimap \beta$

but

if ... *then* **true** *else* **false**

does *not* have a type. (Lilac)

At best it has the type $!\alpha \multimap !\alpha \multimap !\alpha$

Also consider:

S $=$ $\lambda f.\lambda g.\lambda x.$*let* x *be* $y \otimes z$ *in* $f\ y(g\ z)$
$: (!\alpha \multimap \beta \multimap \gamma) \multimap (!\alpha \multimap \beta) \multimap !\alpha \multimap \gamma$

I $=$ $\lambda x.x : \alpha \multimap \alpha$

But (**S I**) is *not* well-typed.

2. Natural Deduction

- Substitution?
- Subject Reduction?

State of Play

Functional Language	Linear F. Language
compile ↓	↓*compile*
Lambda Calculus	Linear Term Calc.
	translate ⟶

Possible solution: Use Girard's mapping:

$$\alpha^\circ = \alpha$$

$$(\sigma \to \tau)^\circ = !(\sigma^\circ) \multimap \tau^\circ$$

Extend mapping to terms.

Problem:

Makes every function *non-linear* in its arguments.

- No strictness analysis
- Garbage collection still not needed
- Dereliction required for every variable access

Another Problem

- Unnecessary non-linearity

$$\lambda x.\mathit{let}\ x\ \mathit{be}\ y \otimes z\ \mathit{in}\ \mathit{let}\ y\ \mathit{be}\ -\ \mathit{in}\ z : !\alpha \multimap !\alpha$$

and

$$\lambda x.x : \alpha \multimap \alpha$$

- Compare with **B**, **C** and **I** combinators where

$$\mathbf{B} = \mathbf{S(KS)K}$$

$$\mathbf{C} = \mathbf{S(BBS)(KK)}$$

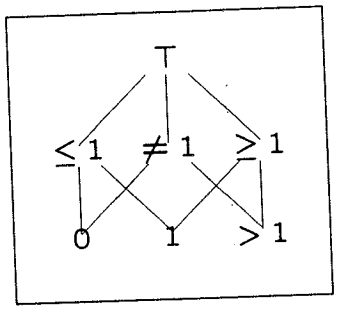
$$\mathbf{I} = \mathbf{SKK}$$

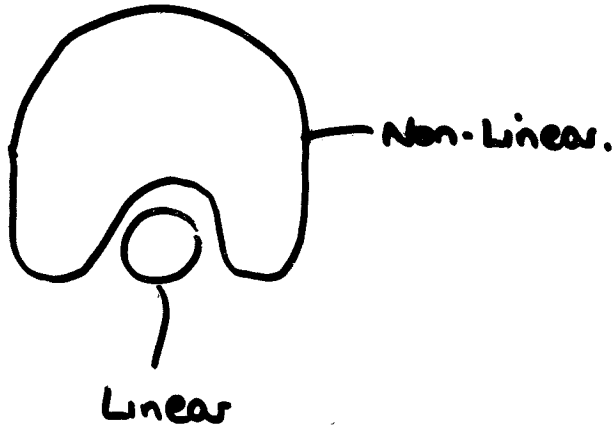
We spilt up the uses:

0 = Discarded

1 = Once

>1 = More than once





Simple Applicative Language:

$$
\begin{array}{rl}
e ::= & k \\
\mid & x \\
\mid & \lambda x.e \\
\mid & e\ f \\
\mid & (e, f) \\
\mid & \textit{case } e \textit{ of } (x, y) \rightarrow f \\
\mid & \textit{if } e \textit{ then } f \textit{ else } g \\
\mid & \textit{fix } e
\end{array}
$$

Type Expressions:

$$
\begin{array}{rl}
\tau ::= & \kappa \\
\mid & \alpha \\
\mid & \sigma \overset{u}{\multimap} \tau \\
\mid & \sigma\ {}_u\!\wedge_v\ \tau
\end{array}
$$

Type Derivations:

$$A \vdash e : \tau$$

where assumption $= x : \tau^a$

$$\text{Id}\ \frac{}{x : \sigma^1 \vdash x : \sigma}$$

$$\text{Dupl}\ \frac{A, y : \sigma^b, z : \sigma^c \vdash e : \tau}{A, x : \sigma^{b+c} \vdash e[x/y, x/z] : \tau}$$

$$\text{Disc}\ \frac{A \vdash e : \tau}{A, x : \sigma^0 \vdash e : \tau}$$

$$\text{Abs}\ \frac{A, x : \sigma^a \vdash e : \tau}{A \vdash \lambda x.e : \sigma \overset{a}{\multimap} \tau}$$

$$\text{App}\ \frac{A \vdash e : \sigma \overset{u}{\multimap} \tau \qquad B^I \vdash f : \sigma}{A, B^{u \star I} \vdash e\ f : \tau}$$

$$\text{Cond}\ \frac{A \vdash e : \textit{bool} \qquad B^I \vdash f : \sigma \qquad B^J \vdash g : \sigma}{A, B^{I \sqcup J} \vdash \textit{if } e \textit{ then } f \textit{ else } g : \sigma}$$

$$\text{Pair}\ \frac{A^I \vdash e : \sigma \qquad B^J \vdash f : \tau}{A^{a \star I}, B^{b \star J} \vdash (e, f) : \sigma\ {}_a\!\wedge_b\ \tau}$$

$$\text{Case}\ \frac{A \vdash f : \sigma\ {}_a\!\wedge_b\ \tau \qquad B, x : \sigma^a, y : \tau^b \vdash e : v}{A, B \vdash \textit{case } f \textit{ of } (x, y) \rightarrow e : v}$$

$$\text{Weak}\ \frac{A, x : \sigma^u \vdash e : \tau}{A, x : \sigma^v \vdash e : \tau}\ \text{where } v \sqsupseteq u$$

Some sample typings:

$\lambda x.\lambda y.x : \alpha \xrightarrow{1} \beta \xrightarrow{0} \alpha$

$\lambda x.(\lambda y.x)x : \alpha \xrightarrow{1} \alpha$

$\lambda f.\lambda x.fx : (\alpha \xrightarrow{>1} \beta) \xrightarrow{1} \alpha \xrightarrow{>1} \beta$

$\lambda f.\lambda x.fx : (\alpha \xrightarrow{0} \beta) \xrightarrow{1} \alpha \xrightarrow{0} \beta$

$\lambda x.x : \alpha \xrightarrow{1} \alpha$

$\lambda x.x : \alpha \xrightarrow{\geq 1} \alpha$

Principal Types

Principal type = Canonical representation of all possible typings

Change Typing derivation:

$$A \vdash e : \tau \parallel C$$

Assumption = $x : \sigma^a$

Constraint = $a \sqsupseteq u$

Example types:

$\lambda x.\lambda y.x : \alpha \xrightarrow{a} \beta \xrightarrow{b} \alpha \parallel a \sqsupseteq 1, b \sqsupseteq 0$

$\lambda x.(\lambda y.x)x : \alpha \xrightarrow{a} \alpha \parallel a \sqsupseteq 1$

$\lambda f.\lambda x.fx : (\alpha \xrightarrow{a} \beta) \xrightarrow{b} \alpha \xrightarrow{c} \beta \parallel b \sqsupseteq 1, c \sqsupseteq a$

Previous typing system, $\vdash_w$

Constraint type system, $\vdash_c$

Theorem

Any proof in $\vdash_w$ is contained in the set of proofs given by $\vdash_c$.

Operational Ideas

Can identify Call-by-Value, Call-by-Name and Call-by-Need

Three Instruction Machine (TIM)

(Fairbairn & Wray, 1987)

Very fast, closure reduction machine for functional languages.

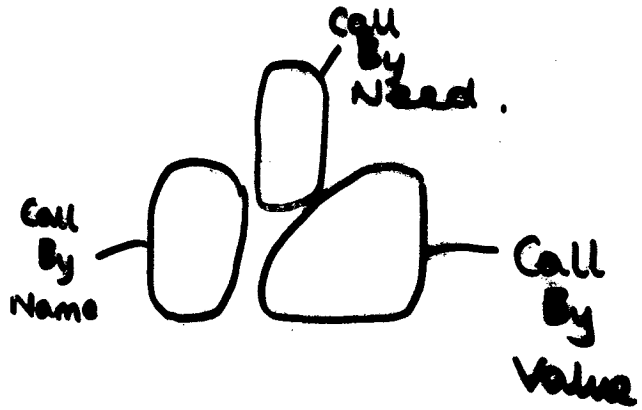
Is a call-by-name machine, adapted to give call-by-need behaviour.

Optimise by *translating Call-by-Need into Call-by-Name.*

(Textual count gives 10% speed-up (Linear Logic?))

Future Work

1. Resource Logic – Bounded Linear Logic
2. Linearity in a denotational set-up
3. Justification of optimisations in an *Operational Semantics* framework
4. Linear Arrays - Single-Threading



DUALITY OF SCHEDULES & AUTOMATA

RATIONALE:
1. CONCURRENCY
2. DUALITY ⇒ COHERENCE
3. SIMPLIFY HILBERT SPACE MODEL → NEW QUANTUM LOGIC
4. NO FIXED NET

SCHEDULE: SPACE OF **EVENTS** WITH A **TEMPORAL** METRIC

AUTOMATON: SPACE OF **STATES** WITH AN **INFORMATION** METRIC

EVENTS **CHANGE** INFORMATION AND **BEAR** TIME

STATES **CHANGE** TIME AND **BEAR** INFORMATION

EVENT ⊥ STATE

TIME ⊥ INFORMATION

SCHEDULE ⊥ AUTOMATON

CONCURRENCY ⊥ NONDETERMINISM :-

	CONC	NONDET.
SCH	+	×
AUT	×	+

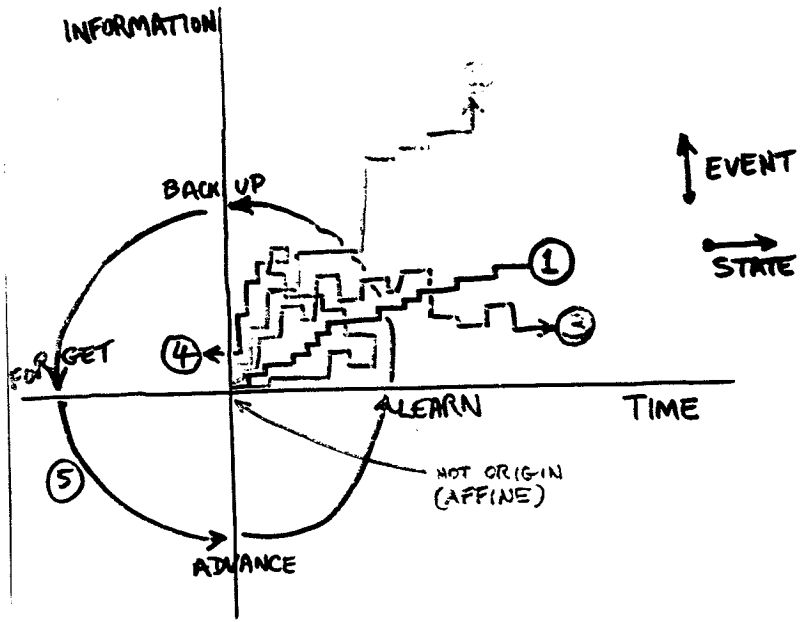
ANALOGIES

LOGIC: TRUE ⊥ FALSE
$\leq \perp \geq$
AND ⊥ OR
THEORY ⊥ CLASS

PHYSICS: PARTICLE ⊥ WAVE
TIME ⊥ ENERGY
POSITION ⊥ MOMENTUM

!! MAXWELL'S DEMON

EXAMPLE PROCESSES



THIS TALK
1. TIME MONOTONIC, INFORMATION MONOTONIC, SLOW
2. " " " " " " , FAST etc

BADLY WANTED
3. " " " , CAN FORGET
4. CAN REVERSE TIME
5. PRIMITIVE ONE-BIT OSCILLATOR:
   LEARN, BACK UP, FORGET, ADVANCE

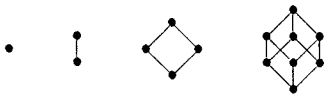
SETS and BOOLEAN ALGEBRAS

SCHEDULES

The 4 sets with up to 3 elements:

AUTOMATA

Their power sets (Boolean algebras):



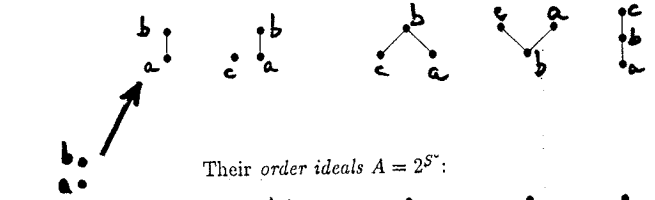
2^{2^S} isomorphic to S !

Reason: $A = 2^S$ contains meets and joins

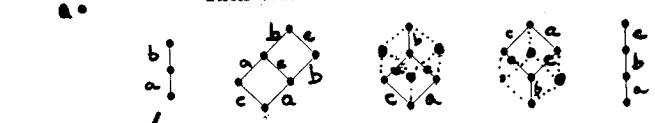
These must be preserved in $S = 2^A$.

POSETS

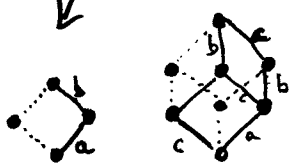
The 5 nondiscrete posets up to 3 elts:



Their *order ideals* $A = 2^{S^\smile}$:



Again $S = 2^{A^\smile}$ ($A^\smile$, not A)

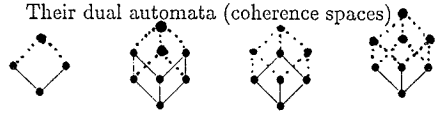


WEBS and COHERENCE SPACES

Already seen the 4 *discrete* webs up to 3 elts.

Nondiscrete webs up to 3 elements:

Their dual automata (coherence spaces)

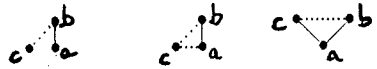


EVENT STRUCTURES

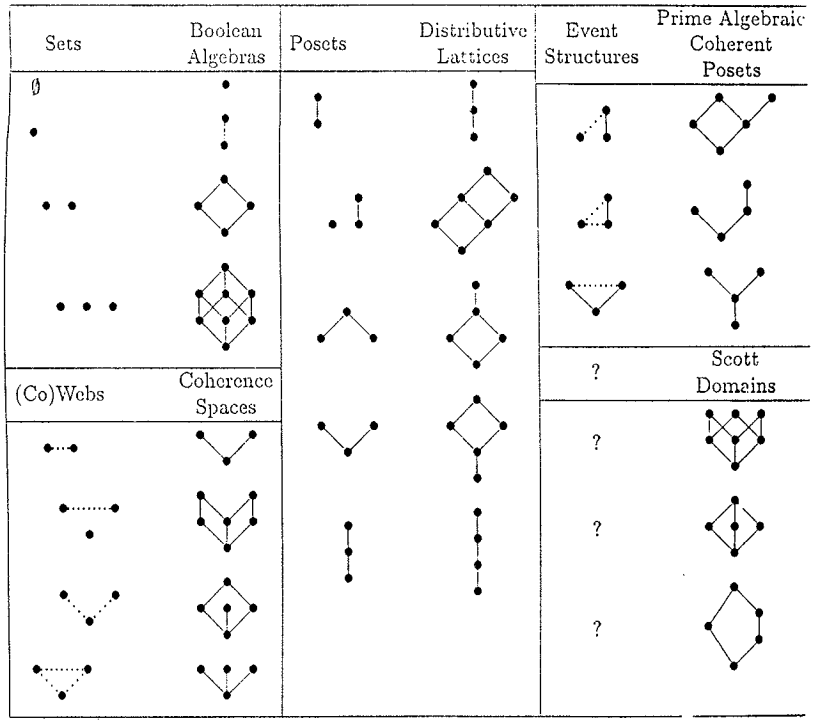
Nielsen-Plotkin-Winskel

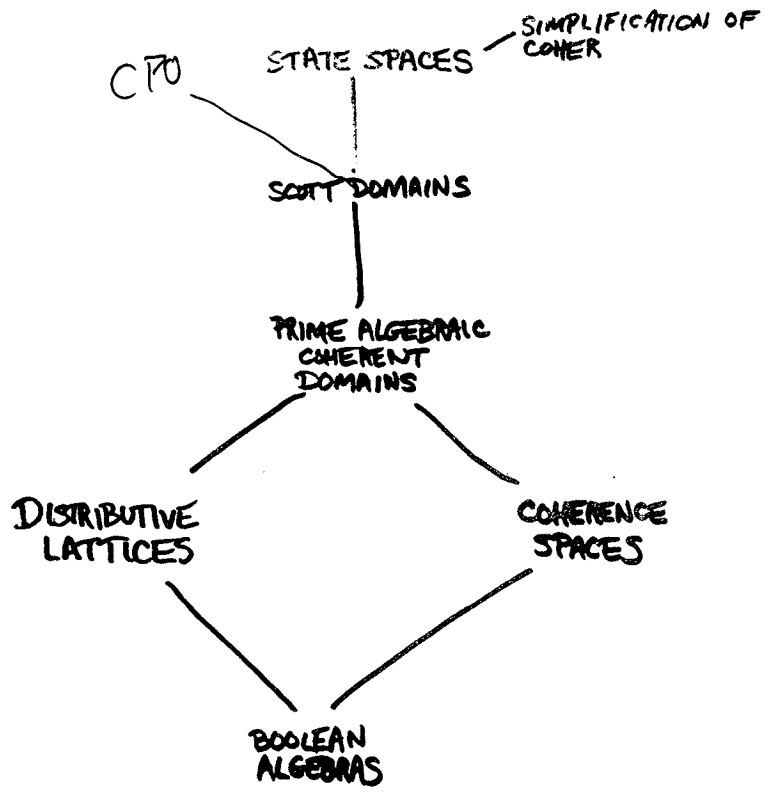
Event structure $(X, \leq, \#)$ is an *ordered web.*

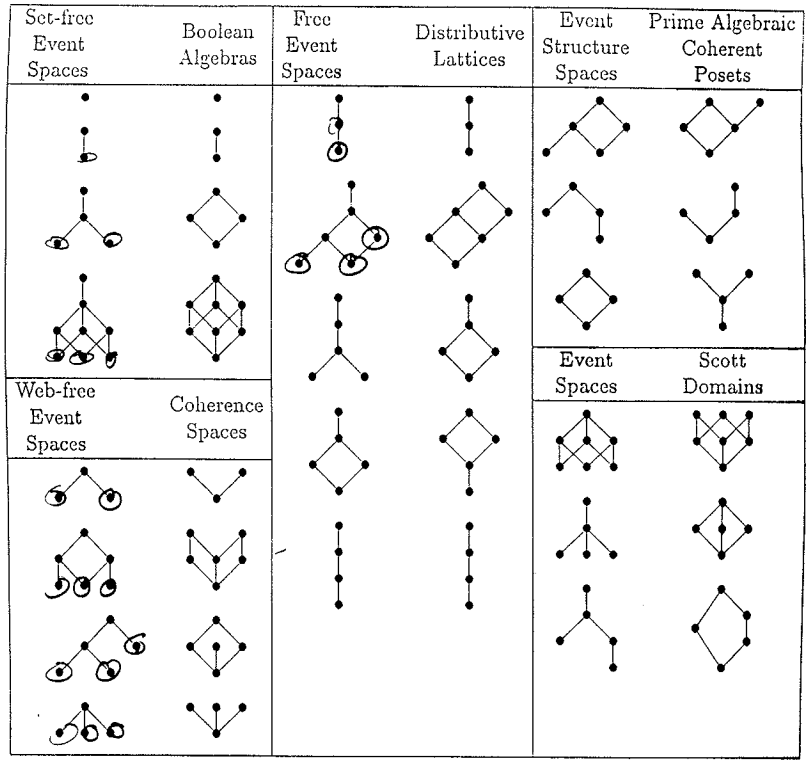
The 3 3-elt event structures not webs or posets:



Their dual automata




CPO
STATE SPACES
SIMPLIFICATION OF COHER
SCOTT DOMAINS
PRIME ALGEBRAIC COHERENT DOMAINS
DISTRIBUTIVE LATTICES
COHERENCE SPACES
BOOLEAN ALGEBRAS


Set-free Event Spaces
Boolean Algebras
Free Event Spaces
Distributive Lattices
Event Structure Spaces
Prime Algebraic Coherent Posets
Web-free Event Spaces
Coherence Spaces
Event Spaces
Scott Domains

Mobility of Bounds

Idea: More bounds on the left means fewer on the right, and vice versa.

Balance 1: Move all joins to schedules, leaving all meets in automata.

Meaning of join in schedule:

concurrent (compound) event

Meaning of meet in automaton:

decision or branch state

This gives *complete semilattices*:

Schedules are complete join semilattices

Automata are complete meet semilattices

Balance 2: Swap empty meet and join.

Schedules become *Event Spaces*

Automata become *State Spaces*

1. Event Spaces and State Spaces

An *Event Space* is a Poset $(X, \vee, \infty)$

where $\vee : (2^X - \emptyset) \to X$ is *Nonempty Join*

and $\infty \in X$ is *Top*

A *State Space* is a Poset $(X, \wedge, q_0)$

where $\wedge : (2^X - \emptyset) \to X$ is *Nonempty Meet*

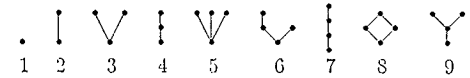
and $q_0 \in X$ is *Bottom*

EXAMPLES

Event Spaces of Order ≤ 4

FREE
SET-FREE

Corresponding State Spaces of Order ≤ 4



5. Maps (= Homomorphisms)

DEF: AN **EVENT MAP** $f: S \to T$ IS A HOMOMORPHISM OF EVENT STRUCTURES S, T

3. $f : (X_S, V_S, \infty_S) \to (X_T, V_T, \infty_T)$

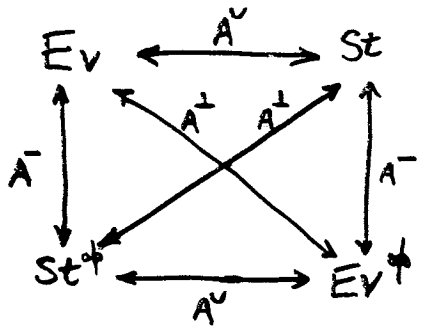
s.t. $f(V_S Y) = V_T f(Y) \quad \emptyset \subset Y \subseteq X$

$f(\infty_S) = \infty_T$

SIMILARLY FOR **STATE MAPS**.

• CATEGORIES Ev AND St.

THREE INVOLUTIONS: $A^\perp, A^\cup, A^-$



↦

$A^\perp = A^{-\cup} = A^{\cup -}$ LINEAR NEGATION (PERP)

$A^\cup = A^{-\perp} = A^{\perp -}$ ORDER DUAL

$A^- = A^{\cup\perp} = A^{\perp\cup}$ "COMPLEMENT" (BY ANALOGY WITH BINARY RELATIONS)

$A^\perp = A \multimap \perp$

$\perp = I$

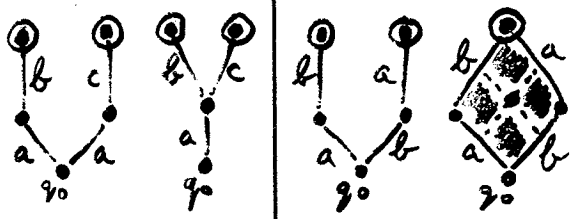
(SEE p.22 AGAIN)

AUTOMATON-SCHEDULE DUALITY

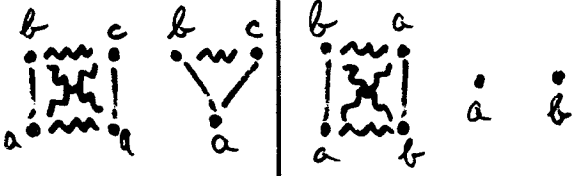
BRANCHING TIME = OR-BRANCHING

TRUE CONCURRENCY = AND-BRANCHING

AUTOMATA (OR-GRAPHS)



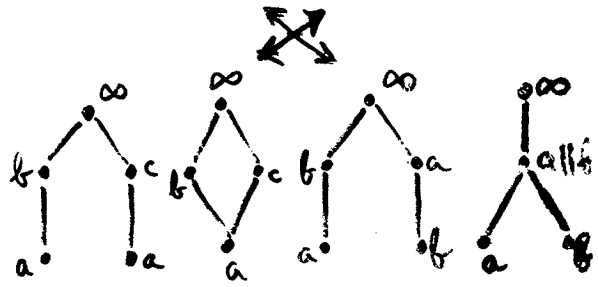
EVENT STRUCTURES (AND-GRAPHS)



x MEANS x PRECEDES y

$x \leq y$

EVENT SPACES



MONADS ON $\mathcal{P}$: Set → Set

$\mathcal{P}$: THE COVARIANT POWER SET FUNCTOR

$$\mathcal{P}X = 2^X$$
$$\mathcal{P}(f : X \to Y)(Z : \mathcal{P}X) = f(Z) : \mathcal{P}Y$$

$\eta : \underset{\text{Set}}{I} \to \mathcal{P}$ FIXED : $\eta_X(x : X) = \{x\} : \mathcal{P}X$

$\mu : \mathcal{P}^2 \to \mathcal{P}$ VARIABLE

THE UNION MONAD : $\mu = \bigcup : \mathcal{P}^2 \to \mathcal{P}$

$$U_X(\mathcal{X} : \mathcal{P}\mathcal{P}X) = \{y \in Y \in \mathcal{P}\mathcal{P}X\} : \mathcal{P}X$$

THE KLEISLI CATEGORY OF $(\mathcal{P}, \eta, U)$ IS Rel, SETS AND BINARY RELATIONS.

MAPS ARE $f : X \to \mathcal{P}Y$, BINARY RELATION FROM X TO Y

COMPOSITION : $X \xrightarrow{f} \mathcal{P}Y \cdot Y \xrightarrow{g} \mathcal{P}Z$

$$= X \xrightarrow{f} \mathcal{P}Y \xrightarrow{\mathcal{P}g} \mathcal{P}\mathcal{P}Z \xrightarrow{U_Z} \mathcal{P}Z$$

EASILY SEEN TO BE ORDINARY RELATION COMPOSITION

THIS IS ESSENCE OF KRIPKE STRUCTURES

THE EILENBERG-MOORE CATEGORY OF $(\mathcal{P}, \eta, U)$ IS CSLat, COMPLETE SEMILATTICES.

HENCE Rel AMOUNTS TO THE SUBCATEGORY OF CSLat CONSISTING OF THE FREE COMPLETE SEMILATTICES (ON A SET).

KLEISLI : $\text{Set} \underset{U}{\overset{F}{\rightleftarrows}} \text{Rel}$

EIL-MOORE : $\text{Set} \underset{U \text{ (MONADIC)}}{\overset{F}{\rightleftarrows}} \text{CSLat}$

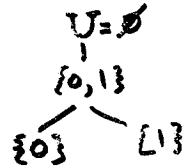
2. THE U-UNION MONAD

$(P, \eta, \hat{U})$

$\hat{U}$ IS U MODIFIED TO TREAT ϕ

AS U SATISFYING $X \subsetneq UX$ FOR ALL $X \neq U(\phi)$

WANT $P\{0,1\} =$



NEED PP TO EXPRESS ORDER (PX IS A SET).

INTUITION: TREAT $\phi : PX$ AS $X \cup \{y\}$ $(y \notin X)$

$\phi : PPX$ AS $\{X \cup \{y\}\}$

THEN TAKE $\hat{U}_x$ TO BE U_x.

FORMALLY: $\hat{U}_x Z = \phi$ if $\phi \in Z$

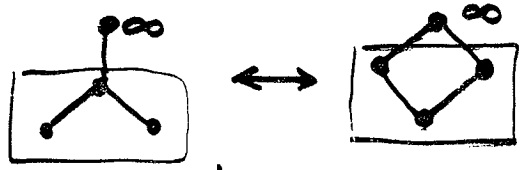
$= U_x Z$ otherwise

THEOREM: $(P, \eta, \hat{U})$ IS A MONAD.

THE KLEISLI CATEGORY IS Rel WITH DIFF'T COMPOSITION

PRIMITIVE OPERATIONS

$a^\perp$: INVERT ALL BUT ∞ :



THEOREM: $a^\perp$ IS AN E.S.

$a\,b$: DIRECT PRODUCT

THEOREM: $a\,b$ IS AN E.S.

$b \multimap a$: SET OF ALL MAPS $f : b \to a$

THAT PRESERVE $\bigvee$ $f(\bigvee Y) = \bigvee f(Y)$

∞ $f(\infty) = \infty$

THEOREM: $b \multimap a$ IS AN E.S.

(ORDER POINTWISE)

$?a$: SET OF ORDER IDEALS OF a

ORDERED BY INCLUSION

THEOREM: $?a$ IS AN E.S.

ARITHMETIC OF EVENT SPACES (= LINEAR LOGIC)

ORDER OF DEFINITION

$a+b$ $\quad a\,b$ (2) $\quad b \Rightarrow a$

$a \oplus b$ $\quad a \otimes b$ $\quad b \multimap a$ (3) $\quad$ (1) $a^\perp = a \multimap \perp$

$?a$ (4) $\quad !a$

GIVEN 1-4, OBTAIN REST THUS:

$$a+b = (b^\perp a^\perp)^\perp$$

$$a \otimes b = (b \multimap a^\perp)^\perp$$

$$a \oplus b = a^\perp \multimap b$$

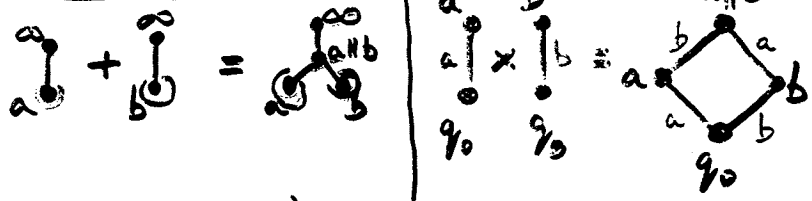
$$!a = (?(a^\perp))^\perp$$

$$b \Rightarrow a = !b \multimap a$$

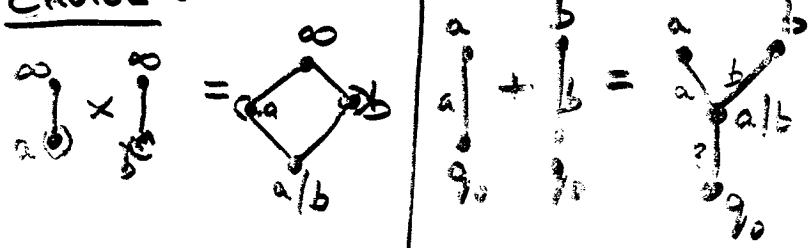
PROGRAMMING WITH EVENT SPACES (PROCESS ALGEBRA)

CONCURRENCE

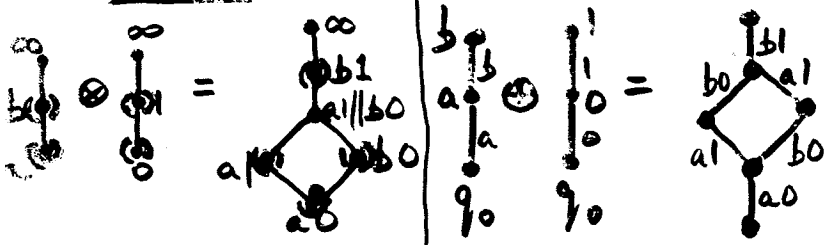
SCHEDULES ← AUTOMATA →



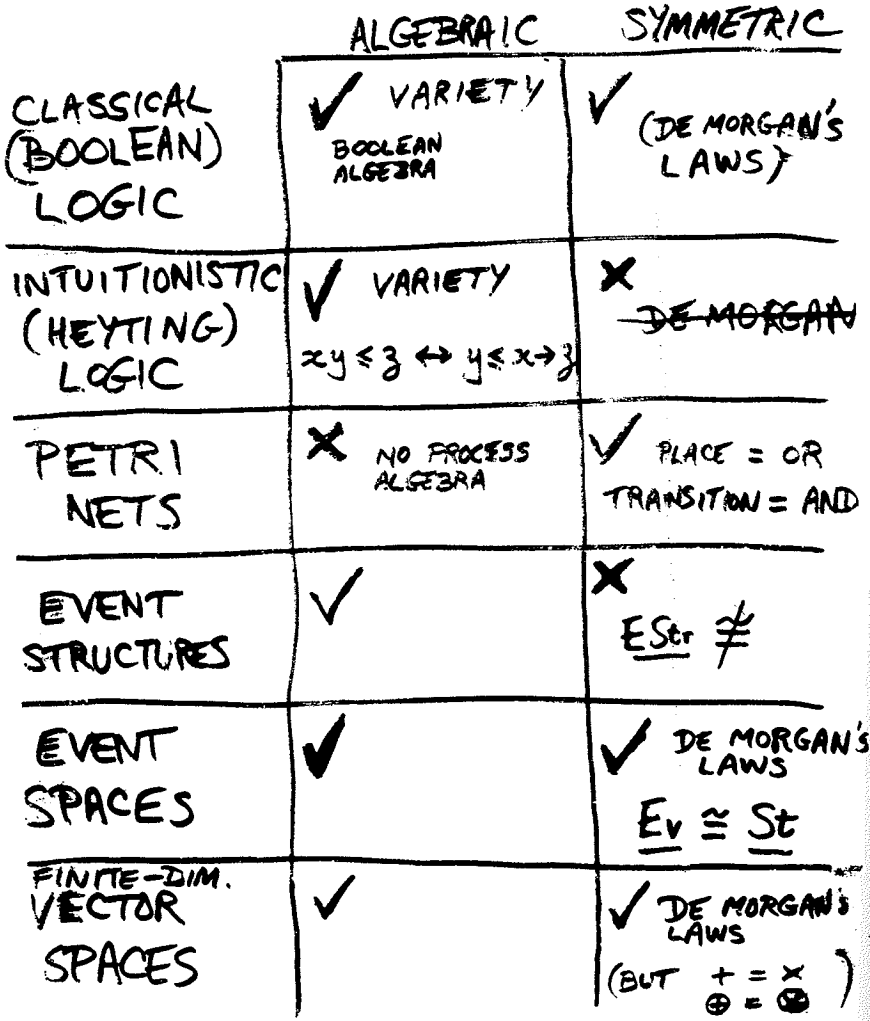
CHOICE (ANGELIC)



FLOW (ORTHOCURRENCE)



(NPW, FIG. 4)



Tail Recursion Through Universal Invariants

C. Barry Jay *
Colloquium
Department of Computer Science
University of Edinburgh

February 27, 1992

Abstract

Tail-recursive constructions suggest a new style of semantics for datatypes, which allows a direct match between specifications and tail-recursive programs. The semantics is expressed in terms of the formal properties of loops, their fixpoints, invariants and convergence. Convergent models of the natural numbers and lists are examined in detail, and, under very mild conditions, are shown to be equivalent to the corresponding initial algebra models.

*Paper to appear in "Theoretical Computer Science". Research supported by The Royal Society of Edinburgh/BP.

Outline

- **Primitive recursion** underlies the initial algebra semantics of datatypes.
- **Tail recursion** is fundamental to programming.
- **Ad hoc translation** of code from primitive-recursion to tail-recursive form arises in optimisation.

Goal To model tail recursion directly.

Benefits

- **Elevates** status of tail recursion to at least that of primitive recursion.
- **Direct translation** of specifications into tail-recursive code.

Disadvantages

- **Harder Theory** resulting from incorporating the ad hoc translations into the formal structure.

Invariants

Consider the imperative program:

$$\texttt{until } b \texttt{ do } f$$

Without loss of generality:

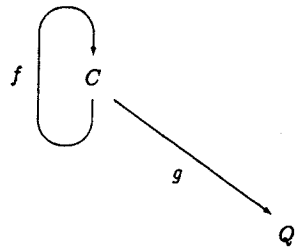
$$b(x) = \texttt{true implies } f(x) = x \quad (x \text{ is fixed by } f)$$

Hence:

$$f;(\texttt{until } b \texttt{ do } f) = \texttt{until } b \texttt{ do } f$$

which is to say that $\texttt{until } b \texttt{ do } f$ is an invariant for f.

Definition 1 *Let f be a loop on C. A function (or morphism) $g : C \to Q$ is an* invariant *for f if*



$$f;g = g$$

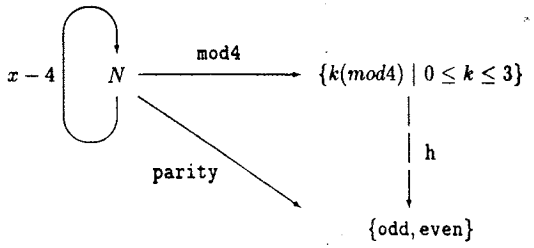
Universal Invariants

The until-loop makes the minimum number of identifications compatible with

$$f(x) \equiv x$$

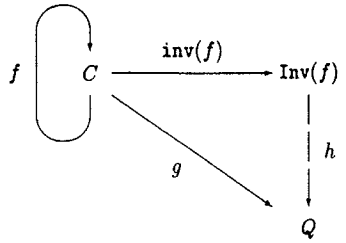
Form the quotient C_0 of C under the corresponding equivalence relation.

Consider the following example: `until` $x < 4$ `do` $x - 4$.



Any invariant for f factors through `mod4` in a unique way. That is `mod4` is the *universal invariant* of the loop.

Definition 2 *A* universal invariant *for a loop f on C is an invariant* $\mathtt{inv}(f) : C \to C_0$ *for f with the property that if $g : C \to Q$ is any other invariant for f then there is a unique $h : C_0 \to Q$ such that $g = \mathtt{inv}(f); h$.*



Convergent Loops

The *control structure* b is to be interpreted as choosing representatives x_0 which satisfy b from the equivalence classes $[x] \in C_0$ i.e. , by a splitting $m : C_0 \to C$ of $\mathtt{inv}(f)$ such that

$$m; b = \mathtt{true}$$

If this can always be done then $f(x) = x$ iff $b(x) = \mathtt{true}$ and m satisfies

$$m; f = f$$

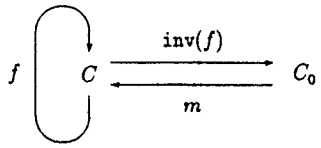
Then we have

$$[\![\mathtt{until}\ b\ \mathtt{do}\ f]\!] = \mathtt{inv}([\![f]\!]); m$$

In our example

$$m(k(mod4)) = k$$

Definition 1 *If a loop f on C has a universal invariant* $\mathtt{inv}(f) : C \to C_0$ *which has a splitting $m : C_0 \to C$ that is fixed by f then f* converges *to the monomorphism $m : C_0 \to C$*



$$m; \mathtt{inv}(f) = \mathtt{id}_{C_0}$$
$$m; f = m$$

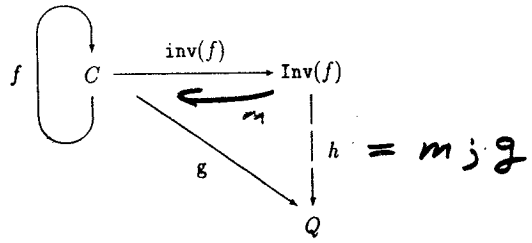
If f converges to a subobject $m : C_0 \to C$ on which b is true then `until` b `do` f converges. The theory of general while-loops on c.p.o.'s is described in

C.B. Jay, Fixpoint and loop constructions as colimits, in: A. Carboni et al (eds), *Category Theory: Proceedings, Como 1990* Lecture Notes in Mathematics 1488 (Springer-Verlag, 1991) 187–192.

25

The Mediating Morphism

When products or sums are defined by a universal property the unique map introduced by the definition (the mediating morphism) is named as the *pairing* or *case* construction. What name should the mediating map out of the universal invariant be called?



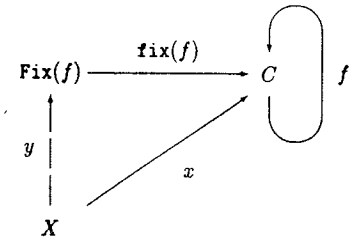
A 5a

Fixed Subobjects

Typically, a convergent loop f converges to its fixed subobject.
An arrow $x : X \to C$ is *fixed* by f if

$$f \circ x = x$$

The universal such arrow, if it exists, is the subobject $\mathtt{fix}(f)$ through which all such x factor.



Its definition is dual to that of $\mathtt{inv}(f)$.

Examples

In **Sets** if a loop converges then it terminates at its fixed subobject.

Theorem 4 *If f is a continuous loop on D in* $\mathbf{Pos}(\omega)$ *(bottomless c.p.o.'s) which is increasing ($x \leq f(x)$) then it converges to its fixed subobject with universal invariant $Yf : D \to \mathtt{Fix}(f)$ defined by*

$$Yf(x) = \bigsqcup_n f^n(x)$$

Proof Clearly Yf is continuous and invariant for f and $Yf \circ \mathtt{fix}(f) = \mathtt{id}$. If $g : D \to Q$ is an invariant for f then $g(f^n(x)) = g(x)$ and so

$$\begin{aligned} g(\mathtt{fix}(f)(Yf(x))) &= g(\bigsqcup_n f^n(x)) \\ &= \bigsqcup_n g(f^n(x)) \\ &= \bigsqcup_n g(x) = g(x) \end{aligned}$$

□

In general $f^n(x)$ will never be fixed by f so that, although f converges, it need not *terminate*.

Primitive Recursion

The primitive recursive functions form the smallest set of functions closed under:

- The zero function $\mathtt{zero} : N \to N$
- The successor function $\mathtt{succ} : N \to N$
- The projections $\pi_i^k : N^k \to N$
- Substitution: if $h : N^k \to N$ and each $g_i : N^n \to N$ for $1 \leq i \leq k$ are primitive recursive then
$$h(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$$
is primitive recursive.
- Primitive Recursion: if $g : N^k \to N$ and $f : N \times N^k \times N \to N$ are primitive recursive then $h : N \times N^k \to N$ defined by
$$\begin{aligned} h(0, y_1, y_2, \ldots, y_k) &= g(y_1, y_2, \ldots, y_k) \\ h(sn, y_1, y_2, \ldots, y_k) &= f(n, y_1, y_2, \ldots, y_n, h(n, y_1, y_2, \ldots, y_k)) \end{aligned}$$
is primitive recursive.

We only *need* one copy of the natural numbers.

- Primitive Recursion(II): if $g : B \to C$ and $f : N \times B \times C \to C$ are *any* functions then there is a $h : N \times B \to C$ defined by
$$\begin{aligned} h(0, y) &= g(y) \\ h(sn, y) &= f(n, y, h(n, y)) \end{aligned}$$
If g and f are primitive recursive then so is h.

Replacing C by $N \times B \times C$ and simplify again to

- Primitive Recursion(III): if $g : B \to C$ and $f : C \to C$ are *any* functions then there is an $h : N \times B \to C$ defined by
$$\begin{aligned} h(0, y) &= g(y) \\ h(sn, y) &= f(h(n, y)) \end{aligned}$$
If g and f are primitive recursive then so is h.

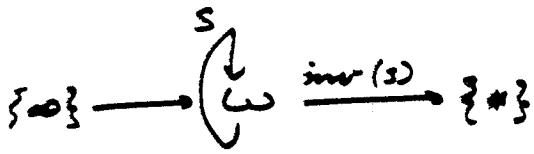
Without using elements this says:

$$\begin{aligned} h \circ (0 \times B) &= g \\ h \circ (s \times B) &= f \circ h \end{aligned}$$

Convergent $\not\Rightarrow$ Terminating

$\omega = 0 < 1 < 2 < \ldots < n < n+1 < \ldots < \infty$

in $Pos(\omega)$



$S(n) = n+1$

$S(\infty) = \infty$

$inv(S)(\infty) = \bigsqcup inv(S)(n)$

$= \bigsqcup *$

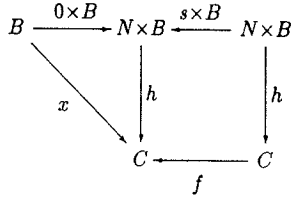
$= *$

Initial Natural Numbers

Definition 5 *An* initial natural numbers object $(N, 0, s)$ *is given by*

$$1 \xrightarrow{0} N \xleftarrow{s} N$$

which has the following universal property: for all morphisms $x : B \to C$ and loops f on C there is a unique morphism $h = \mathtt{It}(x, f) : N \times B \to C$ called the iterator of *x and f making the following diagram commute.*



- Neither B nor C need be powers of N.
- Neither x nor f need be primitive recursive.
- There are categories in which both the above conditions hold.
- Uniqueness of h in recursion theory is a consequence of an induction scheme. Here it is assumed explicitly, as part of a universal property, i.e. initiality.
- In a cartesian closed category the parameter B may be suppressed (replaced by 1) since C can be replaced by $B \to C$.

Hence $(N, 0, s)$ satisfies

$$N \cong 1 + N$$

Any solution of this equation is called a *natural numbers candidate* or NNC.

Tail Recursion

Let's look at those equations for primitive recursion again:

$$\begin{aligned} h(0,z) &= x(z) \\ h(sn,z) &= f(h(n,z)) \end{aligned}$$

Tail recursion modifies the second equation to perform f first:

$$\begin{aligned} h(z,0) &= x(z) \\ h(z,sn) &= h(f(z),n) \end{aligned}$$

This says that h is an invariant for the loop $\mathsf{once}(f)$ on $C{\times}N$ which performs one step of the recursion,

$$\begin{aligned} \mathsf{once}(f)(z,0) &= (z,0) \\ \mathsf{once}(f)(z,sn) &= (f(z),n) \end{aligned}$$

In categorical notation $\mathsf{once}(f)$ is the arrow

$$C{\times}N \cong C{\times}(1+N) \cong (C{\times}1)+(C{\times}N) \xrightarrow{[C{\times}0,\, f{\times}N]} C{\times}N$$

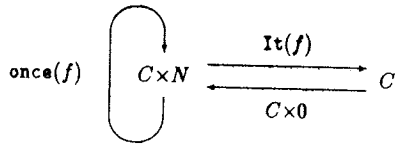
which is defined using the categorical products and coproducts (sums) and the distributive law

$$\begin{aligned} A{\times}(B+C) &\cong (A{\times}B)+(A{\times}C) \\ A{\times}0 &\cong 0 \end{aligned}$$

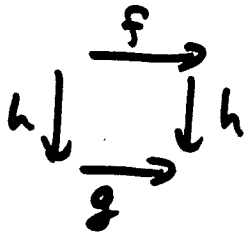
Thus we have a *distributive category.*

Convergent Natural Numbers

Definition 6 *A NNC $(N,0,s)$ is a* convergent *natural numbers object if for every loop f on some object C the loop* $\mathsf{once}(f)$ *converges to $C{\times}0 : C{\to}C{\times}N$. The universal invariant is called the* (convergent) iterator $\mathtt{It}(f) : C{\times}N{\to}C$ *of f.*



Lemma 7 *If $h : C{\to}D$ is a loop morphism from f (on C) to g (on D) then the following diagram commutes*



$$\begin{array}{ccc} C{\times}N & \xrightarrow{\mathtt{It}(f)} & C \\ {\scriptstyle h{\times}N}\downarrow & & \downarrow{\scriptstyle h} \\ D{\times}N & \xrightarrow[\mathtt{It}(g)]{} & D \end{array}$$

Proof Observe that $h{\times}N$ is a loop morphism from $\mathsf{once}(f)$ to $\mathsf{once}(g)$. □

Since f is a loop morphism from f to f we have

$$f \circ \mathtt{It}(f) = \mathtt{It}(f) \circ (f{\times}N) = \mathtt{It}(f) \circ (C{\times}s) : C{\times}N{\to}C$$

which shows that $\mathtt{It}(f)$ is a candidate for the iterator of primitive recursion.

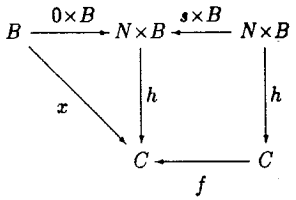
Uniqueness of the Natural Numbers

Theorem 8 *Let $\mathcal{D}$ be a distributive category. Then a NNC $(N, 0, s)$ is initial iff it is convergent. For every loop f on C and $x : B \to C$ the two iterators are related by*

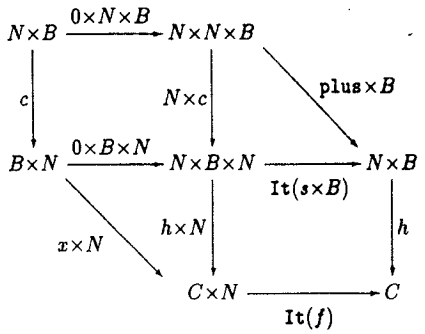
$$\begin{aligned} \mathtt{It}(f) &= \mathtt{It}(C, f) \circ c \\ \mathtt{It}(x, f) &= \mathtt{It}(f) \circ (x \times N) \circ c \end{aligned}$$

Proof We will prove convergent implies initial here.

Let $(N, 0, s)$ be a convergent NNO. Assume now that $h : N \times B \to C$ is a candidate for the iterator of $x : B \to C$ and $f : C \to C$



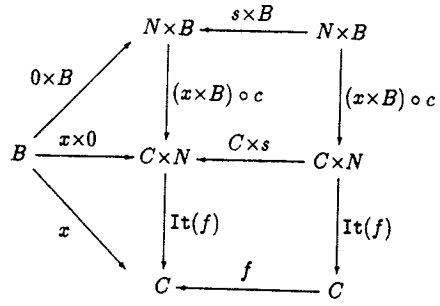
Then h is a loop morphism from $s \times B$ to f.



$$h = \mathtt{It}(f) \circ (x \times N) \circ c$$

Thus if there is an iterator for x and f it is unique.

The adequacy of this choice follows from the commutativity of



□

Corollary 9 *The primitive recursive functions can be represented by a convergent NNO in a distributive category.*

Some Arithmetic Functions

- addition is given by $\mathtt{plus} = \mathtt{It}(s)$.
- truncated subtraction is $\mathtt{subt} = \mathtt{It}(p)$.
- multiplication is $\pi \circ \mathtt{It}(\langle +, \pi' \rangle) \circ \langle 0, N \times N \rangle$.
- the minimum of two naturals is $\pi \circ \mathtt{It}(\mathtt{once}(s)) \circ \langle 0, N \times N \rangle$.
- less-than-or-equals is given by $\mathtt{isZero} \circ \mathtt{subt} : N \times N \to N$.

Convergent Lists

A *list candidate* for A is a solution $(L, \mathtt{nil}, \mathtt{cons})$ of

$$l \cong 1 + (A \times L)$$

A right A-action $\alpha : C \times A \to C$ on C induces a loop $\mathtt{shunt}(\alpha)$ on $C \times L$ by

$$\begin{aligned} \mathtt{shunt}(\alpha)(x, []) &= (x, []) \\ \mathtt{shunt}(\alpha)(x, a :: as) &= (x \oplus a, as) \end{aligned}$$

where $x \oplus a = \alpha(x, a)$.

The list candidate $(L, \mathtt{nil}, \mathtt{cons})$ is *convergent* if every $\mathtt{shunt}(\alpha)$ converges to $C \times \mathtt{nil} : C \to C \times L$. The universal invariant is then *foldleft* of α

$$C \times L \underset{C \times \mathtt{nil}}{\overset{\mathtt{foldl}(\alpha)}{\rightleftarrows}} C$$

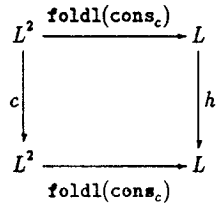
In **Sets** and $\mathbf{Pos}(\omega)$ this is the usual operation *foldleft* on lists. If $x \in C$ and $\alpha(x, a)$ is denoted $x \oplus a$ then

$$\mathtt{foldl}(x, [a_1, a_2, \ldots, a_n]) = (\ldots((x \oplus a_1) \oplus a_2) \ldots) \oplus a_n$$

× 15

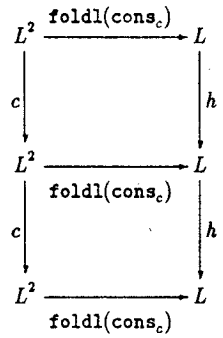
Reverse

Lemma 2 $\mathtt{foldl}(\mathtt{cons}_c) \circ c : L{\times}L{\to}L$ *is an invariant for* $\mathtt{foldl}(\mathtt{cons}_c)$. *Thus there is a unique morphism* $h : L{\to}L$ *making the following diagram commute.*



It is an involution called reverse *and defined by* $\mathtt{rev} = \mathtt{foldl}(\mathtt{cons}_c) \circ (\mathtt{nil}{\times}L)$.

Proof The proof of invariance is a small diagram-chase (or four line proof). Now stack two copies of this square



shows that $h \circ h = \mathtt{id}_L$. Now

$$h = \mathtt{foldl}(\mathtt{cons}_c) \circ c \circ (L{\times}\mathtt{nil}) = \mathtt{foldl}(\mathtt{cons}_c) \circ (\mathtt{nil}{\times}L)$$

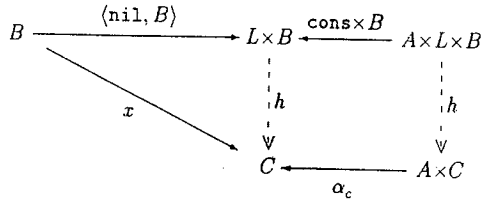
□

Initial Lists

An *initial list object* for A is an $(L, \mathtt{nil}, \mathtt{cons})$ such that for every $x : B{\to}C$ and left A-action

$$\alpha_c = \alpha \circ c : A{\times}C{\to}C$$

there is a unique morphism $h = \mathtt{foldr}(x, \alpha_c) : L{\times}B{\to}C$ which makes the following diagram commute



In Sets and $\mathbf{Pos}(\omega)$ h is *foldright* on lists. For $x \in C$ we have

$$\mathtt{foldr}([a_1, a_2, \ldots, a_n], x) = a_1 \oplus (a_2 \oplus (\ldots (a_n \oplus x) \ldots))$$

Uniqueness of List Objects

Theorem 10 *Let $\mathcal{D}$ be a distributive category. Convergent list objects are initial. Conversely, if there is an initial natural numbers object then initial list objects are convergent. The fold operations are related by*

$$\begin{aligned} \mathtt{foldl}(\alpha) &= \mathtt{foldr}(C, \alpha_c) \circ (\mathrm{rev}' \times C) \circ c : C \times L \to C \\ \mathtt{foldr}(x, \alpha_c) &= \mathtt{foldl}(\alpha) \circ (x \times \mathrm{rev}) \circ c : L \times B \to C \end{aligned}$$

In particular a natural numbers candidate is convergent iff it is initial and the iteratators are related as above.

The proof does not use either projections or diagonals as such, unless they appear as iterators. Thus the proof applies to many categories in which the product is weaker than the usual categorical product, such as categories of partial maps, or categories in which computations $A \to B$ are represented by arrows $A \to TB$ where T is a commutative computational monad. It even applies in models of linear logic, and categories of abelian groups or vector spaces where there simply are not projections or diagonals.

Distributive Monoidal Categori

$(\mathcal{D}, \otimes, +)$

$A \otimes (B+C) \underset{[A\otimes\iota, A\otimes\iota']}{\overset{d}{\rightleftarrows}} (A\otimes B)+(A\otimes C$

Theorem If T is a (commutative) monad on a distributive monoidal category then $F_T : \mathcal{D} \to \mathcal{D}_T$ preserves list objects.

Proof

$$\frac{\alpha : C \otimes A \to TC}{\begin{array}{ccc} TC\otimes A & \xrightarrow{\alpha^+} & TC \\ {\scriptstyle\tau'}\downarrow & & \uparrow{\scriptstyle\mu} \\ T(C\otimes A) & \xrightarrow[T\alpha]{} & T^2C \end{array}}$$

$\mathtt{foldl}(\alpha) = \mathtt{fold}((\alpha^+) \circ (y \times L)$

Domains and Continuous Functions

The results do *not* apply in the category of c.p.o.'s with bottom and all continuous functions. In this category (which is *not* distributive since it lacks sums) there is no inital list object, with respect to either the separated or coalesced sum since the desired fold operation may either fail to be unique or fail to exist (respectiveiy). There is a sense in which the desired list object *is* the convergent list object, however.

If f is a loop on C then the loop $\mathbf{shunt}(\alpha)$ can be defined, and converges to

$$\begin{array}{rcl} C_\perp & \to & C \times L \\ x & \mapsto & (x, \mathtt{nil}) \\ \perp & \mapsto & (\perp, \perp) \end{array}$$

Extensions

These results should apply in an abstract setting to:

- a general class of datatypes.
- full recursion.

Conclusions

The semantics of natural numbers and lists can be constructed directly from tail-recursive intuitions, by using universal invariants. In many categories these new concepts are equivalent to the initial algebra constructions. Where they are known to differ, the tail-recursive approach seems preferable.

This approach allows the specification of tail-recursive progrms directly, without first translating through primitive recursion.

A Calculus for Overloaded Functions with Subtyping

Giuseppe Castagna
Dipartimento d'Informatica dell'Università di Pisa
LIENS(CNRS)-DMI
e-mail: castagna@dmi.ens.fr

Giorgio Ghelli
Dipartimento d'Informatica dell'Università di Pisa
Corso Italia 40, 56100 Pisa, ITALY
e-mail:ghelli@di.unipi.it

Giuseppe Longo
LIENS(CNRS)-DMI, Ecole Normale Supŕieure
45 rue d'Ulm, 75005 Paris, FRANCE
e-mail: longo@dmi.ens.fr

February 1992

1

Object = data + operations

Jargon:
operations (procedures) = methods
data = instance variables

- Objects receive messages at run-time: then a method is selected

Focus also on:
- covariance vs. contravar.
- inheritance vs. subtyping
- no recursion

In the perspective of "methods as global (overloaded) functions".

There are two ways to implement message passing:

• The first way is to consider objects as arrays that associate to each message a method

object	
internal_state	
message_1	method_1
⋮	⋮
message_n	method_n

This first point of view has been extensively studied and corresponds to the "objects as records" analogy [Cardelli88].

• An other way to implement message passing (our approach) is to consider the message as (an identifier for) a function and the receiver as the argument of the function.
Example:

```
[ [myPoint move (3,4)] norm ]
```

`norm` is the message. Viewed as a function, it takes `myPoint` of type `Point`, *after* that this has been moved by (3,4), and returns the norm of `myPoint`.
As `norm` applies to arguments of different types (e.g. dimension), it is an *"overloaded"* function.
In general:

1. A message may behave in a different way on values of different types.

   Messages are (identifiers of) overloaded functions.

2. *Message passing* must act by *call-by-value.*

3. A regular function and its arguments are bound together in the compiled code (compile-time type-checking), while a method and the receiving object are united only at runtime, i.e. during the computation. This tool is called *dynamic-binding*

Circle and *Square* subtypes of *Picture*
draw is a method defined on all of them

$$\lambda x^{Picture} \ldots . \; [x \text{ draw}] \ldots$$

`message_i`	
`class_name_1`	method_1
⋮	⋮
`class_name_n`	method_n

This is the approach used for example in CLOS

The three keyword of our approach:

- overloading
- call-by-value
- dynamic-binding

We look for a λ-calculus WITH TERMS DEPENDING ON TYPES to model them

Some intuition

Overloaded function = finite collection of ordinary functions stuck together to form the different branches

Its type is the set of the types of its branches

Add to λ−terms

$$(M\&N)$$

Add to types

$$\{V_1' \to V_1'', \ldots, V_n' \to V_n''\}$$

$U \leq V \quad \stackrel{def}{\Longleftrightarrow}$ any value of U can be used in the place of a value of V

Given an overloaded function with n branches of type $U_i \to V_i$ if we pass it an argument of type U then we choose the branch j that "*best approximates*" U.

$U_j = \min_{i=1..n}\{U_i | U \leq U_i\}$.

$$(M_1\& \ldots \&M_n)^{\{U_i \to V_i\}_{i=1..n}} \bullet N^U \rhd^* M_j \cdot N$$

The selection is done iff N is a value (to have dynamic binding and call-by-value)

run-time type: used to select branches (to compute)
compile-time type: used to type-check.

An overloaded type $\{U_i \to V_i\}_{i \in I}$ is well-formed if and only if

$\forall i, j \in I$:

$$U_i \leq U_j \Rightarrow V_i \leq V_j \quad (1)$$

$$U_i \Downarrow U_j \Rightarrow \exists \text{ a unique } h \in I \quad U_h = \inf\{U_i, U_j\} \quad (2)$$

Restriction (1)

$M : \{U_1 \to V_1, U_2 \to V_2\}$ with $U_2 < U_1$ and $N : U_1$

the compile-time type of MN is V_1;
if the normal form of N has type U_2 then the run-time type of MN will be V_2 and therefore $V_2 < V_1$ must hold.

Restriction (2)

Condition (2) assure the existence of $\min_{i=1..n}\{U_i | U \leq U_i\}$ during the selection

OO interpretation

(1) When sending a message, the smaller is the class of the receiver the smaller is the type of the output.
Thus messages identify *covariant* families of methods

(2) In case of multiple inheritance methods defined in more than one ancestor must be explicitily redefined.

$$m : \{U_1 \to V_1, U_2 \to V_2\} \quad \text{with } U_3 \leq U_1, U_2$$

then
- U_3 defined by *multiple inheritance* from U_1 and U_2
- m has been defined both in U_1 and U_2

THE λ&-CALCULUS

Pretypes

$$V ::= A \mid V \to V \mid \{V'_1 \to V''_1, \ldots, V'_n \to V''_n\}$$

Subtyping

$$\frac{U_2 \leq U_1 \quad V_1 \leq V_2}{U_1 \to V_1 \leq U_2 \to V_2}$$

$$\frac{\forall i \in I, \exists j \in J \quad U'_j \to V'_j \leq U''_i \to V''_i}{\{U'_j \to V'_j\}_{j \in J} \leq \{U''_i \to V''_i\}_{i \in I}}$$

Types

1. $A \in$ **Types**
2. if $V_1, V_2 \in$ **Types** then $V_1 \to V_2 \in$ **Types**
3. if for all $i, j \in I$
   (a) $(U_i, V_i \in$ **Types**$)$ and
   (b) $(U_i \leq U_j \Rightarrow V_i \leq V_j)$ and
   (c) $(U_i \Downarrow U_j \Rightarrow \exists! h \in I \quad U_h = \inf\{U_i, U_j\})$
   then $\{U_i \to V_i\}_{i \in I} \in$ **Types**

Terms

$$M ::= x^V \mid \lambda x^V M \mid M \cdot M \mid \varepsilon \mid M \&^V M \mid M \bullet M$$

Type-checking Rules

[TAUT] $$x^V : V$$

[→ INTRO] $$\frac{M : V}{\lambda x^U . M : U \to V}$$

[→ ELIM$_{(\leq)}$] $$\frac{M : U \to V \quad N : W \leq U}{M \cdot N : V}$$

[TAUT$_\varepsilon$] $$\varepsilon : \{\}$$

[{}INTRO] $$\frac{M : W_1 \leq \{U_i \to V_i\}_{i \leq (n-1)} \quad N : W_2 \leq U_n \to V_n}{(M \&^{\{U_i \to V_i\}_{i \leq n}} N) : \{U_i \to V_i\}_{i \leq n}}$$

[{}ELIM] $$\frac{M : \{U_i \to V_i\}_{i \in I} \quad N : U \quad U_j = \min_{i \in I}\{U_i | U \leq U_i\}}{M \bullet N : V_j}$$

Operational semantics

β) $(\lambda x^S.M)N \rhd M[x^S := N]$

$\beta_\&$) If $N\colon U$ is closed and in normal form,
$U_j = min\{U_i | U \leq U_i\}$,
$(M_1 \& M_2) : \{U_i \to V_i\}_{i=1..n}$ then

$$(M_1 \& M_2)\bullet N \rhd \begin{cases} M_1 \bullet N & \text{for } j < n \\ M_2 \cdot N & \text{for } j = n \end{cases}$$

Base case

$$(\varepsilon \& M_1 \& \ldots \& M_n)\bullet N \rhd^* M_j \cdot N$$

Main Theorems

- *Generalized Subject Reduction*: Let $M\colon U$. If $M \rhd^* N$ then $N\colon U'$, where $U' \leq U$.
- *Strong Normalization*: There is no infinite sequence of reductions starting form a well-typed term
- *Church-Rosser*: Every term posses a unique normal form.

Deriving records

Records can be encoded.

Take an infinite list of pairwise incomparable atomic types $L_1, L_2, \ldots$ and introduce for each L_i a *constant* $\ell_i\colon L_i$.

1. $\langle\!\langle \ell_1\colon V_1; \ldots; \ell_n\colon V_n \rangle\!\rangle \equiv \{L_1 \to V_1, \ldots, L_n \to V_n\}$
2. $\langle \ell_1 = M_1; \ldots; \ell_n = M_n \rangle \equiv (\varepsilon \ \& \ \lambda x^{L_1}.M_1 \ \& \ldots \& \ \lambda x^{L_n}.M_n)$ with $x^{L_i} \notin FV(M_i)$
3. $M.\ell \equiv M\bullet\ell$

Some examples

Class-name = Atomic type
Class-name = type of the objects of the class.

We associate to a class-name the (record) type of its instance variables

Two classnames are in subtyping relation if this relation has been explicitly declared and it is *feasible* w.r.t the representation types.

Example 1

$$2DPoint \doteq \langle\langle x : \text{Int}; y : \text{Int}\rangle\rangle$$
$$3DPoint \doteq \langle\langle x : \text{Int}; y : \text{Int}; z : \text{Int}\rangle\rangle$$

$$3DPoint \leq 2DPoint$$

$$\begin{aligned} Norm \equiv (\ & \lambda self^{\,2DPoint}.\sqrt{self.x^2 + self.y^2} \\ \& \ & \lambda self^{\,3DPoint}.\sqrt{self.x^2 + self.y^2 + self.z^2} \\ ) & \end{aligned}$$

$$\{2DPoint \to \text{Real}, 3DPoint \to \text{Real}\}$$

- **Covariance**

$$\begin{aligned} Erase \equiv (\ & \lambda self^{\,2DPoint}.\langle x = 0; y = self.y\rangle \\ \& \ & \lambda self^{\,3DPoint}.\langle x = 0; y = self.y; z = self.z\rangle \\ ) & \end{aligned}$$

$$\{2DPoint \to 2DPoint, 3DPoint \to 3DPoint\}$$

- **Multiple inheritance**

$$Color \doteq \langle\langle c : \text{String}\rangle\rangle$$
$$2DColPoint \doteq \langle\langle x : \text{Int}; y : \text{Int}; c : \text{String}\rangle\rangle$$

set $2DColPoint \leq Color$ and $2DColPoint \leq 2DPoint$.

$$\begin{aligned} Erase \equiv (\ & \lambda self^{\,2DPoint}.\langle x = 0; y = self.y\rangle \\ \& \ & \lambda self^{\,3DPoint}.\langle x = 0; y = self.y; z = self.z\rangle \\ \& \ & \lambda self^{\,Color}.\langle c = \text{“white”}\rangle \\ \& \ & \lambda self^{\,2DColPoint}.\langle x = 0; y = self.y; c = \text{“white”}\rangle \\ ) & \end{aligned}$$

$$\begin{aligned} \{\ & 2DPoint \to 2DPoint, \\ & 3DPoint \to 3DPoint, \\ & Color \to Color, \\ & 2DColPoint \to 2DColPoint\} \end{aligned}$$

```
class 2DPoint
  state
    x:Int;
    y:Int
  methods
    Norm = sqrt(self.x^2 + self.y^2);;
    Erase = x <- 0;;
  interface
    Norm: Real;
    Erase: Likeself
endclass


class 3DPoint is 2DPoint and
    state z:Int
endclass


class Color
    state c:String
    methods Erase = c <- "white";;
    interface Erase: Likeself
endclass


class 2DColPoint is Color, 2DPoint and
    methods Erase = x <- 0; c <- "white";;
endclass
```


Definition of Equal

In object as record analogy:

$$
\begin{aligned}
&2DEPoint \equiv \\
&\qquad \langle\!\langle\ x : \text{Int}; \\
&\qquad\quad y : \text{Int}; \\
&\qquad\quad Equal : 2DEPoint \to \text{Bool} \rangle\!\rangle
\end{aligned}
$$

$$
\begin{aligned}
&3DEPoint \equiv \\
&\qquad \langle\!\langle\ x : \text{Int}; \\
&\qquad\quad y : \text{Int}; \\
&\qquad\quad z : \text{Int}; \\
&\qquad\quad Equal : 3DEPoint \to \text{Bool} \rangle\!\rangle
\end{aligned}
$$

$$2DEPoint \not\leq 3DEPoint$$

In our system if we set $3DPoint \leq 2DPoint$ then an equality function,with type:

$$
\begin{aligned}
Equal : \{&2DPoint \to (2DPoint \to \text{Bool}), \\
&3DPoint \to (3DPoint \to \text{Bool})\}
\end{aligned}
$$

would not be well-typed either.

$$
\begin{aligned}
\mathit{Equal} \equiv (\ & \lambda(p,q)^{2DPoint \times 2DPoint}.\ (p.x = q.x)\mathrm{AND} \\
& \qquad\qquad\qquad\qquad (p.y = q.y) \\
& \ \&\ \lambda(p,q)^{3DPoint \times 3DPoint}.(p.x = q.x)\mathrm{AND} \\
& \qquad\qquad\qquad\qquad (p.y = q.y)\mathrm{AND}(p.z = q.z) \\
)&
\end{aligned}
$$

the function above has type:

$\{(\mathit{2DPoint} \times \mathit{2DPoint}) \rightarrow \mathrm{Bool}, (\mathit{3DPoint} \times \mathit{3DPoint}) \rightarrow \mathrm{Bool}\}$

Multiple dispatch

$$
\begin{aligned}
\mathit{Equal} \equiv (\ & \varepsilon\lambda(p,q)^{2DPoint \times 2DPoint}.\ \ \ldots \\
& \ \&\ \lambda(p,q)^{3DPoint \times 3DPoint}.\ \ \ldots \\
& \ \&\ \lambda(p,q)^{2DPoint \times 3DPoint}.(p.x = q.x)\mathrm{AND}(p.y = q.y) \\
& \ \&\ \lambda(p,q)^{3DPoint \times 2DPoint}.(p.x = q.x)\mathrm{AND}(p.y = q.y) \\
& \qquad\qquad\qquad\qquad \mathrm{AND}(p.z = 0) \\
)&
\end{aligned}
$$

Main features

Things which were modeled:

- `self` and `MyClass` (`Likeself`, `Likecurrent`, etc...) without recursion.
- The *Equal* method.
- Multiple dispatch
- The roles of covariance and contravariance in subtyping
- A real type-dependent calculus
- A different way to study and implement object-oriented languages
- The encoding of records
- A type sysem for CLOS

Things that it is possible to model which are not included in the paper:

- Syntactic difference between inheritance and subtyping
- To add a new method to an existing class
- To redefine an existing method without the need of total recompilation and type-checking.
- To encode the full calculus of record values of Cardelli and Mitchell
- modular programming with separate compilation
- `super` to any ancestor of the multiple subtyping hierarchy
- To force an object of a certain class to behave as if it belonged to a superclass

Some of these features need the use of explicit coercions.

Things to do:

- To add recursion.
- To add second order types
- To find out a semantics
- To develop an implementation of an OO-language based on this work.

Semantics of Local Variables

Bob Tennent
LFCS, University of Edinburgh
and
Queen's University, Kingston, Canada

in collaboration with

Peter O'Hearn
Syracuse University, Syracuse, N.Y.

Problem: obtain a suitably abstract semantics for local-variable declarations in an Algol-like language, as in

$$\mathbf{new}\, x{:}\mathbf{int}.\, x := 0\,; \ldots x := x + 1 \ldots$$

Outline:

- Semantical framework
- Traditional approach
- Possible-world approach
- Procedures
- Non-interference
- Realizability

Semantical Framework

Types

$$\theta ::= \mathbf{var} \mid \mathbf{exp} \mid \mathbf{comm} \mid \cdots \mid \theta \to \theta'$$

$\pi = \{\ldots, \iota_i \mapsto \theta_i, \ldots\}$ type assignments

$[\![\mathbf{var}]\!] = \{0, 1, 2, \ldots\}$ locations

$[\![\mathbf{exp}]\!] = S \to V_\perp$ expression meanings

$[\![\mathbf{comm}]\!] = S \to S_\perp$ command meanings

where $S = [\![\mathbf{var}]\!] \to V$ (states), and V is a set of *storable* values.

$[\![\theta \to \theta']\!] = [\![\theta]\!] \to [\![\theta']\!]$ procedures

$[\![\pi]\!] = \prod_{\iota \in \mathrm{dom}\pi} [\![\pi(\iota)]\!]$ environments

Valuations

$$[\![P]\!] : [\![\pi]\!] \to [\![\theta]\!] \text{ when } \pi \vdash P : \theta$$

$$[\![X := E]\!] u\, s = \begin{cases} (s \mid \ell \mapsto v), & \text{if } [\![E]\!] u\, s = v \in V \\ & \text{and } [\![X]\!] u = \ell \\ \perp, & \text{if } [\![E]\!] u\, s = \perp \end{cases}$$

Traditional Approach to Local Variables

Syntax:

$$\frac{(\pi \mid \iota \mapsto \mathbf{var}) \vdash C : \mathbf{comm}}{\pi \vdash \mathbf{new}\, \iota.\, C : \mathbf{comm}}$$

Semantics:

$S = [\![\mathbf{var}]\!] \to (V + \{unused\})$ "marked" states

$$[\![\mathbf{new}\, \iota.\, C]\!] u\, s = \begin{cases} (s'' \mid \ell \mapsto unused), & \text{if } [\![C]\!] u'\, s' = s'' \\ \perp, & \text{otherwise} \end{cases}$$

where $u' = (u \mid \iota \mapsto \ell)$
$s' = (s \mid \ell \mapsto v_0)$
$s(\ell) = unused$
$v_0 \in V$ is a standard initial value

But, e.g., the equivalence

$$\mathbf{new}\, \iota.\, C \equiv C$$

(ι not free in C) fails!

Need to ensure that "unused" locations really *are* unused in the block body. Approaches:

1. determine the "storage support" of semantic entities (Meyer *et al.*);
2. use possible-world semantics, in category theoretic form (Reynolds *et al.*).

Possible-World Semantics

- $w, x, \ldots$ are *possible worlds*.
- $f: w \longrightarrow x$ is a *change* of possible worlds.
- Compatible changes of world are associatively *composable* and, for any world w, there is an *identity* $\mathrm{id}_w: w \longrightarrow w$; i.e., possible worlds and changes make up a *category* $\mathbf{W}$.
- $[\![\theta]\!]w, [\![\pi]\!]w$ are *sets* or *domains* appropriate to world w.
- For any world w, $[\![P]\!]w: [\![\pi]\!]w \longrightarrow [\![\theta]\!]w$ when $\pi \vdash P: \theta$.
- $[\![\theta]\!]f: [\![\theta]\!]w \longrightarrow [\![\theta]\!]x$ and $[\![\pi]\!]f: [\![\pi]\!]w \longrightarrow [\![\pi]\!]x$ are functions induced by f, preserving identities and composites; i.e., $[\![\theta]\!]$ and $[\![\pi]\!]$ are *functors* from $\mathbf{W}$ to $\mathbf{D}$ (or to $\mathbf{S}$).
- For any change $f: w \longrightarrow x$, the diagram

$$\begin{array}{ccc} w & & [\![\pi]\!]w & \xrightarrow{[\![P]\!]w} & [\![\theta]\!]w \\ f\downarrow & & [\![\pi]\!]f\downarrow & & \downarrow[\![\theta]\!]f \\ x & & [\![\pi]\!]x & \xrightarrow{[\![P]\!]x} & [\![\theta]\!]x \end{array}$$

  commutes; i.e., $[\![P]\!]$ is a *natural transformation* from $[\![\pi]\!]$ to $[\![\theta]\!]$.
- Summary: interpret languages in *functor categories* $\mathbf{D}^{\mathbf{W}}$ and $\mathbf{S}^{\mathbf{W}}$ for appropriate categories $\mathbf{W}$ of possible worlds.

Application to Local Variables

Category $\mathbf{W}$:

- worlds are natural numbers (number of locations available)
- changes $f: w \longrightarrow x$ are injective functions from $\{0, 1, 2, \ldots, w-1\}$ to $\{0, 1, 2, \ldots, x-1\}$

$$[\![\mathbf{var}]\!]w = \{0, 1, 2, \ldots, w-1\} \qquad \text{locations}$$

$$[\![\mathbf{exp}]\!]w = S(w) \longrightarrow V_\perp \qquad \text{expression meanings}$$

$$[\![\mathbf{comm}]\!]w = S(w) \longrightarrow S(w)_\perp \qquad \text{command meanings}$$

where $S(w) = [\![\mathbf{var}]\!]w \longrightarrow V$ for any w, and, for any $f: w \longrightarrow x$,

$$[\![\mathbf{var}]\!]f\,\ell = f(\ell)$$

$$[\![\mathbf{exp}]\!]f\,e = \lambda s.\, e(f\,;s)$$

$$[\![\mathbf{comm}]\!]f\,c = \ldots$$

(changes non-local locations like c, leaves local ones invariant)

$$[\![\pi]\!] = \prod_{\iota \in \mathrm{dom}\pi} [\![\pi(\iota)]\!]$$

$$[\![\mathbf{new}\,\iota.\,C]\!]w\,u\,s = \begin{cases} f\,;s'', & \text{if } [\![C]\!]x\,u'\,s' = s'' \\ \perp, & \text{otherwise} \end{cases}$$

where $x = w+1$
$u' = ([\![\pi]\!]f\,u \mid \iota \mapsto w)$
$s' = (s \mid w \mapsto v_0)$
$f: w \longrightarrow x$ is the inclusion

Exponentiation

Functor category $\mathbf{D}^{\mathbf{W}}$ is *cartesian-closed* for *any* small category $\mathbf{W}$. This gives us

- exponentiation in $\mathbf{D}^{\mathbf{W}}$ so that $[\![\theta \to \theta']\!] = [\![\theta]\!] \to [\![\theta']\!]$, and
- interpretations of *abstraction* and *application* that satisfy the laws of the typed λ-calculus, and are *uniformly-defined* for all $\mathbf{W}$.

$(F \to G)(w) \neq F(w) \to G(w)$; in fact, $p \in (F \to G)(w)$ is a *family* of functions indexed by morphisms $f: w \to x$ such that

$$\begin{array}{ccc} F(x) & \xrightarrow{p(f)} & G(x) \\ {\scriptstyle F(g)}\downarrow & & \downarrow{\scriptstyle G(g)} \\ F(y) & \xrightarrow{p(f\,;g)} & G(y) \end{array}$$

commutes for all $f: w \to x$ and $g: x \to y$ in $\mathbf{W}$.

For *any* small category $\mathbf{W}$, $\mathbf{S}^{\mathbf{W}}$ is a *topos*, and this provides an interpretation of logical operators valid for (higher-order) *intuitionistic* theories; furthermore, the following formal law of extensionality is valid:

$$\Big(\forall \iota{:}\theta.\, F(\iota) \equiv G(\iota)\Big) \;\Rightarrow\; F \equiv G,$$

where ι is not free in $F, G: \theta \to \theta'$.

Exponentiation in "Basic" Types

Can we define *basic* types using exponentiation? Consider

$$\begin{aligned} S &= [\![\mathbf{var}]\!] \to V \\ [\![\mathbf{exp}]\!] &= S \to V_\perp \\ [\![\mathbf{comm}]\!] &= S \to S_\perp \end{aligned}$$

where V is a constant functor and $(\cdot)_\perp$ is pointwise lifting.

Problems:

1. $[\![\mathbf{var}]\!] \to V$ gives too many states: commutativity of

$$\begin{array}{ccc} [\![\mathbf{var}]\!]w & \xrightarrow{s(\mathrm{id}_w)} & V \\ {\scriptstyle [\![\mathbf{var}]\!]f}\downarrow & & \downarrow{\scriptstyle \mathrm{id}_V} \\ [\![\mathbf{var}]\!]x & \xrightarrow{s(f)} & V \end{array}$$

   does not constrain $s(f)$ outside the range of $[\![\mathbf{var}]\!]f = f$.

2. Even if defined pointwise, S is naturally *contravariant*:

$$\begin{aligned} S(w) &= [\![\mathbf{var}]\!]w \to V \\ S(f)(s) &= f\,;s. \end{aligned}$$

   But then how can we define *covariant* $[\![\mathbf{exp}]\!]$ and $[\![\mathbf{comm}]\!]$?

Contra-Exponentiation

Define $F \rightarrow G$ from *contravariant* F and G by reversing the vertical arrows in the uniformity condition:

$$\begin{array}{ccc} F(x) & \xrightarrow{p(f)} & G(x) \\ {\scriptstyle F(g)}\uparrow & & \uparrow{\scriptstyle G(g)} \\ F(y) & \xrightarrow{p(f\,;g)} & G(y) \end{array}$$

for all $f : w \rightarrow x$ and $g : x \rightarrow y$, but defining the *morphism part* of $F \rightarrow G$ so it is *covariant*:

$$(F \rightarrow G)(f)(p)(g) \;=\; p(f\,;g)$$

Then $S \rightarrow V_\perp$ is a satisfactory definition of $[\![\mathbf{exp}]\!]$, isomorphic to the pointwise definition used earlier, and similarly for $S = [\![\mathbf{var}]\!] \rightarrow V$.

For $[\![\mathbf{comm}]\!]$, there is a further problem: $S \rightarrow S_\perp$ allows non-local commands to change values of local variables:

$$\begin{array}{ccc} S(w) & \xrightarrow{c(\mathrm{id}_w)} & S(w)_\perp \\ {\scriptstyle S(f)}\uparrow & & \uparrow{\scriptstyle S(f)_\perp} \\ S(x) & \xrightarrow{c(f)} & S(x)_\perp \end{array}$$

Need an apparently *ad hoc* constraint on command meanings.

Procedures

Similar problems arise with procedures; e.g., the equivalence

$$\begin{array}{l} \textbf{new}\ x.\ x := 0; \\ \quad P(x := x + 2); \\ \quad \textbf{if even}\ x\ \textbf{then diverge} \end{array} \quad \equiv \quad \textbf{diverge}$$

(x not free in P:**comm** $\rightarrow$ **comm**) fails, despite the commutativity of

$$\begin{array}{ccc} [\![\mathbf{comm}]\!]w & \xrightarrow{p(\mathrm{id}_w)} & [\![\mathbf{comm}]\!]w \\ {\scriptstyle [\![\mathbf{comm}]\!]f}\downarrow & & \downarrow{\scriptstyle [\![\mathbf{comm}]\!]f} \\ [\![\mathbf{comm}]\!]x & \xrightarrow{p(f)} & [\![\mathbf{comm}]\!]x \end{array}$$

Need to ensure that non-local procedures preserve invariance properties of local variables. Approaches:

1. obtain a sub-model by using logical relations (Meyer and Sieber);
2. obtain stronger uniformity conditions by having more W-maps (Tennent and O'Hearn).

Specification Logic

The "even" equivalence can be *proved* in Reynolds's "specification logic," a multi-sorted intuitionistic first-order theory with atomic formulas of the form

- $\{P\}\,C\,\{Q\}$ (Hoare triple), and
- $C \# E$ (every way of using C preserves any value obtained by using E),

using

Local-Variable Declaration:

$$\frac{\begin{bmatrix} \iota \# E_1, \ldots, \iota \# E_m \\ C_1 \# \iota, \ldots, C_n \# \iota \end{bmatrix} \\ \vdots \\ \{P\}\,C\,\{Q\}}{\{P\}\,\mathbf{new}\,\iota.C\,\{Q\}}$$

(ι not free in assertions P or Q, in phrases E_i or C_j, or in uncancelled assumptions), and

Non-Interference Abstraction:

$$P \# R \;\&\; \{R\}\,C\,\{R\} \;\Rightarrow\; \{R\}\,P(C)\,\{R\}$$

where P: **comm** $\to$ **comm**.

Semantics of Non-Interference

Possible-world semantics, such that

- worlds W, X,... are sets of (allowed) states;
- changes $f\colon W \to X$ include both *expansions* (with $X = W \times V$) and *restrictions* (with $X \subseteq W$).

$[\![\mathbf{exp}]\!] = S \to V_\perp$ (using contra-exponentiation)

$[\![\mathbf{comm}]\!] \prec S \rightharpoonup S$ (constrained *partial* contra-exp.)

$[\![\mathbf{var}]\!] = (V \to [\![\mathbf{comm}]\!]) \times [\![\mathbf{exp}]\!]$ (generalized variables)

For $c \in [\![\mathbf{comm}]\!]W$ and $e \in [\![\mathbf{exp}]\!]W$,

$$c \# e \iff \text{for all } f\colon W \to X \text{ and } s \in X,\ c(f)s = c(f\,;g)s$$

where g is the restriction to

$$\{s' \in X \mid e(f)s' = e(f)s\}.$$

Intuitively, $c \# e$ holds iff any (terminating) execution of c can be restricted to a world for which the value of e is invariant.

Discussion

Non-interference model does *not* solve all problems.

- an equivalence that is valid in the non-interference model and fails in other models:

$$\begin{array}{l}\textbf{new}\, x.\ x := 0;\\ \quad P(x)\end{array} \quad \equiv \quad P(0)$$

(x not free in P:**exp** $\to$ **comm**)

- an equivalence that fails in the non-interference model, but is valid in other models:

$$\begin{array}{l}\textbf{new}\, x.\ x := 0;\\ \quad P(x := x+2)\end{array} \quad \equiv \quad \begin{array}{l}\textbf{new}\, x.\ x := 0;\\ \quad P(x := x+1\,;\, x := x+1)\end{array}$$

(x not free in P:**comm** $\to$ **comm**)

- an equivalence that fails in *all* published models:

$$\begin{array}{l}\textbf{new}\, x.\ x := 0;\\ \quad P(x := x+1)\end{array} \quad \equiv \quad P(\textbf{skip})$$

(x not free in P:**comm** $\to$ **comm**)

Realizability Constraints

Observation: the counter-examples to the equivalences are not just *inexpressible*; they are *unrealizable*.

Conjecture: need to impose uniformity constraints in a realizability framework (c.f., "parametricity" in 2nd-order λ-calculus).

Experiment:

- replace *sets* or *domains* by *pers* (partial equivalence relations)
- require morphism parts of functor objects to be realizable
- require components of indexed products $\Pi_{w \in \mathbf{W}} F(w) \to G(w)$ and $\Pi_{f:w \to x} F(x) \to G(x)$ to be (uniformly) realizable

Results: counter-examples are not realizable and all test equivalences validated. This Per interpretation is currently the best available model of local variables.

Questions:

- How should we deal with recursion?
- Can we prove full abstraction?
- Is there a more abstract form of the realizability constraints, perhaps expressible in the internal language of an appropriate category?

References

R. D. Tennent. Semantical analysis of specification logic. *Information and Computation*, 85(2):135–162, 1990.

P. W. O'Hearn and R. D. Tennent. Semantical analysis of specification logic, part 2. Technical Report 91-304, Department of Computing and Information Science, Queen's University, Kingston, Canada, 1991. To appear in revised form in *Information and Computation.*

P. W. O'Hearn and R. D. Tennent. Semantics of local variables. Technical Report ECS-LFCS-92-192, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1992. To appear in *Applications of Categories in Computer Science* (M. P. Fourman, P. T. Johnstone and A. M. Pitts, editors), the Proceedings of the London Mathematical Society Symposium on Applications of Categories in Computer Science, Durham, England, July 20–30, 1991, to be published by Cambridge University Press.

REMARQUES SUR LA THEORIE DES JEUX A DEUX PERSONNES

par André Joyal
(l'UQAM)

Dans son livre "On numbers and games", John Conway introduit une notion d'équivalence entre des jeux à deux personnes. Il démontre ensuite que les classes de jeux équivalents constituent un groupe abélien ordonné pour une somme convenablement définie. Sa construction est réminiscente de la construction traditionnelle des nombres naturels à partir des ensembles finis, ou encore de celle du groupe de Grothendieck d'une catégorie abélienne. Cependant, Conway ne nous indique pas ce que pourrait être une transformation (un morphisme) entre deux jeux et il manque ainsi d'obtenir une très jolie catégorie des jeux. Nous nous proposons, dans le court texte qui suit, de décrire cette catégorie qui est véritablement un calcul "combinatoire" des stratégies.

Jeu, joueur, partie, gagnant, perdant...

Rappelons qu'un jeu G est essentiellement un "ensemble" muni de deux types de relation d'appartenance:

$$h \overset{L}{\in} G \quad \text{et} \quad g \overset{R}{\in} G.$$

$g \overset{R}{\in} G$ signifiant que g est une position que le joueur de droite (Right) peut prendre s'il ouvre le jeu. De même $h \overset{L}{\in} G$ signifie que h est une position que le joueur de gauche (Left) peut prendre s'il ouvre le jeu.

Il faut compléter cette description en supposant de plus que les positions d'ouvertures g et h sont elles-mêmes des jeux. On ajoute l'hypothèse qu'un jeu est bien fondé: toute chaîne $g \overset{}{\ni} x_1 \ni x_2 \ldots$ de positions est finie peu importe l'alternance des occurences de $\overset{R}{\ni}$ ou de $\overset{L}{\ni}$ dans cette chaîne.

Ces conditions étant satisfaites, on définit une partie comme chaîne de positions $G \ni x_1 \ni x_2 \cdots \ni x_n$ où les occurences de $\overset{R}{\ni}$ et de $\overset{L}{\ni}$ se succèdent en alternance et qui, de plus, soit maximale en ce sens que si la chaîne se termine par $x_{n-1} \overset{L}{\ni} x_n$, alors le joueur de droite ne peut prendre aucune position d'ouverture sur le jeu x_n (car l'ensemble de ces positions est vide); le joueur de droite est alors perdant car le dernier joueur à jouer dans une partie est gagnant. (Les parties qui commencent par $G \overset{R}{\ni} x_1$ sont évidemment les parties ouvertes par le joueur de droite).

On peut écrire $$G = (\{h:\ h \overset{L}{\in} G\},\{g:g \overset{R}{\in} G\})$$

pour indiquer qu'un jeu est entièrement déterminé par ses positions d'ouvertures (de gauche ou de droite).
Cette notation permet de définir

1) le jeu nul $$0 = (\emptyset,\emptyset)$$
sans position d'ouverture. Le premier joueur est immédiatement perdant à ce jeu!

2) L'opposé - G d'un jeu G.
$$-G = (\{-g:\ g \overset{R}{\in} G\},\{-h:h \overset{L}{\in} G\})$$
où sont interverti les rôles des joueurs gauche et droite.

3) La somme H + G de deux jeux
$$H + G = (\{h + G:\ h \overset{L}{\in} H\},\{H+g:\ g \overset{R}{\in} G\})$$

Remarques Les deux dernières définitions sont des définitions inductives: la somme $H + G$ est déterminée si on connait les sommes $h + G$ et $H + g$ etc. Une description moins formelle de la somme: chacun des joueurs a devant lui chacun des jeux H et G ; pour répondre à l'adversaire on choisit d'abord sur lequel des jeux H et

G on jouera, et on prend ensuite sur ce jeu seulement une position qui suit légalement la position actuelle de ce jeu; c'est ensuite le tour de son adversaire de faire de même.

Stratégie gagnante

Une stratégie gagnante est une règle dictant (au joueur qui l'applique dans une partie) le choix d'une position pour répondre à la position choisie par l'adversaire. Le joueur qui applique une stratégie gagnante est gagnant car son adversaire ne peut être le dernier à jouer. Nous noterons $S_o(G)$ l'ensemble des stratégies gagnantes pour le joueur de gauche sur le jeu G lorsque le joueur de droite ouvre le jeu. Un élément $U \in S_o(G)$ est donc une règle permettant de choisir une position $x_{2n-1} \overset{L}{\ni} x_{2n}$ chaque fois que l'on s'est donné une chaîne

$$G \overset{R}{\ni} x_1 \overset{L}{\ni} x_2 \overset{R}{\ni} \cdots \overset{R}{\ni} x_{2n-1}$$

où les positions d'indice pair x_{2r} $1 \le r < n$ ont été choisies antérieurement par application de la règle U.

Nous noterons $S_1(G)$ l'ensemble des stratégies gagnantes pour le joueur de gauche lorsque c'est lui qui ouvre le jeu G.
On a alors

(1) $$S_o(G) \cong \prod_{g \in G}^{R} S_1(g)$$

car le choix d'une stratégie $T \in S_o(G)$ est équivalent au choix d'une stratégie $T_g \in S_1(g)$ pour chaque position $g \overset{R}{\in} G$.

On a aussi

(2) $$S_1(G) \cong \sum_{g \in G}^{L} S_o(g)$$

49

car le choix d'une stratégie $U \in S_1(G)$ est équivalent au choix d'une position $g \overset{L}{\in} G$ et d'une stratégie $h \in S_0(g)$.

Les formules (1) et (2) déterminent alors entièrement $S_0(G)$ et $S_1(G)$ par récurrence (transfini) sur les positions de G.

Calcul des stratégies

Nous allons maintenant décrire une catégorie Y dont les objets seront les jeux à deux personnes. Un morphisme

$$G \xrightarrow{f} H$$

sera une stratégie gagnante pour le joueur de gauche sur le jeu $H - G = H + (-G)$ dans les parties ou c'est le joueur de droite qui ouvre. Un morphisme $f \in Y(G, H)$ est donc une stratégie $f \in S_0(H - G)$.

1) L'identité $G \xrightarrow{1_G} G$ est la stratégie qui consiste (pour le joueur qui joue en second sur $G - G$) à répliquer à tout mouvement sur une composante de $G + (-G)$, le mouvement opposé sur la composante opposée.

2) Stratégie sur la somme $G + H$.

On définit facilement la somme $T + U$ de deux stratégies $T \in S_0(G)$, $U \in S_0(H)$

$$S_0(G) \times S_0(H) \xrightarrow{+} S_0(G + H)$$

3) Stratégie résiduelle.

Nous allons définir une opération

$$S_0(A) \times S_0(B - A) \xrightarrow{*} S_0(B)$$

C'est le point essentiel dans notre texte. Il s'agit de décrire une stratégie $T * U \in S_0(B)$ à partir d'un couple de stratégies $(U, T) \in S_0(A) \times S_0(B - A)$.

50

Supposons que le joueur de droite ouvre une partie sur B. Le joueur de gauche peut très bien considérer que c'est une ouverture sur $B + (-A)$. La stratégie T lui permet alors de répondre sur $B + (-A)$. Deux cas peuvent se produire

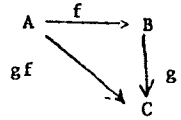
i) La stratégie T lui dicte de répondre sur la composante B du jeu $B + (-A)$. Dans ce cas, il joue sur B et attend que le joueur de droite ait effectué (sur B) le mouvement de son choix.

ii) La stratégie T lui dicte de répondre sur la composante -A du jeu $B + (-A)$. Dans ce cas, la partie serait en suspend si la stratégie -U ne permettait de simuler sur -A un coup de la droite. (C'est bien une simulation car le joueur de droite peut aller jusqu'à ignorer l'existence du jeu $B + (-A)$.)

Ainsi, on voit que la droite (réelle ou simulée, selon le cas) répond sur $B + (-A)$ à la stratégie de gauche T: dans le cas i) elle utilise toute sa liberté et son intelligence; dans le cas ii) c'est la gauche qui simule au moyen de -U sa réaction sur -A. Ceci induit sur B une partie que la droite perd car la gauche gagnera sur $B + (-A)$ et perdra sur -A (puisque U est gagnant sur A) ce qui entraîne qu'elle gagnera sur B. La gauche n'a fait aucun effort, elle ne fait qu'utiliser les ressources des stratégies T et U. C'est donc qu'elle a une stratégie gagnante sur B (que l'on vient de décrire): c'est la stratégie résiduelle $T * U$.

La loi de composition des stratégies

Nous pouvons maintenant définir le composé



des stratégies $f \in S_o(B + (-A))$

$$g \in S_o(C + (-B)).$$

On effectue d'abord la somme

$$g + f \in S_o(C + (-B) + B + (-A))$$

$$\cong S_o(C + (-A) + [(-B) + B])$$

On calcule ensuite la stratégie résiduelle

$$(g + f) * 1_B \in S_o(C + (-A))$$

Théorème

1) La loi de composition des stratégies détermine une structure de catégorie Y.

2) La somme G + H est un bifoncteur associatif, commutatif avec objet neutre 0.

3) Pour tout jeu G le foncteur

$$G + (-): \quad Y \longrightarrow Y$$

est adjoint à gauche au foncteur

$$-G + (-): \quad Y \longrightarrow Y.$$

En resumé, la catégorie Y est monoïdale, fermé, symetrique et autoduale: exactement comme la catégorie des espaces vectoriels de dimension fini sur un corps K. On a la table suivante de concepts correspondants:

0	→	K
G + H	→	$V \otimes_K W$
-G	→	V^*
Hom(G,H)	→	$Hom_K(V,W)$

et les identités:

$$Hom(G,H) \xrightarrow{\sim} H-G \quad \to \quad Hom_K(V,W) \xrightarrow{\sim} W \otimes_K V^*$$

$$Hom(G+H,L) \xrightarrow{\sim} Hom(G,\ Hom(H,L)$$

$$\to \quad Hom_K(V \otimes_K W,\ Z) \simeq Hom_K(V,\ Hom_K(W,Z))$$

etc.

On peut ensuite procéder comme Conway: Deux jeux H et G sont équivalents s'il existe des flèches $H \xrightarrow{f} G$ et $G \xrightarrow{g} H$. C'est évidemment une relation d'équivalence. Les classes de jeux équivalents constituent un groupe abélien sous la somme

$$[H] + [G] = [H + G]$$

$$[H] + [0] = H$$

$$[H] + [-H] = 0$$

C'est même un groupe abélien ordonné car on peut définir une relation d'ordre: $[H] \leq [G]$ s'il existe une flèche $H \xrightarrow{f} G$. C'est ce groupe abélien qui contient les nombres surréels ...

ACTIVITES PREVUES

CONGRES, RENCONTRES...

Séminaire sur la théorie des singularités (25 avril - 25 mai 1977, UQAR)

M. Jean Martinet (dépt. de math. Univ. de Strasbourg) donnera trois conférences par semaine sur la théorie des singularités: applications différentielles, formes différentielles et équations différentielles.

Pour plus d'information communiquer avec Jean-Guy Dubois de l'UQAR ((418)-723-1986).

Les Journées de l'Optimisation 1977 (les 5 et 6 mai 1977, Université Concordia, Montréal)

Cette année les Journées de l'optimisation porteront sur la théorie de l'optimisation, la programmation linéaire, non-linéaire et en nombres entiers, les méthodes combinatoires, la théorie de la commande optimale, la théorie des jeux, et sur les applications aux problèmes du génie, de gestion, de transport, de l'économie, de l'environnement, des ressources, de l'aménagement, de la biologie, etc...

Les conférenciers suivants ont accepté l'invitation du comité d'organisation:

Michael ATHANS, Massachussetts Institute of Technology

Jean-Pierre AUBIN, Université de Paris IX (présentement en congé à l'Université de Montréal)

THIERRY (OQUAND) ①

. Intuitionistic analysis of classical logic

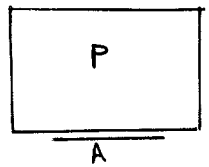
. Analysis of the "computational content" of classical proofs

. Motivated by the first consistency proof of arithmetic by Gentzen (1936)

" Thus propositions of actualist mathematics seem to have a certain utility, but no sense. The major part of my consistency proof, however, consists precisely in ascribing a finitist sense to actualist propositions "

Here the "finitist sense" of a proposition will be . an interactive program

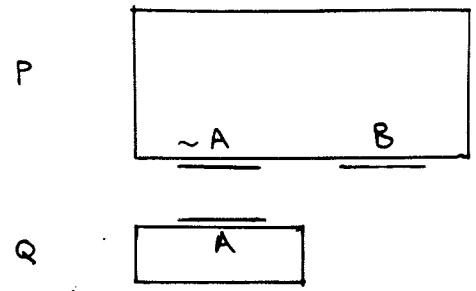
proposition , a strategy for a game associated to the



A : "interface" of P

(2)

Analysis of modus-ponens



modus-ponens = internal communication

parallel composition + hiding

cut-elimination = "internal chatters" end eventually

New (?) proof of cut-elimination

The finiteness of interaction is proved by a direct combinatorial reasoning about sequences of integers

(3)

- Intuitionistic meaning of quantifiers

- an intuitionistic proof of a formula

$\forall x \exists y \forall z \exists t \; R(x,y,z,t)$ (R: decidable relation)

can be seen as a winning strategy for ∃loïse in a game between two players

{ ∀belard

{ ∃loïse

∀ $x = a$

∃ $y = b$

∀ $z = c$

∃ $t = d$

does $R(a,b,c,d)$ hold ?

yes → ∃ wins

no → ∀ wins

$$\exists x \ \forall y \quad [\, D(x) \Rightarrow D(y) \,] \qquad (4)$$
$$\exists x \ \forall y \quad [\, f(x) \leq f(y) \,]$$

where $f : \mathbb{N} \to \mathbb{N}$ is an "oracle"

There is no <u>computable</u> winning strategy for ∃loïse

We allow ∃loïse to "change her mind"

∃ $x = 0$
∀ $y = b_1$
∃ $x = b_1$ if $f(b_1) < f(0)$
∀ $y = b_2$
∃ $x = b_2$ if $f(b_2) < f(b_1)$
⋮

∃loïse wins eventually because $\mathbb{N}$ is well-founded

$$f(0) > f(b_1) > f(b_2) > \dots$$

<u>Remarks</u> . ∃loïse "learns" from the environment

. the first move $x = 0$ is a "guess"

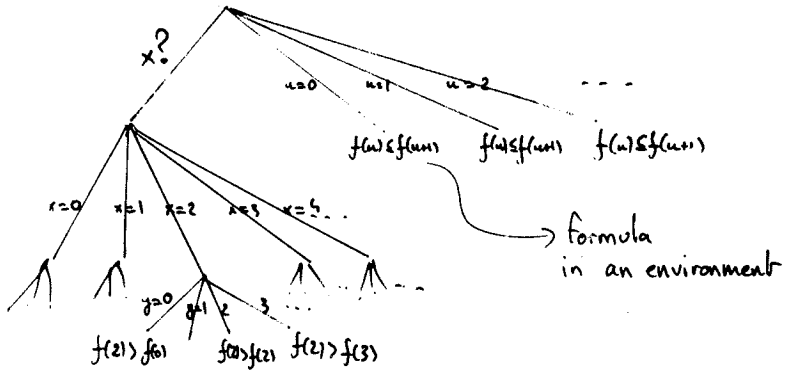
successive approximation towards a solution

In general $\vee$ ∃loïse $\wedge$ ∀belard (5)

and a Formula is a $\wedge$ $\vee$ tree, possibly infinitely branching where leaves are decidable closed formulae

$$[\, \exists x \, \forall y \, [f(x) \leq f(y)] \,] \implies \exists u \, [f(u) \leq f(u+1)]$$

$$\forall x \, \exists y \, [f(x) > f(y)] \quad \vee \quad \exists u \, [f(u) \leq f(u+1)]$$



Strategy for ∃loïse

∃ x?
∀ $x = a$
$f(a) \leq f(a+1)$ → ∃ $u = a$
$f(a) > f(a+1)$ → ∃ $y = a+1$

⑥

Problem for representing modus-ponens

"truth semantics" for classical logic ?

strategies described so far ⟷ cut-free proofs

"tell the proof how to behave in an environment that does not change its mind"

tree picture

"cooperation" between proofs via modus-ponens

P $\quad \exists x \, \forall y \quad [\, f(x) \leq f(y) \,]$

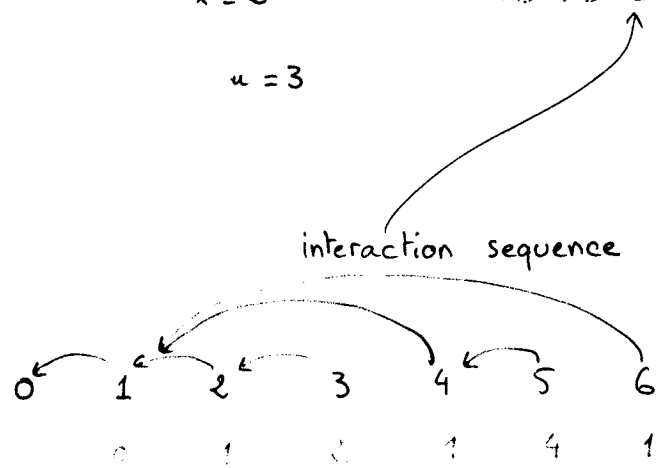
Q $\quad \forall x \, \exists y \quad [\, f(x) > f(y) \,] \quad \vee \quad \exists u \; f(u) \leq f(u+1)$

P and Q together

Q (P) $\qquad\qquad \exists u \; f(u) \leq f(u+1)$

$f(0) = 10 \qquad f(1) = 0 \qquad f(2) = 3 \qquad f(3) = 17 \ldots$ (+)

1	Q	x ?	asks 0
2	P	x = 0	asks 1
3	Q	y = 1	answers 2
4	P	x = 1	answers 1
5	Q	y = 2	answers 4
6	P	x = 2	answers 1
	Q	u = 3	



(8)

Interaction sequences first interaction of 2 cut-free proofs

$\varphi(1)$ $\varphi(2)$ $\varphi(3)$ ---

- $\varphi(1) = 0$
- $\varphi(n) < n$ φ changes parity

If one defines inductively

$$V(0) = \emptyset \qquad V(1) = \{0\}$$

$$V(n+1) = \{n\} \cup V(\varphi(n))$$

then $\varphi(n) \in V(n)$

Examples $\varphi(1) = 0$ $\varphi(2) = 1$ $\varphi(3) = 2$ or 0 ...

The case where the environment does not change its mind

$$\varphi(n) = n-1 \quad \text{if } n \text{ is even}$$

(9)

Behaviour of a proof against an environment that can "change its mind"

Notion of [illegible]

The formula seen as a tree is the "topic of the debate"

argument counter-argument counter-counter-argument ...

Two opponents

They both can change their mind

[illegible] at any point they can resume the debate at a point it was left before

(10)

Analysis of one interaction

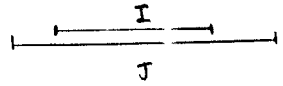
$\varphi(1)$ $\varphi(2)$ $\varphi(3)$ $\varphi(4)$ .... $\varphi(n)$ ....

Definitive interval $[\varphi(n), n]$ $n \notin \mathrm{Im}\varphi$
arguments that are not refuted

(1) The definitive intervals form a

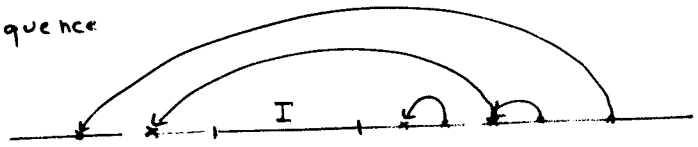
next structure

disjoint 



follows from

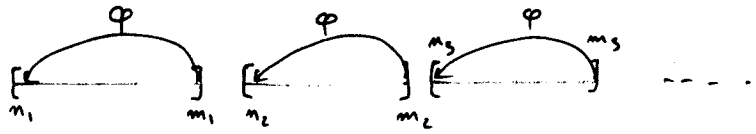
(2) If we take away a definitive interval, what is left is an interaction sequence



Main proposition

If $\varphi(1)$ $\varphi(2)$ $\varphi(3)$ ... infinite interaction sequence, then there exists

$n_1 = \varphi(m_1)$, $n_2 = m_1 + 1 = \varphi(m_2)$, ...



$$n_k = \varphi(n_{k+1} - 1)$$

this gives the finiteness of interaction of cut-free proofs.

(12)

A is a formula → may be partial

total element of [[A]] ?

strategy for the [illegible] of A

which is "winning" in the sense that if the debate game goes on forever

$\varphi(1)$ $\varphi(2)$ $\varphi(3)$ . . .

there is no infinite sequence (n_k)

$$\begin{cases} n_k = \varphi(n_{k+1} - 1) \\ n_k \quad \text{odd} \end{cases}$$

i.e. the relation $n = f(m-1)$ between odd integers is well-founded

$P \in [[A]]$
$Q \in [[\sim A]]$

} if the interaction sequence is infinite, then P or Q is not a total object

(13)

(1) If $P \in [[\sim A \vee B]]$

$Q \in [[A]]$

we can define $P(Q) \in [[B]]$ and it is total if P, Q are total

(this is reduced to a proposition about interaction sequences)

(2) If $P \in [[A]]$ is total, then P, "by restriction", gives a cut-free proof of A

simply apply P against an environment that does not change its mind.

⊢A (14

Classical logic ⊇ Intuitionistic logic
↑
higher-order computations ?

- connections between the idea of "resuming a point in a debate" and the use of dump in SECD evaluation ?
proof of normalisation?

- more examples ?

- variable for functions ?

- ∃f ---
↳ conjectures a law ?
gives a finite partial information about the function ?

- too sequential ?

Interaction Sequences

Thierry Coquand
Department of Computer Science, University of Göteborg/Chalmers
S-412 96 Göteborg, Sweden

March 20, 1992

Introduction

We present an abstract version of the notion of cuts between proofs. This leads to an argument of normalisation based on an analysis of what happens during the process of cut-elimination (and not on an induction on the complexity of the cut formula).

This paper is mathematically self-contained. A knowledge of infinitary propositional calculus, as presented in [5], may be useful for reading section 6.

1 Motivations

The idea of identifying a proof with a winning strategy for a game seems to come from Lorenzen [1, 2][1]. This identification is especially clear if we consider intuitionistic provability of arithmetical prenex formulae. For example, the game defined by a formula

$$\exists x.\forall y.\exists z..A(x,y,z),$$

where $A(x,y,z)$ is decidable, is that a player chooses a value for x, the opponent a value for y and then the player chooses a value for z. The player wins iff the formula $A(x,y,z)$ becomes true for this choice of values for x,y,z. In this case, it is clear that a winning strategy for this game corresponds exactly to an intuitionistic proof of the above formula.

[1] The author was lead to this identification by reading [1]

Looking at examples of prenex formulae that are classically valid, it seems natural to try to extend this analogy between proofs and winning strategy in the case of classical logic by allowing the proof, when it has to make a move, to answer to any previous move of its opponent, or to play a new initial move. One can then hope to identify classical proofs with winning strategy for such games. This was suggested by Lorenz [2].

Another idea, that comes from concurrency theory [3], is to interpret a strategy as an interactive programs and modus ponens as internal communication: given a winning strategy for $A \Rightarrow B$ and a winning strategy for A, one hopes to get a winning strategy for the game corresponding to B by letting the strategy for $A \Rightarrow B$ play against the strategy for A whenever its play concerns A. One expects then that the result of cut-elimination will be replaced by a proof showing that "internal chatters" end eventually.

When trying to put these ideas together, the difficulty is in the exact definition of what it means to "let two strategies play against each other". Trying to precise this leads to the notion of interaction sequence, which is a purely combinatorial notion.

One surprise is then that the main concepts about proofs, like the one of normal proofs, can be lifted at the level of interaction sequence. Basic facts about proofs, like cut-elimination, can also be expressed and proved at the level of interaction sequences.

We first present the notion of interaction sequence, and some of its basic properties. These are directly applied to a definition of classical provability for infinitary propositional formulae [5], for which modus ponens can be interpreted by internal communication.

2 Interaction Sequences

An **interaction sequence** is a pair (V, f) such that $V(0)$ is empty, $V(1) = \{0\}$, $f(1) = 0$, the function f is defined on an initial segment $[1, N]$ and for $n < N$

$$V(n+1) = \{n\} \cup V(f(n)), \quad f(n+1) \in V(n+1).$$

If (V, f) is defined for all positive integers, and for all N, (V, f) is an interaction sequence on $[1, N]$, we say that (V, f) is an **infinite interaction sequence**.

Notice that, if (V, f) is an interaction sequence, we always have $f(n) < n$ and $f(n), n$ are of distinct parity.

We let $y \prec x$ mean that $x \in f(V(y))$. By a direct induction on y, $y \prec x$ iff there exists a sequence $y_1, \ldots, y_n$ such that $y_1 = f(y-1)$, $y_{k+1} = f(y_k - 1)$ and $y_n = x$. Hence $\prec$ is transitive.

Lemma 1 *If $y \prec x$, then $V(x)$ is a strict initial segment of $V(y)$.*

Proof: By the alternative definition of $\prec$. □

We shall need a slight generalisation of the notion of interaction sequence. If $A = \{n_0, \ldots, n_k\}$, with $n_0 < \ldots < n_k$ and f is a function defined at least on $\{n_1, \ldots, n_k\}$, we say that f **defines an interaction** on A iff there exists an interaction sequence (V, g) defined on $[1, k]$ such that $f(n_p) = n_{g(p)}$ for $p = 1, \ldots, k$.

If $p = g(i)$, $q = g(j)$, we write $q \prec p$ (f, A) for the fact that $j \prec i$ relatively to the interaction sequence (V, g).

It can be seen directly that the following algorithm checks whether or not a function f defines an interaction on $\{n_0, \ldots, n_k\}$.

- If $k = 0$, then f does define an interaction on $\{n_0\}$.
- If $k > 0$, check recursively whether or not f defines an interaction on the set $\{n_0, \ldots, n_{k-1}\}$:
  - if not, then f does not define an interaction on $\{n_0, \ldots, n_k\}$,
  - if yes, we know that $f(n_{k-1})$ is of the form n_p, with $p < k-1$. If furthermore $f(n_k) = n_{p+1}$, then f defines an interaction on $\{n_0, \ldots, n_k\}$. Otherwise, f defines an interaction on $\{n_0, \ldots, n_k\}$ iff $f(n_k) \in \{n_0, \ldots, n_{p-1}\}$ and f defines an interaction sequence on the set $\{n_0, \ldots, n_{p-1}, n_k\}$.

Lemma 2 *If f defines an interaction on $\{n_0, \ldots, n_r\}$, $f(n_q) = n_p$ and n_q is not in the set $f(\{n_{q+1}, \ldots, n_r\})$, then f defines an interaction on the set $\{n_0, \ldots, n_{p-1}, n_{q+1}, \ldots, n_r\}$.*

Proof: By induction on $r - q$, using the previous algorithm. □

If A is an infinite subset $\{n_0, n_1, \ldots\}$, and f is a function defined at least on A, we say that f **defines an interaction** on A iff f defines an interaction on each $\{n_0, \ldots, n_k\}$.

Let us define $\mathsf{depth}(f,0) = 0$, $\mathsf{depth}(f,n) = \mathsf{depth}(f,f(n)) + 1$ for $n > 0$. The integer $\mathsf{depth}(f,n)$ is called the **depth** of n for f. We say that (V,f) is of **bounded depth** iff there exists N such that $\mathsf{depth}(f,n) < N$ for all n.

The following definitions will not be needed in the next two sections, but are needed for the definition of classical provability. We say that an interaction sequence f is **cut-free** iff $f(2p) = 2p-1$ whenever $2p$ is in the domain of f.

We define inductively $\mathsf{index}(f,n)$ for n in the domain of f by

- $\mathsf{index}(f,n) = n$ if $f(n) = 0$.
- otherwise, $\mathsf{index}(f,n) = \mathsf{index}(f,f(n))$.

3 Main Proposition

In this section, we suppose given an infinite interaction sequence (V,f).

Lemma 3 *if $f(x) > 0$, then $x \prec f(f(x))$.*

Proof: We have $f(x) \in V(x)$, hence $f(f(x)) \in f(V(x))$. □

If $A \subseteq \mathbf{N}$, $S_A(x)$ denotes $A \cap V(x)$.

An infinite subset $A = \{n_k\}$ is called **good** iff $f(A) \subseteq A$ and $S_A(n_{k+1}) = \{n_k\} \cup S_A(f(n_k))$.

Notice that $A = \mathbf{N}$ is good. Also, if A is good, then f defines an interaction on A.

Lemma 4 *If $A = \{n_k\}$ is good, either, for all q there exists $r > q$ such that $n_q = f(n_r)$, or there exists a good subset $\{m_l\}$ and p such that $n_i = m_i$ for $i < p$ and $m_p \prec n_p$.*

Proof: If A is good, n_q not in $f(A)$, and $n_p = f(n_q)$, let (m_l) be defined by $m_i = n_i$ for $i < p$, and $m_{p+l} = n_{q+1+l}$. It is clear that (m_l) is strictly increasing. Let $B = \{m_l\}$. Lemma 2 shows that $f(B) \subseteq B$. Furthermore

$$S_A(n_{q+1}) = \{n_q\} \cup S_A(n_p) = \{n_q, n_{p-1}\} \cup S_A(f(n_{p-1}))$$

and hence

$$S_B(m_p) = \{m_{p-1}\} \cup S_B(f(m_{p-1})).$$

It follows that B is good.

Notice also that $m_p \prec n_p$ because $n_q \in V(n_{q+1})$. □

Proposition 1 *Given an infinite interaction sequence (V,f), there exists an infinite sequence $u_1 < u_2 < u_3 \ldots$ such that $f(u_{p+1} - 1) = u_p$ for all p.*

Proof: This can be reformulated by saying that $\prec$ is not well-founded. Were $\prec$ well-founded, we could find a good subset $\{n_k\}$ such that n_{k+1} is $\prec$-minimal for good subsets starting with $n_0, \ldots, n_k$. By lemma 4, we have that for all p, there exists $q > p$ such that $n_p = f(n_q)$, and we get a contradiction by lemma 3. □

In the important special case of bounded depth sequences, we can build effectively a sequence (u_p) such that $u_{p+1} \prec u_p$. The algorithm is built by induction on a bound N of the depth. If $\mathsf{depth}(f,n)$ is always $< N$, we apply the induction hypothesis. Otherwise, lemma 2 shows that two segments of the form $[f(n), n]$ with $\mathsf{depth}(f,n) = N$ are such that they are disjoint or one is strictly included into another. We progressively remove all these segments that are maximal. In this way, either we are left with an infinite subset, which is a good subset $\{n_k\}$ where all $\mathsf{depth}(f,n_k)$ are $< N$, and we apply the induction hypothesis, or we are left with a finite subset, and the left extremity of the segments form a sequence (u_p) such that $u_{p+1} \prec u_p$ for all p.

4 Cut-elimination

An infinite interaction sequence (V,f) is said to be **winning** iff $\prec$ is well-founded over odd integers. If $A \subseteq \mathcal{N}$ is infinite, we define in a corresponding way when f defines a winning interaction on A.

Lemma 5 *If (V,f) is an interaction sequence on $[1,n_k]$ and $\{n_0,\ldots,n_k\}$ is a set X such that $f(n_j) \in X$ for $j = 1,\ldots,k$ and $f(n) \in \{n_1,\ldots,n_k\}$ implies $n \in X$, then f defines an interaction sequence on X.*

Proof: By induction on k.

If $k = 1$, then we have $f(n_1) = n_0$ and hence f defines an interaction on $\{n_0, n_1\}$.

If $1 < k$, and the lemma holds for all $p < k$, let (V,f) and X satisfying the hypothesis of the lemma. By induction hypothesis, f defines an interaction on $\{n_0, \ldots, n_{k-1}\}$.

If $f(n_k) = n_{k-1}$, then f defines an interaction on $\{n_0,\ldots,n_k\}$.

Otherwise, we have $f(i) \neq n_{k-1}$ for $i \in]n_{k-1}, n_k]$, and hence, by lemma 2, if we let n_p be $f(n_{k-1})$ we have $f(n_k) < n_p$. The hypothesis of lemma 5 apply then

to the set $\{n_0, \ldots, n_{p-1}, n_k\}$ and hence f defines an interaction on this set. This implies that f defines an interaction on $\{n_0, \ldots, n_k\}$. □

We suppose given an interaction sequence (V, f).

Let $I \subset N$ be the set of integers i such that $f(i) = 0$. If $i \in I$, let A_i be the set of integers n such that $\mathsf{index}(f, n) = i$. The set A_i satisfies the two conditions of lemma 5, and so f defines an interaction sequence on A_i.

Lemma 6 *If $i \in I$, and n is even, then $n \in A_i$ iff i is the least element of $V(n)$. If n is odd and $n \in A_i$, then $n + 1 \in A_i$.*

Proof: First, it is clear that i is odd, and that $i + 1 \in A_i$. Let $n > 0$ be even. The least integer k such that $f^k(n) = 0$ is even. Let $i = f^{k-1}(n) = \mathsf{index}(f, n)$. By lemma 1 and lemma 3, $V(f^{k-2}(n)) = \{i\}$ is an initial segment of $V(n)$, and hence i is the least element of $V(n)$. If $n > i$ is odd, and $n \in A_i$, then $f(n) \in A_i$ and $f(n)$ is even, $i \in V(f(n))$. Hence $i \in V(n+1)$ and $n + 1 \in A_i$. □

Corollary 1 *If $i \in I$, and n is even, $m \prec n$, and $n \in A_i$, then $m \in A_i$, and $m \prec n$ (f, A_i).*

If $J \subseteq I$ and X_J denotes the complement of the union of all sets A_i for $i \in J$, then X_J satisfies the two conditions of lemma 5, and so f defines an interaction sequence on X_J.

Proposition 2 (cut-elimination) *Let $J \subseteq I$ be such that f defines a winning interaction sequence on each infinite A_i for $i \in J$. If (V, f) is a winning interaction sequence, then f defines a winning interaction on X_J.*

Proof: Proposition 1 and the corollary of lemma 6 show that X_J is infinite, because otherwise, $\prec$ will be well-founded both on odd and even integers.

If f does not define a winning interaction on X_J, then there exists two infinite increasing sequences (x_k) and (y_k) in X_J such that $f(y_k) = x_k$, and x_{k+1} is the next element coming after y_k in X_J.

For each k, we show by induction on $l \leq k$ that f defines an interaction on $Y_l = [0, x_{k+1}[\setminus \cup_{i<l}[x_{k-i}, y_{k-i}]$. Indeed, we have $f(p) \neq y_{k-l}$ for $p \in Y_l$ and $y_{k-l} < p$. Hence, by lemma 2, if f defines an interaction on Y_l, for $l < k$, then it defines an interaction on Y_{l+1}.

It follows that f defines an interaction on $Y = [0, x_1[\cup \bigcup]y_k, x_{k+1}[$. Since $f(y_k) = x_k$ for all k, we have that $n \prec m$ (f, Y) implies $n \prec m$. It follows that $\prec$ (f, Y) is well-founded on odd integers. Since $X_J \cap Y$ is finite, the corollary of lemma 6 shows that $\prec$ (f, Y) is also well-founded on even integers. We get then a contradiction from proposition 1. □

5 Games

We use capital letters $A, B, S, \ldots$ for denoting finite sequences (or words). We denote by Sx the concatenation of S and x, and $<>$ denotes the empty sequence. If $S = x_1 \ldots x_n$, then n is the **length** of S. We say that a sequence T **extends** the sequence S iff T is of the form $Sx_1 \ldots x_p$.

All the objects we consider here, games and strategies, are considered given intuitionistically. In particular, they are computable objects.

5.1 Games and Strategies

A **game** G is a set of sequences which is such that $<> \in G$ and $S \in G$ whenever $Sx \in G$. The elements of G are called **game history**. If $S \in G$, the set $M_G(S) = \{x \mid Sx \in G\}$ is called the set of **possible moves from** S.

A **strategy** is a function σ defined on some elements of G of even length, and such that $\sigma(S) \in M_G(S)$ whenever $\sigma(S)$ is defined. The strategy is exactly defined on elements of G of even length that **follow the strategy** σ, where $s_1 \ldots s_n$ follows the strategy σ iff $\sigma(s_1 \ldots s_{2k})$ is defined and is s_{2k+1} for all k such that $2k < n$.

Given a strategy σ, we say that an infinite sequence $s_1 s_2 \ldots$ **follows the strategy** σ iff $s_1 \ldots s_n$ follow the strategy σ for all n.

5.2 Debate associated to a game

Let f be an interaction on $[1, n]$ and S a sequence $x_1 \ldots x_n$ of length n, we define for each $k \leq n$ a sequence $I(f, S, k)$ of length $\mathsf{depth}(f, k)$ by

- $I(f, S, 0) = <>$,
- $I(f, S, k)$ is the concatenation $I(f, S, f(k))x_k$ if $m > 0$.

Given a game G we let G^* be the set of sequences $(f(1), s_1) \ldots (f(n), s_n)$ such that

- f is an interaction on $[1, n]$ and
- for all $k \leq n$ we have $I(f, s_1 \ldots s_n, k) \in G$.

It is direct that this defines a game, called the **debate associated** to the game G.

We say that a strategy for G^* is **winning** iff for any infinite sequence $(f(1), s_1)(f(2), s_2)\ldots$, that follows this strategy, the infinite interaction sequence f is winning.

It may help the intuition of the reader to think about what happens during a real debate on a given topic between two persons. Both defend arguments, can change for a while their position, but also, at any point, can resume the debate at a point it was left before. This is what the game G^* represents, where G can be said to represent the "topic" of the debate.

5.3 Cut-Free Strategy

If G is a game, an element $(f(1), s_1)\ldots(f(n), s_n) \in G^*$ is **cut-free** iff f is cut-free.

A **cut-free strategy** for a game G^* is a function ϕ defined on some elements of G^* of even length that are cut-free. Such a strategy ϕ is defined exactly on sequences that **follow the strategy** ϕ and the sequence $(f(1), s_1)\ldots(f(n), s_n)$ follows the strategy ϕ iff f is cut-free and $(f(p+1), s_{p+1})$ is equal to $\phi((f(1), s_1)\ldots(f(p), s_p))$ for all even $p < n$.

It is clear that any strategy for G^* defines a cut-free strategy by restriction.

Intuitively, a cut-free strategy tells how to behave in a debate against an opponent that never changes in mind.

We recall that, if f is an interaction sequence on $[1, n]$, we have written $V(n+1)$ the set inductively defined $\{n\} \cup V(f(n))$. The following is the motivation behind the introduction of the set $V(n)$.

If $S = (f(1), s_1)\ldots(f(n), s_n) \in G^*$ is of even length, we define inductively a cut-free sequence $C(S) = (g(1), t_1)\ldots(g(l), t_l) \in G^*$ of even length and a strictly increasing function $F(S): [1, l] \to [1, n]$ such that $s_{F(S)(j)} = t_j$, $F(S)(l) = n$, $V(n+1)$ is exactly the image $F(S)(\{1, 3, \ldots, l-1\})$ and $f \circ F(S) = F(S) \circ g$:

- $C(<>) = <>$, and $F(<>)$ is the identity on the empty set,
- otherwise, we have $f(n) = p$ where $p < n$ is odd. We let T be $(f(1), s_1)\ldots(f(p-1), s_{p-1})$ and $C(T)$ be $(g(1), t_1)\ldots(g(l), t_l)$. We know by induction hypothesis that $f(p)$ is of the form $F(T)(q)$ for an odd $q \in [1, l]$. We define then $C(S)$ to be $C(T)(q, s_p)(l+1, s_n)$, and let $F(S)$ be the extension of $F(T)$ defined by $F(S)(l+1) = p$ and $F(S)(l+2) = n$.

Let ϕ be a cut-free strategy. We define a strategy $F(\phi)$ for G^* by computing $(q, s) = \phi(C(S))$ and letting $F(\phi)(S)$ be $(F(S)(q), s)$ for S of even length. The strategy $F(\phi)$ is called the **extension** of the cut-free strategy ϕ.

A cut-free strategy is said to be **winning** iff the relation of extension is well-founded on sequences that follow this strategy.

Lemma 7 *A winning strategy for G^* defines a winning cut-free strategy by restriction. Conversely, the extension of a winning cut-free strategy is a winning strategy.*

Proof: Direct from the definition. □

6 Classical provability

6.1 Classical Formulae

The formulae are defined inductively by the unique rule:

- if A_i, $i \in I$ is a family of formulae, then $A = |(A_i, i \in I)$ is a formula.

Intuitively, $|$ is a generalised Scheffer connective, and A says that the formulae A_i are incompatible, i.e. A holds iff at least one A_i does not hold.

In particular, the formula $0 = |(A_i, i \in \emptyset)$, is false under this interpretation. We write $|A$ for $|(A)$ where (A) is a family with one formula A. It represents the negation of A. Thus the formula $1 = |0$ is true under this interpretation.

If $A = |(A_i, i \in I)$ is a formula, and K is a subset of I, we let $A(K)$ be the formula $|(A_i, i \in K)$.

This language is directly seen to be equivalent to infinitary propositional calculus as described in [5]. As shown in Tait's paper [5], this calculus contains naturally Peano arithmetic.

6.2 Classical Games

Each formula can be seen as a tree. To each formula A, we associate the game G_A where, intuitively, each player chooses alternatively a subtree of the tree already chosen by the opposite player. Formally, if $A = |(A_i, i \in I)$, then G_A is the set with the empty sequence and the sequences of the form iS, with $i \in I$ and $S \in G_{A_i}$.

We define a **proof** of A to be a winning strategy for the game G^*_A. We say that A is **provable** iff it has a proof.

Notice that the formula 0 is not provable with this definition. There is only one strategy for G^*_A if $A = 1$, and it is a winning strategy, so that $1 = |0$ is provable.

A winning cut-free strategy of G^*_A can directly be seen as a normal proof of A in the sense of Tait in [5] where rules of or-introduction and rules of and-introduction are forced to alternate.

6.3 Principal Properties

Let $A = |(A_i, i \in I)$ and K be a subset of I. If $S \in G^*_A$ is the sequence $(f(1), s_1) \dots (f(n), s_n)$, we say that a move $(f(p), s_p)$ **plays in** $A(K)$ iff $\mathsf{index}(f, p) \in K$. Let $(f(p_1), s_{p_1}) \dots (f(p_l), s_{p_l})$ be the subsequence of S of elements $(f(p), s_p)$ that play in K. By lemma 5, there exists an interaction sequence g on $[1, l]$ such that $f(p_i) = p_{g(i)}$ for $i = 1, \dots, l$. We let $p_K(S) \in G^*_{A(K)}$ be the sequence $(g(1), s_{p_1}) \dots (g(l), s_{p_l})$.

If S is the sequence $(f(1), s_1) \dots (f(n), s_n)$, and $k \leq n$ is such that $f(k) = 0$, let $(0, s_k)(f(p_1), s_{p_1}) \dots (f(p_l), s_{p_l})$ be the subsequence of S of elements $(f(p), s_p)$ such that $\mathsf{index}(f, p) = k$. By lemma 5, there exists an interaction sequence g on $[1, l]$ such that $f(p_i) = p_{g(i)}$ for $i = 1, \dots, l$. We let $p_k(S) \in G^*_{A_{s_k}}$ be the sequence $(g(1), s_{p_1}) \dots (g(l), s_{p_l})$.

Proposition 3 (modus ponens) *If*

- $A = |(A_i, i \in I)$ *is provable.*
- $I = J \cup K$ *is a partition of* I.
- A_j *is provable for* $j \in J$.

then the formula $A(K) = |(A_i, i \in K)$ *is provable.*

Proof: Let σ be a winning strategy for A, and σ_j be a winning strategy for A_j, for $j \in J$. We say that a sequence $S \in G^*_A$ following σ is **correct** w.r.t. (σ_j) iff it is such that $p_k(S)$ follows σ_{s_k} whenever $f(k) = 0$ and $s_k \in J$.

Proposition 2 shows that the following extension $G(S)$ of S, for S sequence of even length $(f(1), s_1) \dots (f(n), s_n)$ following σ and correct w.r.t. (σ_j), is well defined:

- if $\phi(S) = (f(n+1), s_{n+1})$ and $\mathsf{index}(f, f(n+1)) = k$ is such that $s_k \in K$, then $G(S) = S(f(n+1), s_{n+1})$,
- otherwise, $\mathsf{index}(f, f(n+1)) = k$ is such that $s_\in J$. Let $(0, s_k)(f(p_1), s_{p_1}) \dots (f(p_l), s_{p_l})$ be the subsequence of $S(f(n+1), s_{n+1})$ of elements $(f(p), s_p)$ such that $\mathsf{index}(f, p) = k$, and g such that $f(p_i) = p_{g(i)}$ for $i = 1, \dots, l$. Since $p_k(S)$ follows σ_k, the element

$$\sigma_k(p_k(S(f(n+1), s_{n+1}))) = \sigma_k(\langle g(1), s_{p_1} \rangle \dots \langle g(l), s_{p_l} \rangle) = (m, s)$$

is well defined. We let $G(S)$ be $G(S(f(n+1), s_{n+1})(p_m, s))$.

Notice that $G(S)$ is of odd length, extends S and its last move plays in K.

We can now define simultaneously by induction a strategy ψ for $A(K)$, and for any sequence S following ψ, a sequence $F(S)$ such that $F(S)$ follows σ_i is correct w.r.t. (σ_j) and $p_K(F(S)) = S$. If S is of even length, let (p, s) be the last element of $G(F(S))$. There exists then a unique q such that $p_K(G(F(S)) = S(q, s)$ and we let $\psi(S)$ be (q, s) and $F(S\psi(S))$ be $G(F(S))$. If S is of odd length, and $S(p, s) \in G_{A(K)}$, we take $F(S(p, s))$ to be $F(S)(p, s)$. □

Proposition 4 *(consistency) For any formula* A, *at least one formula* A *or* $|A$ *is not provable.*

Proof: Because 0 is not provable. This follows also directly from proposition 1. □

It is clear that if $A = |(A_i, i \in I)$ and $K \subseteq I$ is such that $A(K)$ is provable, then A is provable, because a winning strategy for $A(K)$ is also a winning strategy for A.

Proposition 5 *If* $A = |(A_i, i \in I)$ *is provable,* $K \subseteq I$ *and there is an onto map* $\rho : I \to K$ *such that* $A_{\rho(i)} = A_i$ *for all* $i \in I$ *and* $\rho(i) = i$ *for* $i \in K$, *then* $A(K)$ *is provable.*

Proof: If $S \in G^*_A$ is the sequence $(f(1), s_1) \dots (f(n), s_n)$, let $G(S)$ be the sequence $(f(1), s'_1) \dots (f(n), s'_n)$, where $s'_i = \rho(s_i)$ if $f(i) = 0$, and $s'_i = s_i$ if $f(i) \neq 0$. It is clear that $G(S) \in G^*_{A(K)}$.

Let σ be a winning strategy of A. We define by simultaneous induction a strategy ψ for $A(K)$ and for any sequence S following ψ, a sequence $F(S)$ such that $F(S)$ follows σ and $G(F(S)) = S$.

If S is of even length, we compute $\sigma(F(S)) = (p, s)$. If $p = 0$, we let $\psi(S)$ be $(p, \rho(s))$ and $F(S(p, s))$ be $F(S)(p, s)$. If $p \neq 0$, we let $\psi(S)$ be (p, s) and $F(S(p, s))$ be $F(S)(p, s)$.

If S if of odd length, and $S(p, s) \in G_{A(K)}$, we let $F(S(p, s))$ be $F(S)(p, s)$. □

From proposition 3 and proposition 5 follows easily the equivalence of our notion of provable formulae with the usual definition of classical provability (as defined in [5]).

6.4 Example

A winning strategy can be seen as an interactive program, and proposition 3 interprets modus ponens as internal communication [3]. Here is an example of such a situation.

Given a function f on integer as a parameter, both formulae

$$A(f) = \forall x.\exists y \geq x.\forall z \geq x.\ [f(y) \leq f(z)]$$

and

$$B(f) = A(f) \Rightarrow \exists u_1, u_2, u_3.\ [u_1 < u_2 < u_3] \ \wedge\ [f(u_1) \leq f(u_2) \leq f(u_3)]$$

are provable. The second formula is even provable intuitionistically, but $A(f)$ holds only classically, if f is a parameter.

We will now define a winning cut-free strategy P for $A(f)$ and a winning cut-free strategy Q for $B(f)$. By lemma 7, this defines a winning strategy for $A(f)$ and $B(f)$ and proposition 3 leads then to a winning strategy for

$$\exists u_1, u_2, u_3.\ [u_1 < u_2 < u_3] \ \wedge\ [f(u_1) \leq f(u_2) \leq f(u_3)].$$

Such a winning strategy can be seen as a program computing u_1, u_2, u_3 such that $u_1 < u_2 < u_3$ and $f(u_1) \leq f(u_2) \leq f(u_3)$.

Rather than giving formally these winning cut-strategy, we will explain them heuristically.

The winning cut-free strategy P for $A(f)$ can be described as follows:

- the opponent gives a value for $x = a$.

- P answers $y = a$.
- the opponent gives a value for $z = a_1$. If $f(a) \leq f(a_1)$, P has won.
- If $f(a) > f(a_1)$, P changes its mind and plays $y = a_1$ instead.
- the opponent gives a value for $z = a_2$. If $f(a_1) \leq f(a_2)$, P has won.
- If $f(a_1) > f(a_2)$, P changes its mind and plays $y = u_2 \ldots$

Since $\mathbf{N}$ is well-founded, P is going to win eventually.

Here is a description of Q seen as a cut-free strategy for the formula

$$\exists x.\forall y \geq x.\exists z \geq x.[f(y) > f(z)] \vee (\exists u_1, u_2, u_3)[u_1 < u_2 < u_3 \wedge f(u_1) \leq f(u_2) \leq f(u_3)].$$

This is described informally:

- Q chooses $x = 0$.
- the opponent chooses a value $y = a_1$.
- Q changes its mind and plays $x = a_1 + 1$.
- the opponent chooses a value $y = a_2$, such that $a_2 \geq a_1 + 1$.
- if $f(a_1) > f(a_2)$, Q resumes the game with its initial value 0 for x, and wins by playing $z = a_2$. If $f(a_1) \leq f(a_2)$, Q changes its mind and plays $x = a_2 + 1$.
- the opponent chooses a value $y = a_3$, such that $a_3 \geq a_2 + 1$.
- if $f(a_3) > f(a_2)$, Q resumes the game with the value $a_1 + 1$ for x, and wins by playing $z = a_3$. Otherwise, $f(a_1) \leq f(a_2) \leq f(a_3)$, and Q wins by playing $u_1 = a_1, u_2 = a_2, u_3 = a_3$.

We are going now to show an example of an interaction between these two proofs (identified with cut-free strategies), in the case where the values of f are given by

$$f(0) = 10,\ f(1) = 5,\ f(2) = 3,\ f(3) = 7,\ f(4) = 4,\ f(5) = 11,\ f(6) = 29, \ldots$$

Here are the moves, as they are given by proposition 3:

1. Q plays $x = 0$.

2. P plays $y = 0$.

3. Q changes its mind, plays $x = 1$.

4. P plays $y = 1$.

5. $f(0) > f(1)$, hence Q plays $z = 1$.

6. P plays $y = 1$.

7. Q plays $x = 2$.

8. P plays $y = 2$.

9. $f(1) > f(2)$, hence Q plays $z = 2$.

10. P plays $y = 2$.

11. Q plays $x = 3$.

12. P plays $y = 3$.

13. $f(3) \geq f(2)$, hence Q plays $x = 4$.

14. P plays $y = 4$.

15. $f(4) < f(3)$, hence Q plays $z = 4$.

16. P plays $y = 4$.

17. $f(4) \geq f(2)$, hence Q plays $x = 5$.

18. P plays $y = 5$.

19. $f(5) \geq f(-)$, hence Q plays $u_1 = 2$, $u_2 = 4$, $u_3 = 5$.

The interaction sequence g associated to this interaction is given by:

$$g(1) = 0,\ g(2) = 1,\ g(3) = 0,\ g(4) = 3,\ g(5) = 2,\ g(6) = 1,$$

$$g(7) = 0,\ g(8) = 7,\ g(9) = 6,\ g(10) = 1,\ g(11) = 0,\ g(12) = 11,$$

$$g(13) = 0,\ g(14) = 13,\ g(15) = 12,\ g(16) = 11, g(17) = 0,\ g(18) = 17.$$

The computation of (u_1, u_2, u_3) consists in an exchange of values between P and Q, until a value $(u_1, u_2, u_3) = (2, 4, 5)$ is found by Q.

Conclusion

Our treatment seems to extend directly to the case of non necessarily well-founded formulae. We can even consider partial strategy, and prove for instance proposition 5 by a bissimulation argument.

The approach followed in this paper leads to a (may be new) proof of cut-elimination in a strictly deterministic framework. We think that it can be extended by allowing each player to play simultaneously a finite set of moves.

References

[1] Bernays P. (1968) "On the original Gentzen consistency proof for number theory." In Intuitionism and Proof Theory, Kino, Myhill, Vesley eds, pp. 409-417.

[2] Felsher, W. (1985) "Dialogues as a Foundation for Intuitionistic Logic" in D. Gabbay and F. Guenthner (eds.) Handbook of Philosophical Logic, Vol. III, pp. 341-372.

[3] Hoare, C.A.R. "Communicating Sequential Processes," Prentice-Hall, 1985.

[4] Lorenzen, K. (1958) "Ein dialogisches Konskruktivitatskriterium" in Infinistic Methods, Proceed. Symp. Found. of Math., PWN, Warszawa, pp. 193-200.

[5] Tait W.W. (1968) "Normal Derivability in Classical Logic," Springer Lecture Notes in Mathematics 72, J. Barwise ed, pp. 204-236.

Fixpoint Objects in a c.c.c.

ROY CROLE

A fixpoint object (ω, σ, Ω) in a c.c.c. with strong monad $\mathbb{T}$ is specified by

(i) An initial T-algebra

$$\begin{array}{ccc} T\Omega & \xrightarrow{\sigma} & \Omega \\ {\scriptstyle T\hat{f}}\downarrow & & \downarrow{\scriptstyle \exists!\hat{f}} \\ TA & \xrightarrow{f} & A \end{array}$$

(ii) A global element $\omega : 1 \to T\Omega$ for which there is an equaliser

$$1 \xrightarrow{\omega} T\Omega \underset{id}{\overset{\eta\sigma}{\rightrightarrows}} T\Omega$$

■

THEN if $f : TA \to TA$ in $\mathbb{C}$, we have $\widehat{f\mu}\sigma\omega : 1 \to TA$ and writing $Fix(f) := \widehat{f\mu}\sigma\omega$,

$$\begin{aligned} Fix(f) &= f\mu T(\widehat{f\mu})\omega \\ &= f\mu T(\widehat{f\mu})\eta\sigma\omega \\ &= f\mu\eta\,\widehat{f\mu}\sigma\omega \\ &= f \circ Fix(f). \end{aligned}$$

16/3/92 RECURSIVE TYPES VIA FIX POINT OBJECTS

Talk for the CLICS workshop, Aarhus Denmark, March 92

(i) Introduction

Recall the following:

(i) Computation Types, introduced by Moggi in LICS 1989. Formal equational system in which for any type α there is a type $T\alpha$ of "computations of type α." This equational system is an internal language for categories with strong monad.

(ii) Fixpoint type [see Crole/Pitts LICS 90]. Formal equational system involving a type fix in which given a judgement $\Gamma, x: T\alpha \vdash F(x): T\alpha$ there is a term-in-context $\Gamma \vdash Fix(F): T\alpha$ for which

$$\Gamma \vdash F(Fix(F)) = Fix(F): T\alpha.$$

(i) $T\Omega \xrightarrow{\sigma} \Omega$ initial

(ii) $1 \xrightarrow{u} T\Omega \xrightarrow[id]{\eta\sigma} T\Omega$

(iii) Universal type. There is a type dom and judgements of the form $\Gamma \vdash El(D)$ type where $\Gamma \vdash D: dom$. Think of elements of type dom as names of types, and $El(D)$ as the type named by D.

Basic idea is to combine (i) to (iii) in a single logical system which is consistent (!) and has a good categorical semantics. We do this in such a way that if $\Gamma, x: dom \vdash D(x): dom$ then $\Gamma \vdash Fix(D): dom$ and $\Gamma \vdash D(Fix(D)) = D: dom$. One can then think of the raw term $Fix(D)$ as a solution of a domain equatio

2 Description of Logic $FIX^{u}_{=}$

$FIX^{u}_{=}$ is a dependently typed equational logic with judgements of the form

- $\Gamma \equiv [x_1:\alpha_1, \ldots, x_n:\alpha_n]$ ctxt $FV(\alpha_i) \subseteq \{x_1, \ldots x_{i-1}\}$
- $\Gamma \vdash \alpha$ type
- $\Gamma \vdash M: \alpha$
- $\chi: \Gamma \to \Gamma'$

The system is presented using Martin Löf's theory of arities and expressions with the ground arities being TYPE and TERM. Raw types and terms α and M are closed meta-expressions in which we adopt the convention that object level variables are constants of the underlying meta-λ-calculus.

We shall aim to work with $FIX^{u}_{=}$ theories which are specified by a collection of axioms of the form

$$\Gamma \vdash M = M': \alpha$$
$$\Gamma \vdash \alpha = \alpha'$$

together with rules for deducing the theorems of the theory. Such a theory is specified by giving a $FIX^{u}_{=}$ signature. This is given by a collection of basic type/^{t} and term/^{f} valued function symbols, together with

(i) $El: 1 \to O$

(ii) $unit: O$, $null: O$, $nat: O$, $fix: O$, $dom: O$, $\times: O \to (O \to O)$, $+$, $\to$, T,

(iii) Symbols for the terms associated with $+, \times, T, \to$, etc etc and $\ulcorner unit \urcorner: 1, \ldots, \ulcorner \times \urcorner: 1 \to (1 \to 1), \ldots$ $I_p, J_p, \ldots$, Ret.

Examples of fixed point rules

$$\frac{\Gamma \vdash D : \mathrm{dom} \qquad \Gamma \vdash D' : \mathrm{dom}}{\Gamma \vdash D \times D' : \mathrm{dom}} \qquad \frac{\Gamma \vdash D : \mathrm{dom}}{\Gamma \vdash El(D) : \mathrm{typ}}$$

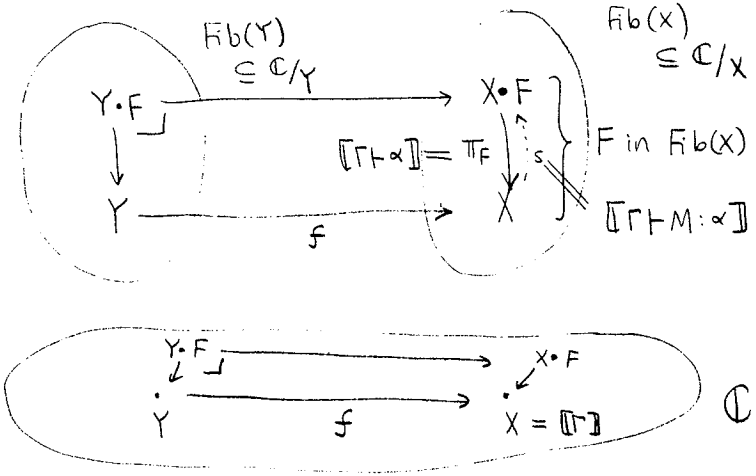
$$\frac{\Gamma, x : \alpha \vdash F(x) : T\beta \qquad \Gamma \vdash M : \alpha}{\Gamma \vdash \lambda\alpha(F)\, M = F(M) : T\beta}$$

$$\frac{\Gamma \vdash M : El(D \times D')}{\Gamma \vdash \mathrm{Ip}(M) : El(D) \times El(D')} \qquad \frac{\Gamma \vdash E : T\mathrm{dom}}{\Gamma \vdash \mathrm{Ret}(E) : \mathrm{dom}}$$

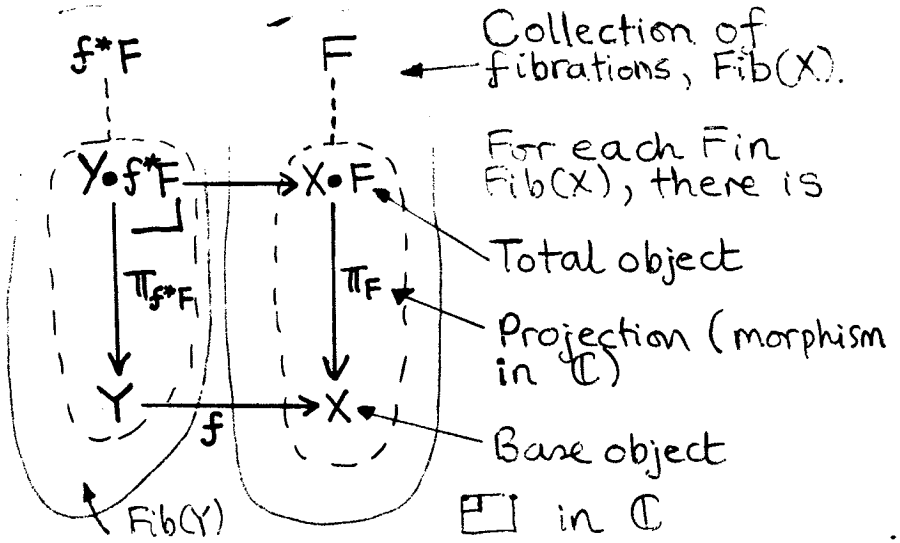
Theorem Required fixpt. property for dom does hold $\Rightarrow$ $\mathrm{Fix}(D) \equiv \mathrm{It}(g.D(\mathrm{Ret}(g)), \sigma(\omega))$

3. Categorical Semantics

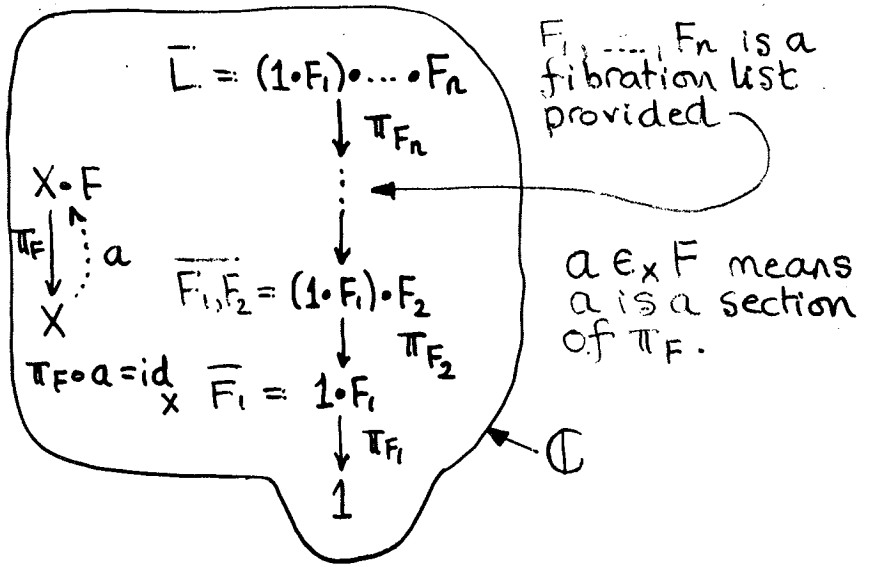
We shall base the semantics of FIX$^{\leq}$ on the notion of a category-with-attributes.



Categories with Attributes



May regard $\mathrm{Fib}(X) \subseteq \mathbb{C}/X$ via $\mathrm{Fib}(X)(F, F')$ iff $\mathbb{C}/X(\pi_F, \pi_{F'})$.



Defn $\mathbb{C}$ a let-category / strong monad. If for all objects C, C', $\mathbb{C}(-\times C, TC')$ represented by $\varepsilon : C \to C'$, say $\mathbb{C}$ has T-exponentials.

Defn FIX-category-with-attributes

(i) Each $\mathrm{Fib}(X)$ is let-category with ~~stable~~ distributive finite coproducts and T-exp.

(ii) The pullback fns. f^* preserve structure of the $\mathrm{Fib}(X)$, and the monad structure.

(iii) There are $\hat{1}, \hat{0}, \hat{N}, \hat{\Omega}$ in $\mathrm{Fib}(1)$ such that (for example) $!^*(\hat{\Omega})$ is the FPO of $\mathrm{Fib}(X)$ where $! : X \to 1$ in $\mathbb{C}$

(iv) There's some $\hat{U} \in \mathrm{Fib}(1)$ for which

$$\hat{U} \underset{\sim}{\overset{i}{\rightleftarrows}} T\hat{U} \text{ in } \mathrm{Fib}(1)$$

and there's $\tau \in \mathrm{Fib}(1 \cdot \hat{U})$ along with (for example) $U^2 \to U$ with $1 \cdot \hat{U}$

$$\ulcorner X \urcorner : 1 \cdot \hat{U} \times 1 \cdot \hat{U} \longrightarrow 1 \cdot \hat{U}$$

with $\ulcorner X \urcorner^*(\tau) \cong \pi_1^*(\tau) \times \pi_2^*(\tau)$ in $\mathrm{Fib}(1 \cdot \hat{U} \times 1 \cdot \hat{U})$

where $\pi_1 : 1 \cdot \hat{U}^2 \to 1 \cdot \hat{U}^2$.

Given a FIX$^{\Omega}_{=}$ signature Σ, a structure $\mathcal{M}$ in a FIXCwA is specified by

(i) For each basic Typ. V. fn symbol t a fibration list L_t and fib^n $F_t \in \mathrm{Fib}(\overline{L}_t)$

(ii) For each basic term valued fn. sym. f, L_f, $F_f \in \mathrm{Fib}(\overline{L}_f)$, $a_f \in_{\overline{L}_f} F_f$.

Define a semantics by giving relations

$$[\![J]\!] \sim ?$$

(i) $[\![\Gamma]\!] \sim L$ a fib. list

(ii) $[\![\Gamma \vdash \alpha]\!] \sim L \vdash F \quad (F \in \mathrm{Fib}(\overline{L}))$

(iii) $[\![\Gamma \vdash M : \alpha]\!] \sim L \vdash a : F \quad (a \in_{\overline{L}} F)$

Have a FPO $!^*\hat{\Omega}$ in $\mathrm{Fib}(\overline{L})$

$$\begin{array}{ccc} \overline{L} \cdot T!^*\hat{\Omega} & \longrightarrow & \overline{L} \cdot !^*\hat{\Omega} \\ \Big\downarrow T(\eta_F \circ g) & & \exists! \Big\downarrow g \\ \overline{L} \cdot TF & \longrightarrow & \overline{L} \cdot F \end{array}$$

The assignments $\vdash \longmapsto \llbracket - \rrbracket$ give rise to partial functions; we say a [illegible] is satisfied if $\llbracket \vdash \rrbracket$ is defined. $\mathcal{U}$ is a model of Th if it satisfies the Th axioms.

Thm $\mathrm{FIX}^{\mathcal{U}}_{=}$ soundness. Given Th, $\mathcal{U}$ satisfies the theorems of Th. ▨

(4) Models of $\mathrm{FIX}^{\mathcal{U}}_{=}$

A Scott predomain is a bounded cocomplete alg. dcpo; these form a category Spd, with full-subcat. Spd_ω of countably based predomains.

The category pInS has objects $(A, \downarrow, \vdash)$ as for information systems, with the following changes to the axioms for inf sys.

(i) $\phi \downarrow_A$ (ii) $X \subseteq^f Y \downarrow_A$ and $X \neq \phi$ then $X \downarrow_A$

The morphisms are (essentially) approximable maps with the additional axiom $\forall a \in A.\ \exists Y \downarrow_B .\ \{a\} \Gamma Y$ where $\Gamma : (A, \downarrow, \vdash) \longrightarrow (B, \downarrow, \vdash)$. We have

Theorem There are functors $|-| : \mathrm{pInS}_\omega \longrightarrow \mathrm{Spd}_\omega$ and $\mathcal{I} : \mathrm{Spd}_\omega \longrightarrow \mathrm{pInS}_\omega$ giving rise to a strong equivalence. ▨

Lemma The set $\mathrm{PS}_{\mathbb{N}}$ of all preinformation systems with token set $A \subseteq \mathbb{N}$ forms an ωcpo wrt. the usual restriction ordering.

Lemma There are morphisms in ωCpo (eg. $\ulcorner - \urcorner : \mathrm{PS}_{\mathbb{N}} \times \mathrm{PS}_{\mathbb{N}} \longrightarrow \mathrm{PS}_{\mathbb{N}}$) which "give rise to" a choice of (eg. binary product) in pInS.

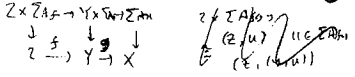
Pf // Note that if A, B in pInS, then $A \ulcorner \times \urcorner B$ and also $A \ulcorner \to \urcorner B$ have slightly different constructions than for ordinary inf. sys. ▨

Theorem ωCpo is a $\mathrm{FIX}_{c\omega a}$.

Pf (Sketch)

The collection of fibrations over ωcpo X is given by the continuous functors $A : X \longrightarrow \omega\mathrm{Cpo}^{ep}$, where $(\,)^{ep}$ means embedding/partial projection pairs. Performing the (covariant) Grothendieck construction yields a morphism $\pi_A : \mathbb{G}(A) \equiv \sum_{x \in X} A_x \longrightarrow X$.

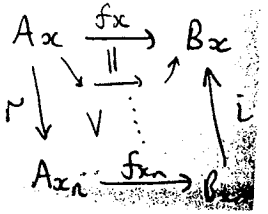
If $f : Y \to X$, define $f^* A \stackrel{def}{=} Af$. Simple to check that $\sum_{y \in Y} A_{fy} \cong Y \times_X \mathbb{G}(A)$ – so the functoriality conditions for a cwa. hold.



Lemma $f \in \omega\mathrm{Cpo}/X\,(\pi_A, \pi_B)$ corresponds to a family of cts. fns. $(f_x : A_x \to B_x \mid x \in X)$ for which

(i) $A_x \longrightarrow B_x$, $A_{x'} \longrightarrow B_{x'}$ $(x \sqsubseteq x')$

(ii) $A_x \xrightarrow{f_x} B_x$, $A_{x_n} \xrightarrow{f_{x_n}} B_{x_n}$ $x = \bigvee x_n$



$Fib(X)$ is a let-category with $(X), +, \to, N, \Omega, \perp$

Define structure pointwise - use lemma to check details.

N.B. Care as dealing with partial projections.

So, eg. $\eta_A : A \to \perp A$ given by $(\eta_A)_x : Ax \longrightarrow (Ax)_\perp$

Example of fibration $\hat{\Omega}$; of course $\hat{\Omega} : 1 \to \omega\underline{CPO}^{ep}$ and $\hat{\Omega}(*) = \Omega \stackrel{def}{=} \{0 < 1 < 2, \ldots, < T\}$.

Need to check $!^*(\hat{\Omega}) \stackrel{def}{=} \hat{\Omega}!$ is a F.P.O. in $Fib(X)$.

- unravel definitions, use lemma.

Define $\hat{U} : 1 \to \omega\underline{CPO}^{ep}$ by $\hat{U}(*) = PS_N$. Retract condition is immediate. Define τ by

$$\tau : PS_N \longrightarrow \underline{Spd}_\omega^{ep} \subseteq \omega\underline{CPO}^{ep}$$

$$A \longmapsto |A| \qquad |-| : p\underline{InS}_\omega \longrightarrow \underline{Spd}_\omega$$

$$A \leq A' \longmapsto |A| \underset{r}{\overset{i}{\rightleftarrows}} |B| \quad \text{similar to inf. sys}$$

Finally need to verify that (eg)

$$\ulcorner \urcorner^*(\tau) \cong \pi_1^*(\tau) \to (\pi_2^*(\tau))_\perp$$

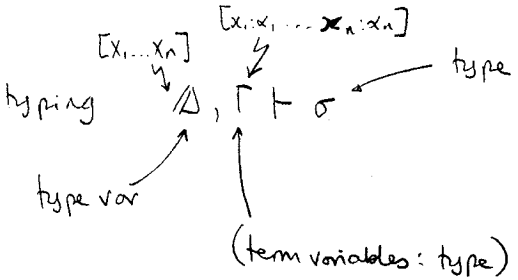
ie $$\sum_{(A,B) \in PS^2} |A \ulcorner\urcorner B| \cong \sum_{A \in PS} |A| \to |B|_\perp \quad \text{in } \omega CPO / PS_N^2$$

Use the preinf. sys. isomorphisms on summands + the lemma. ▨

Semantics For Programming Languages

Language with

(i) Static typing



(ii) Dynamic evaluation

$$\Delta, \Gamma \vdash M \to C : \sigma$$

term — canonical form

Give semantics by using a $FIX^{\mathcal{M}}_{=}$ - theory. Translate (i) to

$$[d_1 : dom, \ldots, d_n : dom], [x_1 : El([\![\sigma_1]\!]), \ldots] \vdash [\![M]\!] : T[\![\sigma]\!]$$

for a call-by-value language.

Categorical Semantic for the Fixpoint Term. in FIX $\hat{=}$.

$$\frac{[\![\Gamma]\!] \blacktriangleright \mathcal{L}}{[\![\Gamma \vdash \omega : T_{fix}]\!] \blacktriangleright \mathcal{L} \vdash !^* \omega : !^* T\hat{\Omega}}$$

$$\frac{[\![\Gamma \vdash E : T_{fix}]\!] \blacktriangleright \mathcal{L} \vdash \langle id, e \rangle : T!^*\hat{\Omega}}{[\![\Gamma \vdash \sigma(E) : fix]\!] \blacktriangleright \mathcal{L} \vdash \langle id, \sigma e \rangle : !^*\hat{\Omega}}$$

$$\frac{[\![\Gamma, x : T\alpha \vdash F(x) : \alpha]\!] \blacktriangleright \mathcal{L}, TF \vdash a : \pi^*_{TF} F}{[\![\Gamma \vdash It_x(F, N) : x]\!] \blacktriangleright \mathcal{L} \vdash g\langle id, n \rangle : F}$$

$$[\![\Gamma \vdash N : fix]\!] \blacktriangleright \mathcal{L} \vdash \langle id, n \rangle : !^*\hat{\Omega}$$

E. Moggi
moggi@igecuniv.bitnet
DISI[1], Univ. di Genova
v. Benedetto XV 3
16132 Genova, Italy

SUMMARY

- **motivations**: framework for program logics
- Evaluation Logic: language and informal semantics
- papers overview:
  [Mog89,Mog91a,CP90,Pit90,Mog91b]
- categorical semantics and axioms:
  hyperdoctrines versus toposes (see [Pit81])
- expressiveness of Evaluation Logic and limitations
- issues and further research

[1] Dipartimento di Informatica e Scienze dell' Informazione.

- METALANGUAGE $ML_T(\Sigma)$: $\tau \in Types$, $e \in Terms$

  T $\dfrac{\Gamma \vdash \tau \text{ type}}{\Gamma \vdash T\tau \text{ type}}$

  the type[3] of computations (= denotations of programs) which return values of type τ

  lift $\dfrac{\Gamma \vdash e : \tau}{\Gamma \vdash [e]_T : T\tau}$

  the computation which simply evaluates to e

  let $\dfrac{\Gamma \vdash e_1 : T\tau_1 \qquad \Gamma, x:\tau_1 \vdash e_2 : T\tau_2}{\Gamma \vdash (\text{let}_T\ x \Leftarrow e_1 \text{ in } e_2) : T\tau_2}$

  the computation which evaluates e_1, then binds the result to x and finally evaluates e_2

- EVALUATION LOGIC $EL_T(\Sigma)$: $\phi \in Formulas$

  box $\dfrac{\Gamma \vdash e : T\tau \qquad \Gamma, x:\tau \vdash \phi \text{ prop}}{\Gamma \vdash [x \Leftarrow e]\phi \text{ prop}}$

  every result produced by e satisfies ϕ

  diamond $\dfrac{\Gamma \vdash e : T\tau \qquad \Gamma, x:\tau \vdash \phi \text{ prop}}{\Gamma \vdash \langle x \Leftarrow e \rangle \phi \text{ prop}}$

  some result produced by e satisfies ϕ

  eval $\dfrac{\Gamma \vdash e : T\tau \qquad \Gamma \vdash v : \tau}{\Gamma \vdash e \Downarrow v \text{ prop}}$

  v is a result produced by e

[3] The interaction of computational types with polymorphic and dependent types is problematic.

- **Frameworks for Program Logics**:

  uniform presentation of/modular approach to progr. lang. and progr. logics

- ***LCF* approach to program correctness**:

  programs as terms and assertions as formulas, reasoning about programs=reasoning about math. structures for progr. lang. semantics

- **Semantic approach** and mathematical theories[2] (under development):

  Synthetic Domain Theory

  Axiomatic Domain Theory and Inductive Principles

  Monadic Approach and computational aspects:

  * Modular approach to Denotational Semantics
  * Interaction with recursive definitions
  * Interaction with (Higher Order) Logic
    Evaluation Logic

Expressiveness and Viability of frameworks: test cases, applications.

[2] New ways of looking at/structuring Denotational Semantics.

OVERVIEW [Mog89]

- monads and λ_c-calculus.
- "The λ_c-calculus open the possibility to develop a new LCF, based on an abstract semantics of computations rather than Domain Theory, for studying axiomatically different notions of computation"

NB: non-standard interpretation of λ_c-terms (biased towards CBV) $\beta\eta$-equivalence unsound.

OVERVIEW [Mog91a]

- metalanguages ML_T extending typed $\lambda\beta\eta$-calculus;
- translation of programming languages into ML_T:

  $\vdash_{pl} e:\tau$ get translated into $\vdash_{ml} e^*:T(\tau^*)$;
- modalities $\gamma: T\Omega \to \Omega$ in a topos.

  "HML_T, which combines computational types and HOL, seems to be the ideal framework for program logics .... In ML_T one can describe a programming language by introducing additional constants and axioms. In HML_T one can describe also a program logic, by adding constants $p: TA \to \Omega$ corresponding to properties of computations."

  Examples of **modal operators** $\Box: T\Omega \to \Omega$ to express:
  1) *may* and *must* for non-deterministic computations,
  2) Hoare's triples for computations with side-effects.

NB: HML_T does not provide any general purpose predicate constructors (such as the modalities of EL_T).

ML_T AXIOMS[4] for LET and LIFT

- $T\beta$ $\dfrac{\Gamma \vdash e:TB}{\Gamma, x:A \vdash (\mathrm{let}_T\, x \Leftarrow [v]_T \,\mathrm{in}\, e) =_{TB} [v/x]e}$
- $T\eta$ $c:TA \vdash (\mathrm{let}_T\, x \Leftarrow c \,\mathrm{in}\, [x]_T) =_{TA} c$
- associativity of LET

$$\frac{\Gamma, x:A \vdash e_1:TB \qquad \Gamma, y:B \vdash e_2:TC}{\begin{array}{l}\Gamma, c:TA \vdash (\mathrm{let}_T\, y \Leftarrow (\mathrm{let}_T\, x \Leftarrow c \,\mathrm{in}\, e_1)\,\mathrm{in}\, e_2) =_{TC} \\ \qquad (\mathrm{let}_T\, x \Leftarrow c \,\mathrm{in}\, (\mathrm{let}_T\, y \Leftarrow e_1 \,\mathrm{in}\, e_2))\end{array}}$$

Improved CONGRUENCE RULE[5]

- congruence for LET

$$\frac{\Gamma, x:A \vdash e_1:TB \qquad \Gamma, x:A \vdash e_2:TB}{\begin{array}{l}\Gamma, c:TA \vdash [x \Leftarrow c](e_1 =_{TB} e_2) \Longrightarrow \\ \qquad (\mathrm{let}_T\, x \Leftarrow c \,\mathrm{in}\, e_1) =_{TB} (\mathrm{let}_T\, x \Leftarrow c \,\mathrm{in}\, e_2)\end{array}}$$

[4] Valid in a category $\mathcal{C}$ with products and a strong monad T.

[5] Valid in a category $\mathcal{C}$ with products, a dominion $\mathcal{M}$ s.t. $\Delta_c \in \mathcal{M}[c \times c]$ for every $c \in \mathcal{C}$ ($\mathcal{M}$ supports equality) and a strong monad T preserving pullbacks of monos in $\mathcal{M}$.

- FIX-type (extension to ML_T for recursive definitions);
- FIX-logic (a fragment of FOL) with inductive principles

  "The induction rule for *fix* is consistent, but only because the proposition of FIX-logic have limited forms."
- The evaluation predicate $c{\Downarrow}v$ is treat in a very extensional way, i.e. $c = [v]_T$ (this is fine for partiality $TX = X_\perp$ and exceptions $TX = X + E$).

$$\frac{\Gamma, x{:}A \vdash \phi \text{ prop}}{\begin{array}{l}\Gamma, c{:}TA \vdash (\langle x{\Leftarrow}c\rangle\phi) \stackrel{\Delta}{\Longleftrightarrow} (\exists x{:}A.c =_{TA} [x]_T \wedge \phi)\\ \Gamma, c{:}TA \vdash ([x{\Leftarrow}c]\phi) \stackrel{\Delta}{\Longleftrightarrow} (\forall x{:}A.c =_{TA} [x]_T \supset \phi)\end{array}}$$

  "It is possible to envisage a weaker logic in which $c{\Downarrow}v$ is an atomic predicate"

  "FIX is not an *integrated* logic. Something to aim for is a system combining features of FIX with those of CC"

  "There are strong connections between FIX and LCF. However, the precise relationship between FIX and **Axiomatic Domain Theory** has yet to be clarified."

NB: the categorical semantics of FIX-logic (Def 4.1) is quite involved (5 pages long); its concern is more on inductive principles than the integration on computational types in HOL.

- Evaluation Logic (EL_T), general properties of necessity and possibility

  "One might consider axiomatising computation-related properties of a monad within the framework of a suitable logic. It is the question of what is a suitable logic which is addresses in this paper. We will extend ML_T to a constructive predicate logic which permits the formulation of statements about **evaluation of computations to values**[6]"
- connections with operational semantics.

  "...the motivation for formulating EL_T came more from the Natural Semantics style of operational semantics (than Dynamic Logic)"
- modalities $\Box, \Diamond{:}\mathcal{P}(X \times Y) \to \mathcal{P}(X \times TY)$, axioms and hyperdoctrine semantics;

  Models of EL_T for partiality and side-effects: "$\mathcal{P}(X)$ is the collection of inclusive subsets of $X \times S$";
- translation of operational judgments in EL_T and adequacy

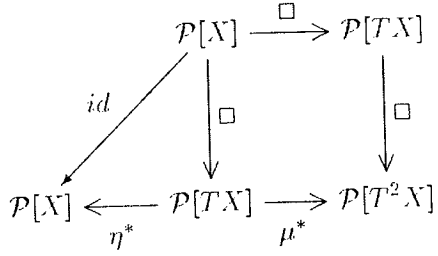
$$c{:}TA, v{:}A \vdash (c{\Downarrow}v) \stackrel{\Delta}{\Longleftrightarrow} (\langle x{\Leftarrow}c\rangle v =_A x)$$

  "The full modal logic should come into its own when devising computationally adequate theories for language with non-determinism. Even for deterministic languages, evaluation modalities appear useful when we address the question of logical principles for reasoning about the behavior of programs"

NB: The semantics of necessity and possibility is specified by additional structure, and allows **non-standard** models.

[6] If c is a non-deterministic computation, there is no way to say "c may diverge", nor can we express the modalities of Hennessy-Milner Logic.

- $\mathcal{B}$ category with products, $\mathcal{P}$ $\mathcal{B}$-indexed meet-semilattice
- T strong monad over $\mathcal{B}$, $\Box_X, \Diamond_X : \mathcal{P}[X] \to \mathcal{P}[TX]$ s.t.



and few additional properties hold (see axioms for EL_T).
$\Diamond_X$ must satisfy similar properties.

$\Gamma, x{:}A \vdash \phi$ prop	$=$	$p \in \mathcal{P}[\Gamma \times A]$
$\Gamma, c{:}TA \vdash [x{\Leftarrow}c]\phi$ prop	$=$	$\mathrm{t}^*_{\Gamma,A}(\Box p) \in \mathcal{P}[\Gamma \times TA]$

- Models with $\mathcal{B}$=category of sets:

  non-determinism: TX=set of subsets of X,
  $\mathcal{P}[X]$=poset of subsets of X,
  $(\Box p)(c)$ iff $c \subseteq p$,
  $(\Diamond p)(c)$ iff $c \cap p$ inhabited;

  side-effects (S non-empty set): $TX = (X \times S)^S$,
  $\mathcal{P}[X]$=poset of subsets of $X \times S$,
  $(\Box p)(c, s)$ iff $(\Diamond p)(c, s)$ iff $p(cs)$;

  in the latter case we don't get a model by taking

  $\mathcal{P}[X]$=poset of subsets of X,
  $(\Box p)(c)$ iff $\forall s{:}S.p(\pi_1(cs))$,
  $(\Diamond p)(c)$ iff $\exists s{:}S.p(\pi_1(cs))$

- preservation of entailment

$$\frac{\Gamma, x{:}A \vdash \Phi, \phi \Longrightarrow \psi}{\begin{array}{l}\Gamma, c{:}TA \vdash \Phi, [x{\Leftarrow}c]\phi \Longrightarrow [x{\Leftarrow}c]\psi \\ \Gamma, c{:}TA \vdash \Phi, \langle x{\Leftarrow}c\rangle\phi \Longrightarrow \langle x{\Leftarrow}c\rangle\psi\end{array}} \quad x \notin \mathrm{FV}(\Phi)$$

- values (LIFT)

$$\frac{\Gamma, x{:}A \vdash \phi \text{ prop}}{\begin{array}{l}\Gamma, v{:}A \vdash [v/x]\phi \Longleftrightarrow [x{\Leftarrow}[v]]\phi \\ \Gamma, v{:}A \vdash [v/x]\phi \Longleftrightarrow \langle x{\Leftarrow}[v]\rangle\phi\end{array}}$$

- sequential composition (LET)

$$\frac{\Gamma, x{:}A \vdash e{:}TB \qquad \Gamma, y{:}B \vdash \phi \text{ prop}}{\begin{array}{l}\Gamma, c{:}TA \vdash [x{\Leftarrow}c]([y{\Leftarrow}e]\phi) \Longleftrightarrow [y{\Leftarrow}(\mathrm{let}_T\, x{\Leftarrow}c\,\mathrm{in}\,e)]\phi \\ \Gamma, c{:}TA \vdash \langle x{\Leftarrow}c\rangle(\langle y{\Leftarrow}e\rangle\phi) \Longleftrightarrow \langle y{\Leftarrow}(\mathrm{let}_T\, x{\Leftarrow}c\,\mathrm{in}\,e)\rangle\phi\end{array}}$$

- commutativity with el and $\vee$

$$\frac{\Gamma, x{:}A \vdash \phi_i \text{ prop}}{\begin{array}{l}\Gamma, c{:}TA \vdash \wedge_i([x{\Leftarrow}c]\phi_i) \Longleftrightarrow [x{\Leftarrow}c](\wedge_i\phi_i) \\ \Gamma, c{:}TA \vdash \vee_i(\langle x{\Leftarrow}c\rangle\phi_i) \Longleftrightarrow \langle x{\Leftarrow}c\rangle(\vee_i\phi_i)\end{array}}$$

- possibility and necessity

$$\frac{\Gamma, x{:}A \vdash \phi \text{ prop} \qquad \Gamma, x{:}A \vdash \psi \text{ prop}}{\Gamma, c{:}TA \vdash ([x{\Leftarrow}c]\phi), (\langle x{\Leftarrow}c\rangle\psi) \Longrightarrow (\langle x{\Leftarrow}c\rangle\phi \wedge \psi)}$$

- possibility and $\wedge$

$$\frac{\Gamma \vdash \phi^* \text{ prop} \qquad \Gamma, x{:}A \vdash \psi \text{ prop}}{\Gamma, c{:}TA \vdash \phi^*, (\langle x{\Leftarrow}c\rangle\psi) \Longleftrightarrow (\langle x{\Leftarrow}c\rangle\phi^* \wedge \psi)}$$

- truth preservation (not in [Pit90])

$$\frac{\Gamma \vdash \phi^* \text{ prop}}{\Gamma, c{:}TA \vdash \phi^* \Longrightarrow [x{\Leftarrow}c]\phi^*} \quad x \notin \mathrm{FV}(\phi)$$

NB: ϕ^* must be **state-independent** (e.g. an equality).

- Topos-semantics of EL_T uniquely determined by the strong monad T. Do we always get the expected interpretation of modalities?

  NB: dynamic allocation is problematic.

- Approaches to semantics of EL_T:

  - hyperdoctrine-like: T on the base $\mathcal{B}+$ **additional structures** on the fibers
  - topos-like: T on $\mathcal{C}+$ **additional properties** of T and $\mathcal{C}$

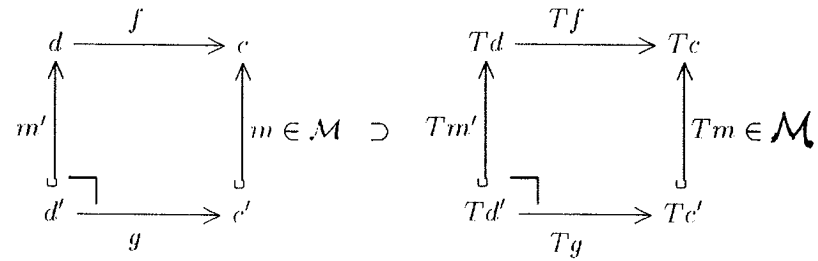
- Additional properties of T and $\mathcal{C}$

  - for necessity: T preserves pullbacks of monos (not satisfied by continuations $TX = R^{R^X}$);
  - for possibility: $\ldots+\mathcal{C}$ is a topos+ necessity commutes with $\forall$ and $\supset$

    NB: no simple properties for possibility.

  NB: relation between the two approaches to be done (should be based on [Pit81]).

- $\mathcal{C}$ category with products, $\mathcal{M}$ dominion[7] over $\mathcal{C}$
- T strong monad over $\mathcal{C}$ s.t.



one may consider additional properties of $\mathcal{M}$ and T

$\Gamma, x{:}A \vdash \phi$ prop	=	$m \in \mathcal{M}[\Gamma \times A]$
$\Gamma, c{:}TA \vdash [x{\Leftarrow}c]\phi$ prop	=	$t^*_{\Gamma,A}(Tm) \in \mathcal{M}[\Gamma \times TA]$

- Models with $\mathcal{C}$=category of sets, $\mathcal{M}$=collection of all monos:

  - non-determinism: TX=set of subsets of X, $(\Box p)(c)$ iff $c \subseteq p$;
  - side-effects (S non-empty set): $TX = (X \times S)^S$, $(\Box p)(c)$ iff $\forall s{:}S.p(\pi_1(cs))$

  We don't get a model by taking $TX = R^{R^X}$

[7] A dominion is a collection of monos close w.r.t. finite compositions and pullbacks along arbitrary morphisms.

- preservation of entailment

$$\frac{\Gamma, x{:}A \vdash \Phi, \phi \Longrightarrow \psi}{\Gamma, c{:}TA \vdash \Phi, [x{\Leftarrow}c]\phi \Longrightarrow [x{\Leftarrow}c]\psi} \quad x \notin \mathrm{FV}(\Phi)$$

- $\Box$-lift $\dfrac{\Gamma, x{:}A \vdash \phi \text{ prop}}{\Gamma, v{:}A \vdash [v/x]\phi \Longrightarrow [x{\Leftarrow}[v]]\phi}$

- $\Box$-let $\dfrac{\Gamma, x{:}A \vdash e{:}TB \qquad \Gamma, y{:}B \vdash \phi \text{ prop}}{\Gamma, c{:}TA \vdash [x{\Leftarrow}c]([y{\Leftarrow}e]\phi) \Longrightarrow [y{\Leftarrow}(\mathrm{let}_T\, x{\Leftarrow}c \,\mathrm{in}\, e)]\phi}$

- T on morphisms

$$\frac{\Gamma, x{:}A \vdash e{:}B \qquad \Gamma, y{:}B \vdash \phi \text{ prop}}{\Gamma, c{:}TA \vdash [y{\Leftarrow}(\mathrm{let}_T\, x{\Leftarrow}c \,\mathrm{in}\, [e]_T)]\phi \Longleftrightarrow [x{\Leftarrow}c]([e/y]\phi)}$$

- commutativity with $\wedge$

$$\frac{\Gamma, x{:}A \vdash \phi_i \text{ prop}}{\Gamma, c{:}TA \vdash \wedge_i([x{\Leftarrow}c]\phi_i) \Longleftrightarrow [x{\Leftarrow}c](\wedge_i \phi_i)}$$

- truth-pres $\dfrac{\Gamma \vdash \phi \text{ prop}}{\Gamma, c{:}TA \vdash \phi \Longrightarrow [x{\Leftarrow}c]\phi} \quad x \notin \mathrm{FV}(\phi)$

NB: the converse of ($\Box$-lift), ($\Box$-let) and (truth-pres) hold, when η, μ and t are $\mathcal{M}$-cartesian[9] (respectively).

[8] Valid in a category $\mathcal{C}$ with products, a dominion $\mathcal{M}$ and a strong monad T preserving pullbacks of monos in $\mathcal{M}$.

[9] A natural transformation $\sigma: F \to G$ is $\mathcal{M}$-cartesian, when for all $m \in \mathcal{M}$ the commuting square (asserting naturality of σ) for m is a pullback.

If $\mathcal{C}$ is a topos, $\mathcal{M}$ is the dominion of all monos in $\mathcal{C}$ and T as in topos-semantics, then

- the following formulas are semantically equivalent
  - $\Gamma, c{:}TA \vdash [x{\Leftarrow}c]\phi$
  - $\Gamma, c{:}TA \vdash (\mathrm{let}_T\, x{\Leftarrow}c \,\mathrm{in}\, [\phi]_T) =_{T\Omega} (\mathrm{let}_T\, x{\Leftarrow}c \,\mathrm{in}\, [\top]_T)$
  - $\Gamma, c{:}TA \vdash \Box(\mathrm{let}_T\, x{\Leftarrow}c \,\mathrm{in}\, [\phi]_T)$, where $\Box{:}T\Omega \to \Omega$ s.t. $\Box(c) = ([x{\Leftarrow}c]x)$

  i.e. necessity expressible in HOL

- possibility and evaluation definable using necessity

$$c{:}TA, v{:}A \vdash (c{\Downarrow}v) \overset{\Delta}{\Longleftrightarrow} \forall z{:}\Omega^A.([x{\Leftarrow}c]z(x)) \supset z(v)$$

$$\frac{\Gamma, x{:}A \vdash \phi \text{ prop}}{\Gamma, c{:}TA \vdash (\langle x{\Leftarrow}c\rangle\phi) \overset{\Delta}{\Longleftrightarrow} \forall w{:}\Omega.([x{\Leftarrow}c](\phi \supset w)) \supset w}$$

- if necessity commutes[10] with $\forall$ and $\supset$, i.e.

$\supset$-comm $\dfrac{\Gamma \vdash \phi \text{ prop} \qquad \Gamma, y{:}A \vdash \psi \text{ prop}}{\Gamma, c{:}TA \vdash \phi \supset [y{\Leftarrow}c]\psi \Longleftrightarrow [y{\Leftarrow}c]\phi \supset \psi}$

$\forall$-comm $\dfrac{\Gamma, x{:}B, y{:}A \vdash \phi \text{ prop}}{\Gamma, c{:}TA \vdash \forall x{:}B.[y{\Leftarrow}c]\phi \Longleftrightarrow [y{\Leftarrow}c]\forall x{:}B.\phi}$

then necessity and possibility expressible using evaluation

$$\frac{\Gamma, x{:}A \vdash \phi \text{ prop}}{\begin{array}{l}\Gamma, c{:}TA \vdash (\langle x{\Leftarrow}c\rangle\phi) \Longleftrightarrow (\exists x{:}A.c{\Downarrow}x \wedge \phi) \\ \Gamma, c{:}TA \vdash ([x{\Leftarrow}c]\phi) \Longleftrightarrow (\forall x{:}A.c{\Downarrow}x \supset \phi)\end{array}}$$

[10] The entailments in the direction $\Longleftarrow$ hold already.

Modalities $[c]p$ and $\langle c\rangle p$ of dynamic logic, with $c \in T1$ command (where $TX = \mathcal{P}(X \times S)^S$) and $p \subseteq S$ state-dependent assertion:

- $([c]p)(s)$ iff "$p(s')$ for every s' reachable from s by c",
- $\langle c\rangle p$ iff "$p(s')$ for some s' reachable from s by c".
- encoding in EL_T with operations for side-effects

$$\frac{\vdash c: T1 \qquad s': S \vdash \phi \text{ prop}}{\begin{array}{l} s: S \vdash ([c]\phi) \overset{\Delta}{\Longleftrightarrow} [s' \Leftarrow update(s); c; lookup]\phi \\ s: S \vdash (\langle c\rangle\phi) \overset{\Delta}{\Longleftrightarrow} \langle s' \Leftarrow update(s); c; lookup\rangle\phi \end{array}}$$

where $\vdash lookup = \lambda s.\{\langle s, s\rangle\}: TS$ and
$s: S \vdash update(s) = \lambda s'.\{\langle *, s\rangle\}: T1$.

NB: no need of *lookup* and *update* to expresses $[c]p$ and $\langle c\rangle p$ in the non-standard semantics for side-effects of [Pit90].

LIMITATIONS of EL_T EXPRESSIVENESS

Plotkin's powerdomain TX=non-empty subsets of $X_\perp$, divergence=$\{\perp\}$
Observational judgements about $c: TX$

- c "may terminate" iff $c \cap X$ inhabited iff $\langle x \Leftarrow c\rangle true$
- c "must diverge" iff $c = \{\perp\}$ iff $[x \Leftarrow c] false$
- c "may diverge" iff $\perp \in c$
- c "must terminate" iff $\perp \notin c$

NB: properties of concurrent computations $TX = \mu\gamma.\mathcal{P}(X + A \times \gamma)$ are even more problematic to express.

- K=category of finite cardinals[11] and injective maps
- $\mathcal{C}$=topos of presheaves, i.e. functors from K to Set
- $L \in \mathcal{C}$ type of location, $L(\_) = \_$
- $new: TL$ allocation operation, $new_k = \langle 1, k\rangle: TLk$

Strong monad T over $\mathcal{C}$ for dynamic allocation

- $TAk = \Sigma n: N.A(k + n)$
- $a: Ak \vdash \eta_k(a) = \langle 0, a\rangle: TAk$
- $x: Xk, c: TAk \vdash f^*_k(x, c) = \langle m + n, b\rangle: TBk$, where $\langle m, a\rangle = c$ and $\langle n, b\rangle = f_{k+m}(X(in^m_k)x, a)$

$A, B, X \in \mathcal{C}$, $f: X \times A \to TB$, $k \in K$ and $in^m_k: k \hookrightarrow k + m$.

PROPERTIES OF T

- T over $\mathcal{C}$ preserves pullbacks of monos
- $(\Box p)_k(c)$ iff $\forall\langle m, a\rangle = c.p_{k+m}(a)$, doesn't commute with $\supset$ and $\forall$

  $\forall x: L.[y \Leftarrow new]x \neq y$ holds, but $[y \Leftarrow new]\forall x: L.x \neq y$ doesn't

  $\phi^1 \supset [y \Leftarrow new]\phi^2$ holds, but $[y \Leftarrow new](\phi^1 \supset \phi^2)$ doesn't, where $\phi^i \subseteq 1$ s.t. ϕ^i_k iff $k \geq i$
- $c \Downarrow a$ iff $c = \langle 0, a\rangle$, cannot express observational judgements
- $(\Diamond p)_k(c)$ iff $\exists a: Ak.c \Downarrow a$ and $p_k(a)$

$A \in \mathcal{C}$, $p \subseteq A$ and $k \in K$.

[11] The cardinal k is the set of its predecessors.

PROBLEMATIC EXAMPLES of TOPOS-SEMANTICS

- K=category of finite cardinals[12] and injective maps
- $\mathcal{C}_j$=topos of $\neg\neg$-sheaves, i.e. pullback preserving functors from K to Set
- $L \in \mathcal{C}_j$ type of location, $L(\_) = \_$
- $new: TL$ allocation operation, $new_k = \langle 1, k\rangle : TLk$

Strong monad T over $\mathcal{C}_j$ for dynamic allocation

- $TAk = \Sigma n{:}N.A(k+n)$
- $a{:}Ak \vdash \eta_k(a) = \langle 0, a\rangle : TAk$
- $x{:}Xk, c{:}TAk \vdash f^*_k(x,c) = \langle m+n, b\rangle : TBk$, where $\langle m, a\rangle = c$ and $\langle n, b\rangle = f_{k+m}(X(in^m_k)x, a)$

$A, B, X \in \mathcal{C}_j$, $f{:}X \times A \to TB$, $k \in K$ and $in^m_k : k \hookrightarrow k+m$.

PROPERTIES OF T

- T over $\mathcal{C}_j$ preserves pullbacks of monos
- $(\Box p)_k(c)$ iff $\forall \langle m, a\rangle = c.p_{k+m}(a)$, commutes with $\supset$ but not with $\forall$

  $\forall x{:}L.[y \Leftarrow new]x \neq y$ holds, but $[y \Leftarrow new]\forall x{:}L.x \neq y$ doesn't
- $c{\Downarrow}a$ iff $\exists m{:}N.c = \langle m, a\rangle$, expresses observational judgements
- $(\Diamond p)_k(c)$ iff $(\Box p)_k(c)$, not expressible in terms of evaluation .

$A \in \mathcal{C}_j$, $p \subseteq_j A$ and $k \in K$.

[12] The cardinal k is the set of its predecessors.

- Simpler topos-semantics for possibility;
- Increased expressiveness of EL_T to cover other program logics;
- Test EL_T on new monads (e.g. variants of dynamic allocation, monads for local variables based on O'Hearn and Tennent's work).

GENERAL ISSUES

- Integration of EL_T and Axiomatic/Synthetic Domain Theory;
- Use of EL_T to specific examples (work in progress at Cambridge and Edinburgh).

References

[CP90] R.L. Crole and A.M. Pitts. New foundations for fixpoint computations. In *4th LICS Conf.* IEEE, 1990.

[Mog89] E. Moggi. Computational lambda-calculus and monads. In *4th LICS Conf.* IEEE, 1989.

[Mog91a] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1), 1991.

[Mog91b] E. Moggi. Computational types and logic: Evaluation logic. presented at the Meeting on Applications of Categories in Computer Science, Durham UK, 1991.

[Pit81] A.M. Pitts. *The Theory of Triposes.* PhD thesis, University of Cambridge, 1981.

[Pit90] A.M. Pitts. Evaluation logic. Technical Report 198, Computer Lab, Univ. of Cambridge, 1990.

CONSTRUCTING A FREGE STRUCTURE IN PREDICATIVE TYPE THEORY

Peter Dybjer
Chalmers

PLAN:

- FREGE STRUCTURES IN SET THEORY
  - definition
  - construction
- FREGE STRUCTURES IN TYPE THEORY
  - definition
  - construction
- DISCUSSION

AARHUS MARCH 92

WHAT IS A FREGE STRUCTURE?

THEORY

LTC = Logical Theory of Constructions

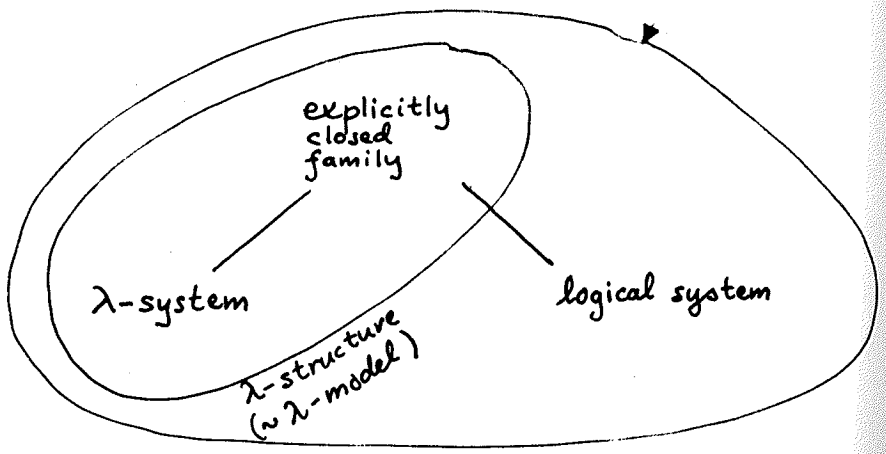
combines
{untyped λ-calculus
intuitionistic predicate logic

- consistent version of Grundgesetze FREGE / illative combinatory logic (CURRY)

MODEL

Frege Structure

ACZEL 1980



λ-STRUCTURES IN SET THEORY

- EXPLICITLY CLOSED FAMILY

$\mathcal{F}_n$ set

$\mathcal{F}_n \subseteq \mathcal{F}_0^n \to \mathcal{F}_0$ (n-ary functions on objects of $\mathcal{F}$)

$\pi_n^i \in \mathcal{F}_n$ (projections)

$K_n(a) \in \mathcal{F}_n \quad (a \in \mathcal{F}_0)$ (constant fns)

$g[fs] \in \mathcal{F}_m \quad (g \in \mathcal{F}_n,\ fs \in \mathcal{F}_m^n)$ (composition)

- λ-SYSTEM

$\dot{\lambda} \in \mathcal{F}_1 \to \mathcal{F}_0$

$\dot{ap} \in \mathcal{F}_0 \to \mathcal{F}_0 \to \mathcal{F}_0$

are $\mathcal{F}$-functionals, i.e. we have

$\dot{\lambda}_n \in \mathcal{F}_{n+1} \to \mathcal{F}_n$

$\dot{ap}_n \in \mathcal{F}_n \to \mathcal{F}_n \to \mathcal{F}_n$

such that

$\dot{\lambda}_n(f)(as) = \dot{\lambda}(\lceil f \rceil(as))$

$\dot{ap}_n(f,g)(as) = \dot{ap}(f(as), g(as))$

and we also have

$\dot{ap}(\dot{\lambda}(f), a) = f(a)$

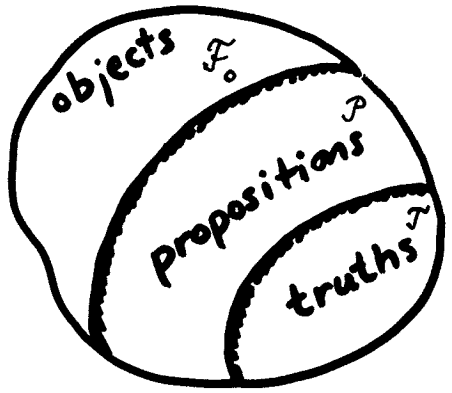
BUILDING λ-STRUCTURES

Standard constructions

- model of open terms
- domain-theoretic models

3

LOGICAL SYSTEMS IN SET THEORY



$\mathcal{P}, \mathcal{T}, \dot{\supset}, \dot{\wedge}, \dot{\vee}, \dot{\forall}, \dot{\exists}, \doteq$ such that

$\mathcal{T} \subseteq \mathcal{P} \subseteq \mathcal{F}_0$

$\dot{\supset} \in \mathcal{F}_n \to \mathcal{F}_n \to \mathcal{F}_n$

etc.

satisfying <u>logical</u> <u>schemata</u>

$(\mathcal{P}(a) \wedge (\mathcal{T}(a) \overset{*}{\supset} \mathcal{P}(b))) \supset$
$(\mathcal{P}(a \dot{\supset} b) \wedge$
$(\mathcal{T}(a \dot{\supset} b) \equiv (\mathcal{T}(a) \supset \mathcal{T}(b)))$ etc.

*Note non-standard notion of implication

BUILDING A LOGICAL SYSTEM

Idea: Logical schemata are introductory clauses of inductive definition for $\mathcal{P}$ and $\mathcal{T}$ (simultaneously)

Problem: $\mathcal{T}$ occurs negatively; the corresponding operator Θ is non-monotone.

Solution: Consider non-standard partial ordering

$$\langle \mathcal{P}, \mathcal{T} \rangle \prec \langle \mathcal{P}', \mathcal{T}' \rangle$$

iff

$$\mathcal{P} \subseteq \mathcal{P}' \quad \text{and} \quad \mathcal{T} = \mathcal{T}' \cap \mathcal{P}$$

Hence Θ monotone w.r.t. $\prec$ and has a least fixed point.
(in classical set theory)

Intuition: "Old truths" $\mathcal{T}$ remain the same when generating "new propositions" and "truths".

ACZEL (FITCH, FEFERMAN, SCOTT)

ACZEL (1977) p. 22

"The above is a standard construction in the theory of inductive definitions but it involves non-constructive reasoning at one or two points. For example the proof that families of $\mathcal{F}$-systems involved form chains requires that ordinals are comparable, and the proof that the increasing family of $\mathcal{F}$-systems Θ^α is eventually constant seems to require a non-constructive indirect argument. Nevertheless I believe the result to be constructively valid. A rigorous elaboration of this point would require an explicit discussion of the role of inductive definitions in constructive mathematics. It will have to suffice here if I simply assert that the logical schemata form the clauses of an inductive definition that generates the propositions and simultaneously give conditions for their truth."

NOTIONS OF THEORY AND MODEL IN TYPE THEORY

"Open" type theory has judgement forms

$$\Gamma, \Gamma \vdash \mathcal{J} \begin{cases} \alpha \text{ type} & \alpha = \alpha' \\ a : \alpha & a = a' : \alpha \end{cases}$$

Γ: intuitionistic "set theory"; Γ: context; $a : \alpha$: typing; $a = a' : \alpha$: definitional equalities

- Γ is the current fragment of intuitionistic "set theory" (MARTIN-LÖF) contains

declare typings of INTERPRETED entities $c : \alpha$, $a = a' : \alpha$:
- formation rules, introdution rules } inductive definitions
- typing rules, equality rules } recursive definitions

- Γ is the current context which declares typings of variables (UNINTERPRETED entities)

  $x : \alpha$

- "algebraic theory" = context

  $T \vdash \Gamma$ (theory, context)

  (more general than generalised algebraic theory CARTMELL)

- model = assigment of objects to variables $\gamma : \Gamma$ ($T \vdash \gamma : \Gamma$, $T' \vdash \gamma : \Gamma$ (TCT'))
  = INTERPRETING ENTITIES

λ-STRUCTURES IN TYPE THEORY

"algebraic theory" of λ-structures assumes "set theory" containing theory of n-tuples.

This needs $1, X, N$ and N_n defined inductively

$N_n : \text{set} \quad (n : N)$ — formation rules

$0_n : N_{s(n)} \quad (n : N)$
$s_n(i) : N_{s(n)} \quad (n : N, i : N_n)$ } introduction rules

Also A^n, f^n, R^n defined by N-recursion

$A^n : \text{set} \quad (A : \text{set}, n : N)$ typing rule
$A^0 = 1$
$A^{s(n)} = A^n \times A$ } equality rules

$f^n : (A^n)B^n \quad (A, B : \text{set}, n : N, f : (A)B)$ typing
$f^0(as) = \langle\rangle$
$f^{s(n)}(as) = \langle f^n(fst(as)), f(snd(as))\rangle$ } equality

$R^n : (A^n)(B^n)\,\text{prop} \quad (A, B : \text{set}, n : N, R : (A)(B)\,\text{prop})$ typing
$R^0(as, bs) = T$
$R^{s(n)}(as, bs) = R^n(fst(as), fst(bs)) \wedge R(snd(as), snd(bs))$ } α

and π^i_n defined by N_n-recursion

$\pi^i_n : (A^n)A \quad (A : \text{set}, n : N, i : N_n)$ typing
$\pi^{0_n}_{s(n)}(as) = snd(as)$
$\pi^{s_n(i)}_{s(n)}(as) = \pi^i_n(fst(as))$ } equality

8

EXPLICITLY CLOSED FAMILIES IN TYPE THEORY

- EXPLICIT TREATMENT OF EQUALITY (NO QUOTIENTS) CF BISHOP

  set = preset + equivalence relation

- FREGE STRUCTURE = MODEL OF LTC
  WOULD LIKE TO CONSIDER INTENSIONAL MODELS

INTENSIONAL λ-STRUCTURE	λ-ALGEBRA
(EXTENSIONAL) λ-STRUCTURE	λ-MODEL

- HENCE CONSIDER MULTICATEGORIES IN GENERAL INSTEAD OF EXPLICITLY CLOSED FAMILIES
- IT'S ALMOST A GENERALIZED ALGEBRAIC THEORY BUT ASSUMES THEORY OF N-TUPLES

THEORY OF MULTICATEGORIES WITH ONE OBJECT:

$\mathcal{F} : (n:N)\ \mathrm{set}$ — n-arrows

$\mathrm{var} : (n:N)(i:N_n)\ \mathcal{F}_n$ — projections

$-[-] : (m,n:N)(g:\mathcal{F}_n)(fs:\mathcal{F}_m^n)\ \mathcal{F}_m$ — composition

EXPLICIT EQUALITY OF ARROWS

$\sim : (n:N)(f,f':\mathcal{F}_n)\ \overset{\mathrm{set}}{\mathrm{prop}}$ — equality

ref : ...
sym : ...
trans : ... } equivalence relation

compcong : ... — congruence

MULTICATEGORY AXIOMS

$\mathrm{projax} : (m,n:N)(i:N_n)\ \mathrm{var}_n(i)[fs] \sim_m \pi_n^i(fs)$

$\mathrm{assoc} : (l,m,n:N)(h:\mathcal{F}_n)(gs:\mathcal{F}_m^n)(fs:\mathcal{F}_l^m)\ (h[gs])[fs] \sim_l h[gs \circledcirc fs]$

(ENOUGH POINTS

$\mathrm{ext} : (n:N)(g,g':\mathcal{F}_n)((m:N)(fs:\mathcal{F}_m^n)\ g[fs] \sim_m g'[fs]) \quad (fs \sim fs')$
$g \sim_n g'$)

+ CONSTANT FUNCTIONS TO GET EXPL CL FAM

THEORY OF λ-SYSTEMS IS OBTAINED BY ADDING THE FOLLOWING :
(CF CHURCH ALGEBRAIC THEORY OBTULOWICZ)

$\dot\lambda : (n:N)(f:\mathcal{F}_{s(n)})\,\mathcal{F}_n$

$\dot{ap} : (n:N)(g,f:\mathcal{F}_n)\,\mathcal{F}_n$

$\xi : (n:N)(f,f':\mathcal{F}_{s(n)})(f\sim_{s(n)} f')\ \dot\lambda_n(f)\sim_n \dot\lambda_n(f')$

apcong : $(n:N)(f,f',g,g':\mathcal{F}_n)(f\sim_n f')(g\sim_n g')\,\dot{ap}_n(g,f)\sim_n \dot{ap}_n(g',f')$

$\beta : (n:N)(g:\mathcal{F}_{s(n)})(f:\mathcal{F}_n)$
$\dot{ap}_n(\dot\lambda_n(g),f)\sim_n g[\langle \mathrm{var}^{(n)}_n(\mathrm{id}_n), f\rangle]$

λsub : $(m,n:N)(f:\mathcal{F}_{s(n)})(g^s:\mathcal{F}^n_m)$
$(\dot\lambda_n(f))[g^s]\sim_m \dot\lambda_m(f[\langle g^s \circ \mathrm{var}^{(n)}_{s(n)}(\uparrow_n), \mathrm{var}_{s(n)}(0)\rangle])$

apsub : $(m,n:N)(g,f:\mathcal{F}_n)(g^s:\mathcal{F}^n_m)$
$\dot{ap}_n(g,f)[g^s]\sim_m \dot{ap}_m(g[g^s], f[g^s])$

(INTERPRETED CONSTANTS
$\mathrm{id} : (n:N)\,N^n_n$
$\uparrow : (n:N)\,N^n_{s(n)}$
} defined by N-recursion

var^n_m } are defined from var_m and $\_[\_]$ by
$\circ$ } mapping)

CF CURIEN , CURIEN & RIOS , ABADI & CARDELLI & LÉVY & CURIEN .

BUILDING A λ-STRUCTURE IN TYPE THEORY

INTENSIONAL λ-STRUCTURES

- STANDARD FREE INTERPRETATION OF THE AXIOMS AS FORMATION AND INTRODUCTION RULES OF (AN) INDUCTIVE DEFS

	formation	introduction
$\mathcal{F}$		var, $\dot\lambda$, $\dot{ap}$, $\_[\_]$
$\sim$		ref, sym, trans, compcong, ... etc.

→ TYPE-THEORETIC FORMALIZATION OF λσ-CALCULUS OF CURIEN et.al.

- USE DISTINCTIONS

inductive vs. recursive

and

$f\sim_n f' : \mathrm{prop}$ vs $f \overset{\text{definitional equality}}{=} f' : \Lambda_n$

inductive	$\mathcal{F}$	var, $\dot\lambda$, $\dot{ap}$,
$\mathcal{F}$-recursive	$\_[\_]$	
inductive	$\sim$	...

→ MORE EFFICIENT CONSTRUCTION

EXTENSIONAL λ-STRUCTURES ?

LOGICAL SYSTEMS IN TYPE THEORY

$\mathcal{P}(a)$: set $(a : \mathcal{F}_0)$ $\quad$ $\mathcal{P}cong(\dots) : \mathcal{P}(a')$ $(a, a' : \mathcal{F}_0, \mathcal{P}(a))$

$\mathcal{T}(a)$: set $(a : \mathcal{F}_0)$ $\quad$ $\mathcal{T}cong(\dots) : \mathcal{T}(a')$ $(a, c' : \mathcal{F}_0, \mathcal{T}(a))$

$a \supset b : \mathcal{F}_0$ $(a, b : \mathcal{F}_0)$ $\quad$ $\supset cong(\dots) : a' \supset b'$ $(a, a', b, b' : \mathcal{F}_0, a' \supset b$

(etc, for the other logical constants ...)

Logical schemata

$(\mathcal{P}(a) \textcircled{\wedge} (\mathcal{T}(a) \supset \mathcal{P}(b))) \textcircled{\supset}$
$\mathcal{P}(a \supset b) \textcircled{1}$
$(\mathcal{T}(a \supset b) \textcircled{\equiv} \mathcal{T}(a) \textcircled{\supset} \mathcal{T}(b))$ (etc...)

(Remark: the "algebraic theory" of logical systems
- extends the "algebraic theory" of explicitly closed families
- uses the interpreted constants $\supset, \wedge, \vee, \exists, \forall$)

BUILDING LOGICAL SYSTEMS IN TYPE THEORY

- The inductive definition of a (family of) sets P can be done simultaneously with the P-recursive definition of a function f (the introduction rules for P refer to f)

PRIME EXAMPLE: THE FIRST TYPE-THEORETIC UNIVERSE (À LÀ TARSKI: MARTIN-LÖF 1984)

$\begin{cases} U : \text{set} & \text{defined inductively} \\ T : (U)\,\text{set} & \text{defined by } U\text{-recursion} \end{cases}$

PROPOSITIONS AND TRUTHS IN A FREE STRUCTURE CAN BE DEFINED SIMILARLY

$\begin{cases} \mathcal{P} : (\mathcal{F}_0)\,\text{set} & \text{defined inductively} \\ \mathcal{T}_{\mathcal{P}} : (a : \mathcal{F}_0)(\mathcal{P}(a))\,\text{set} & \text{defined by } \mathcal{P}\text{-recursion} \end{cases}$

NOTE THAT WE DEFINE THE AUXILIARY FAMILY

$\mathcal{T}_{\mathcal{P}}$ - truth of a proposition

AND THEN

$\mathcal{T}$ - truth of an object (not assumed to be a prop)

WILL BE DEFINED BY

$$\mathcal{T}(a) = \exists p : \mathcal{P}(a).\ \mathcal{T}_{\mathcal{P}}(a, p)$$

"predicative" negative occurrence

$\mathcal{P}$-introduction

$$\frac{a:\mathcal{F}_0 \quad p:\mathcal{P}(a) \quad b:\mathcal{F}_0 \quad q(x):\mathcal{P}(b) \quad c:\mathcal{F}_0 \quad z:c\sim_0 a\,\dot{\supset}\,b}{\dot{\supset}intro(a,p,b,q,c,z):\mathcal{P}(c)}\;(x:\mathcal{T}_p(a,p))$$

$\mathcal{T}_p$-equality

$$\mathcal{T}_p(c,\dot{\supset}intro(a,p,b,q,c,z)) = \forall x:\mathcal{T}_p(a,p).\mathcal{T}_p(b,q(x))$$

Proof that this yields a logical system

- $\mathcal{P}cong$: that $\mathcal{P}$ preserves $\sim$ is proved by $\mathcal{P}$-induction (universe-elimination-like)
- $\mathcal{T}cong$: that $\mathcal{T}$ preserves $\sim$ is proved as a corollary of the fact that $\mathcal{T}_p$ preserves $\sim$ by double $\mathcal{P}$-induction and

$$\mathcal{T}(a) \equiv \mathcal{T}_p(a,p) \quad (a:\mathcal{F}_0, p:\mathcal{P}(a))$$

- Logical schema follows by this too

SUMMARY

- Construction of $\mathcal{P}$ and $\mathcal{T}$ (logical system)
  - precise justification of Aczel's claim
  - providing explicit account of inductive definitions in type theory
  - using explicit proof-objects and explain in terms of a simultaneous inductive-recursive definition
  - "refinement" of Aczel's set-theoretic explanation
- Construction of two variants of $\lambda\sigma$-calculus
  - good candidate for formalization of λ-calculus à la Barendregt e.g.
  - makes use of distinctions between different kind of equalities (definitional, propositional)
- Notions of "theory", "model" in type theory
  - type theory as a foundation for constructive category theory ?

QUESTION

- Categorical vs Natural Deduction formulation of inductive definitions (cf MENDLER 91)

  Initial algebra view not sufficient !!

  (Category-theoretic semantics abstraction of set-theoretic semantics, explains construction as an inductive definition)

TOPICAL CATEGORIES OF DOMAINS

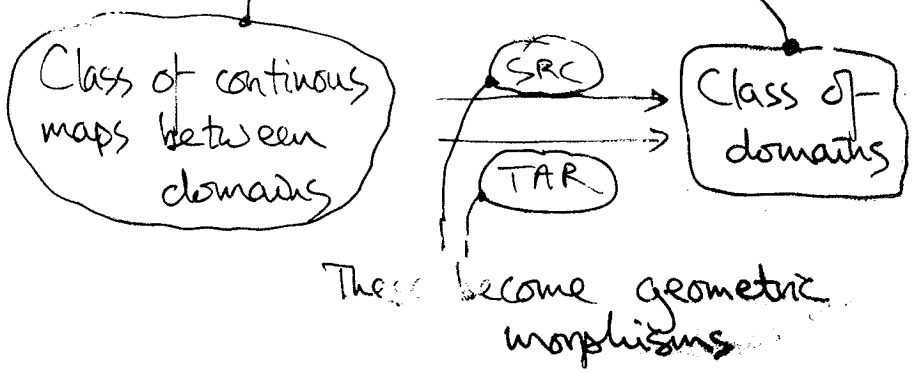
Steve Vickers
Imperial

Internal in category of Grothendieck toposes and geometric morphisms

"Always topologize" - Stone?

"Toposes are generalized topological spaces" - Grothendieck

Topologizing category of domains:



Information Systems

- Scott information systems: (Scott)
  point of Scott domain = join of tokens
- algebraic information systems: (?)
  point of algebraic poset = directed join of tokens
- continuous Scott information systems (Hoofman)
- continuous information systems (Vickers)
- categorical algebraic: (?)
  object of algebraic category = filtered colimit of "tokens"

In every case —

- information systems are models of a geometric theory $\mathbb{V}_{\mathcal{J}}$
- so are approximable mappings (account for continuous maps)
- two classifying toposes → topical category

Strongly algebraic (SFP) domains: topical c.c.c.

Puts Abramsky's "Domain theory in logical form" in topical form.

Terminology & Notation

A Grand frame — what would usually be defined as a Grothendieck topos.

A (Grothendieck) topos $\mathcal{D}$ is equipped with a Grand frame $\mathcal{S}\mathcal{D}$ (objects of $\mathcal{S}\mathcal{D}$ are sheaves over $\mathcal{D}$)

A Grand frame homomorphism $A \to B$ is adjoint pair $f^* \dashv f_*$, $f^* : A \to B$ preserving finite limits

A geometric morphism $\mathcal{D} \to \mathcal{E}$ is a Grand frame homomorphism from $\mathcal{S}\mathcal{E}$ to $\mathcal{S}\mathcal{D}$.

If widgets are the models of some geometric theory, then [widget] is its classifying topos.

$\mathbb{V}\mathcal{B} = \mathcal{S}[\text{widget}]$ — sheaves over topos of widgets; Grand frame of Sets with generic widget adjoined

Finite power sets as geometric

X an object of a Girard frame.

Write $\mathcal{F}X$ for free semilattice over X

It's nature is preserved by Girard frame homomorphisms, i.e. construction $\mathcal{F}$ "is geometric"

If X is a sort in a geometric theory, we are free to use $\mathcal{F}X$ as a derived sort.

Elements of $\mathcal{F}X$ are (Kuratowski) finite subsets of X. They're of form

$$S = \{x_1, \ldots, x_n\}$$

Observationally: We can list elements of S finitely. (We don't necessarily know x_i's are distinct.

Universal quantification is geometric - provided its bounded over a finite set.

$$\forall x \in S.\ \phi(x) \equiv_{def} \bigvee_{n \in \mathbb{N}} \exists x_1, \ldots, x_n.\ \left(S = \{x_1, \ldots, x_n\} \wedge \bigwedge_{i=1}^{n} \phi(x_i)\right)$$

Strongly algebraic information systems

- One sort X.
- Predicates $\sqsubseteq$ and $\not\sqsubseteq$ $X \times X$
- Axioms - $\sqsubseteq$ is a preorder
  $\not\sqsubseteq$ is its negation
- Predicate $CUB : \mathcal{F}X \times \mathcal{F}X$
- Axioms:
  define $\neg CUB(S,T) \equiv_{def}$
  $\exists s \in S.\ \exists t \in T.\ s \not\sqsubseteq t$
  $\vee\ \exists u.\ (\forall s \in S.\ s \sqsubseteq u \wedge \forall t \in T.\ t \not\sqsubseteq u)$
  i.e. T is not a complete set of upper bounds of S.
  require - CUB is negation of $\neg CUB$
- Axiom - every finite set has a finite complete set of upper bounds
  $\vdash \exists T : \mathcal{F}X.\ CUB(S,T)$
- Axiom - every finite set has a finite "CUB-closure":
  $\vdash \exists M : \mathcal{F}X.\ (S \subseteq M \wedge CUBcl(M))$

where
$CUBcl(M) \equiv_{def} \forall S \subseteq M.\ \exists T \subseteq M.\ CUB(S,T)$

Classifying topos [IS] - topos "of information systems"

Homomorphisms between information systems

$f : X_1 \longrightarrow X_2$

f preserves $\Subset$ - it's monotone

f preserves $\not\Subset$ - it's an order embedding

f preserves CUB

can show f's continuous extension $\mathrm{Id}\,X_1 \to \mathrm{Id}\,X_2$ has continuous right adjoint

Homomorphisms ~ embedding projection pairs.

Approximable mappings

- Two sorts X_1, X_2
- Predicates $\Subset_i$, $\not\Subset_i$, CUB_i $(i=1,2)$
- Axioms to make each X_i a strongly algebraic information system
- Predicate $f \subseteq X_1 \times X_2$
- Axioms to make f an approximable mapping:

$$s \Subset s' \wedge f(s,t) \vdash f(s',t)$$

$$f(s,t) \wedge t' \Subset t \vdash f(s,t')$$

$$\vdash \exists t.\, f(s,t)$$

$$f(s,t_1) \wedge f(s,t_2) \vdash \exists t.\,(t_1 \Subset t \wedge t_2 \Subset t \wedge f(s,t))$$

Two information systems X_1, X_2 in $\mathcal{S}[\mathrm{AM}]$

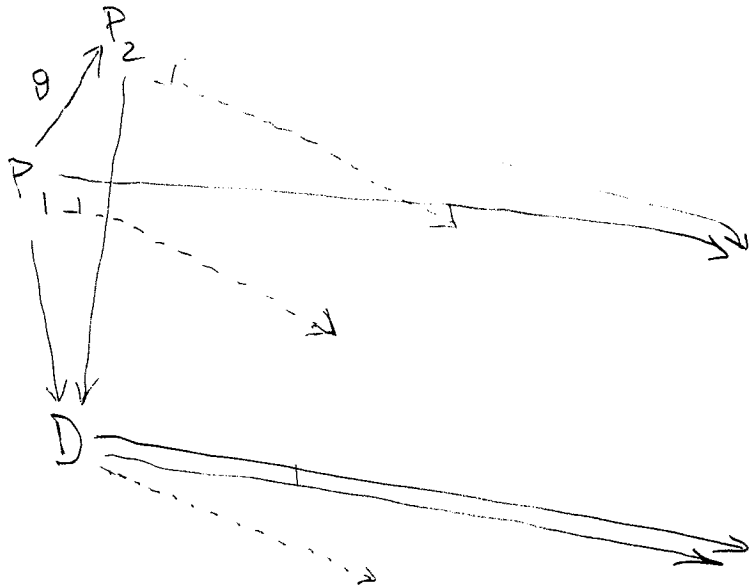
Two Girard frame homomorphisms

$\mathcal{S}\,\mathrm{SRC}$ (for X_1) and $\mathcal{S}\,\mathrm{TAR}$ (X_2): $\mathcal{S}[\mathrm{IS}] \to \mathcal{S}[\mathrm{AM}]$

- Two geometric morphisms

$$[\mathrm{AM}] \underset{\mathrm{TAR}}{\overset{\mathrm{SRC}}{\rightrightarrows}} [\mathrm{IS}]$$

Continuous maps between strongly algebraic predomains are equivalent to approximable mappings



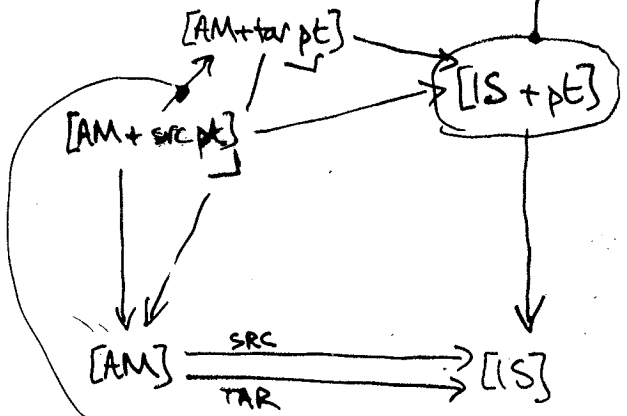
Note: internal category structure given by geometric morphisms into [AM]:

$[IS] \xrightarrow{ID} [AM]$

$[AM] \times_{[IS]} [AM] \xrightarrow{COMP} [AM]$

The generic strongly algebraic predomain

Classifies theory of:
strongly algebraic
IS $X, \Subset, \not\Subset, CUB$
+ unary predicate
$I(s)$ (set)
+ axioms to make
I an ideal of
X



In $\mathcal{E}[AM + src\ pt]$:
have ISs X_1, X_2
AM $f: X_1 \to X_2$
I_1 ideal of X_1
Define
$I_2(t) = \exists s. \left(I_1(s) \wedge f(s,t)\right)$

Proof

in $\mathcal{SP}_1$, have —

model of theory classified by $\mathcal{D}$,

including two IS's X_1, X_2

($X_1 = \mathcal{S}\,SRC(X)$

$X_2 = \mathcal{S}\,TAR(X)$)

+ generic ideal x of X_1

ϑ is given by ideal $\vartheta(x)$ of X_2

Continuous map ϑ is point $\vartheta(x)$ of target where x is a generic source point

X_1 — internal category in $\mathcal{SD}$

ideal of X_1 = flat internal presheaf on X_1

DIACONESCU ideals of X_1 classified by topos whose sheaves are internal diagrams on X_1

$(\mathcal{SD})^{X_1}$.

Get concrete representation of $\mathcal{SP}_1$

Can calculate points of X_2 in $\mathcal{SP}_1$

They're equivalent to approximable mappings from X_1 to X_2 in $\mathcal{SD}$

Function spaces

$$EXP : [IS]^2 \longrightarrow [IS]$$

Write X_1, X_2 for two generic IS's in $\mathcal{S}[IS]^2$, $[X_1 \to X_2]$ for $\mathcal{S} \in EXP(X)$.

$[X_1 \to X_2]$ is subobject $C(\mathcal{U})$ of $\mathcal{P}(X_1 \times X_2)$

$$C(\mathcal{U}) \equiv_{def} \forall V \subseteq \mathcal{U}.\ \exists W \subseteq \mathcal{U}.\ (CUB(\mathcal{P}\pi_1(V), \mathcal{P}\pi_1(W)) \wedge \forall v \in V.\ \forall w \in W.\ \pi_2(v) \subseteq \pi_2(w))$$

$$\mathcal{U} \sqsubseteq \mathcal{U}' \equiv_{def} \forall u \in \mathcal{U}.\ \exists u' \in \mathcal{U}'.\ (\pi_1(u') \subseteq \pi_1(u) \wedge \pi_2(u') \supseteq \pi_2(u'))$$

$$\mathcal{U} \not\sqsubseteq \mathcal{U}' \equiv_{def} \exists u \in \mathcal{U}.\ \forall u' \in \mathcal{U}'.\ (\pi_1(u') \not\subseteq \pi_1(u) \vee \pi_2(u) \not\supseteq \pi_2(u'))$$

$$CUB(\mathcal{U}, \mathcal{U}') \equiv_{def} \forall u \in \mathcal{U}.\ \forall u' \in \mathcal{U}'.\ u \sqsubseteq u'$$

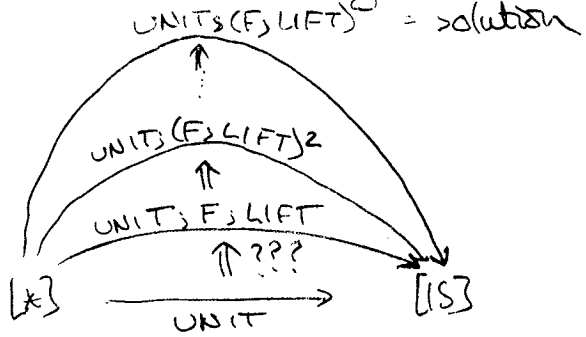
$$\wedge\ \exists M_1 : \mathcal{P}X_1.\ \exists M_2 : \mathcal{P}X_2.\ CUBd(M_1) \wedge \mathcal{P}\pi_1(\cup\mathcal{U}) \subseteq M_1 \wedge CUBd(M_2) \wedge \mathcal{P}\pi_2(\cup\mathcal{U}) \subseteq M_2 \wedge \forall R \subseteq_{fin} M_1 \times M_2.\ ((CCL(R) \wedge \forall u \in \mathcal{U}.\ u \subseteq R) \to \exists u' \in \mathcal{U}'.\ u' \subseteq R)$$

Get internal c.c.c.

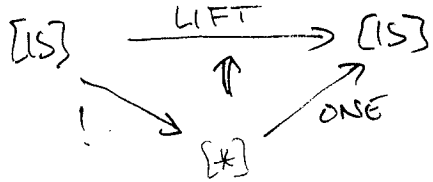
Fixpoints: solving domain equations

Points of any topos are closed under filtered colimits.

$$\mathbb{D} \cong F(\mathbb{D})_\perp$$

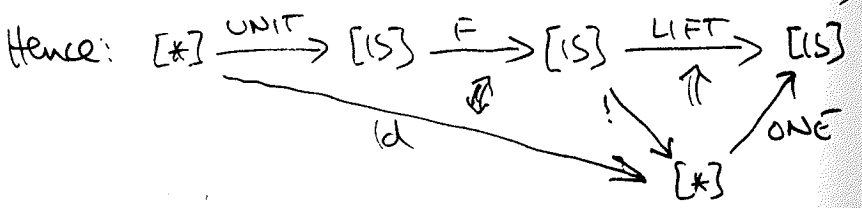


Starting it off:



in $\mathcal{S}[IS]$: homomorphism from 1 to $\mathcal{S}LIFT(X)$

Hence:



ML style polymorphism

(Girard, Reynolds)

Polymorphic types = geometric morphisms: $[IS]^n \to [IS]$

e.g. $\alpha \to \beta$ $\qquad [IS]^2 \xrightarrow{EXP} [IS]$

$\alpha \to \alpha$ $\qquad [IS] \xrightarrow{\Delta} [IS]^2 \xrightarrow[EXP]{} [IS]$

Polymorphic terms – lots of homes

e.g. polymorphic identity:

- 

- $[IS] \xrightarrow{ID} [AM]$

$$[IS] \longrightarrow [IS+pt]$$
$$\sigma \longmapsto (\phi, \lambda x^{\sigma}.\, x)$$
$$\sigma \to \sigma,$$

Other topical c.c.c.'s ?

- HOT [function] ⟹ [set]
  (morphism & object classifiers)

? • Continuous domains:
  ? Jung's FS (finitely separable) domains

? • Categorical domains
  ? Categorical analogue of strongly algebr
  ("sequence of finite categories")

? • Smyth-Jung theorems on maximal topical c.c.c.'s (e.g. maximal algebraic)

VARIATIONS ON THE BAGDOMAIN THEME

P.T. JOHNSTONE

D.P.M.M.S., UNIVERSITY OF CAMBRIDGE

References

S. Vickers, Geometric theories and databases, in Proc. 1991 Durham Symposium

P.T. Johnstone, Partial products, bagdomains and hyperlocal toposes, Ibid.

P.T. Johnstone, Fibrations and partial products in a 2-category, in preparation

R. Street, Fibrations in bicategories, Cahiers Top. Géom. Diff. 21 (1980), 111-160.

R. Dyckhoff & W. Tholen, Exponentiable morphisms, partial products and pullback complements, J. Pure Appl. Alg. 49 (1987), 103-116.

P.T. Johnstone & A. Joyal, Continuous categories and exponentiable toposes, J. Pure Appl. Alg. 25 (1982), 255-296.

(Lower) Bagdomain $\mathcal{B}_L(\mathbb{D})$ [Vickers 1990]

Given a domain $\mathbb{D}$, want a 'domain' B whose points are bags (= set-indexed families) $(d_i \mid i \in I)$ of points of $\mathbb{D}$, with 'refinement ordering' given by maps $(d_i \mid i \in I) \longrightarrow (e_j \mid j \in J)$ corresponding to functions $f : I \longrightarrow J$ with $d_i \leq e_{f(i)}$ for all i.

$\mathcal{B}_L(\mathbb{D})$ is (not a locale but) a topos:
its localic reflection is the lower (Hoare) powerdomain $\mathcal{P}_L(\mathbb{D})$
(On points, the reflector sends $(d_i \mid i \in I)$ to $\{d_i \mid i \in I\}^-$)

More generally, given a Grothendieck topos $\mathcal{E}$
(classifying a geometric theory $\mathbb{T}$, say)
we can build a topos $\mathcal{B}_L(\mathcal{E})$ classifying the theory "$\mathbb{T}$-bag of bags of $\mathbb{T}$-models".

$\mathcal{B}_L$ is a functor $\underline{\underline{\mathrm{BTop}}}/\mathcal{S} \longrightarrow \underline{\underline{\mathrm{BTop}}}/\mathcal{S}$, and carries a (2-)doctrine structure:
its algebras ("hyperlocal toposes") classify theories with 'natural' finite coproducts.

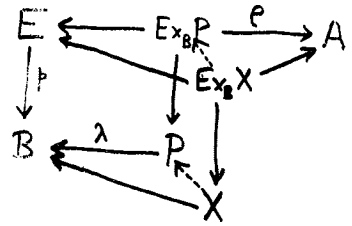
Can view $\mathcal{B}_L$ as an extension of the scone (= lift) monad on toposes, whose algebras are local toposes [Johnstone-Moerdijk 1989].

Problems

- Can we develop a similar theory of upper/mixed bagdomains?
- What is the general construction on toposes of which $\mathcal{B}_L$ and the scone monad are special cases?

Answer: partial products.

Partial Product $P = P\left(\begin{smallmatrix} E \\ \downarrow p \\ B \end{smallmatrix}, A\right)$ [Pasynkov 1965, Dyckhoff 1984]



Examples $P(1_B, A) = B \times A$

$P\left(\begin{smallmatrix} B \\ \downarrow 1 \end{smallmatrix}, A\right) = A^B$

Lemma [Dyckhoff-Tholen 1987] $P(p, A)$ exists for all A
$\Leftrightarrow$ p is exponentiable in $\mathcal{E}/B$.
Moreover, $P(p, -)$ is then the functor

$$\mathcal{E} \xrightarrow{B^*} \mathcal{E}/B \xrightarrow{(-)^p} \mathcal{E}/B \xrightarrow{\Sigma_B} \mathcal{E}.$$

Examples The free monoid functor $MX = \Sigma_N X^{[n]}$ in a topos with NNO [cf. Johnstone-Wraith 1978]

More generally, the free functor for any algebraic theory which is a club in the sense of [Kelly 1992], i.e. has a presentation by operations and equations

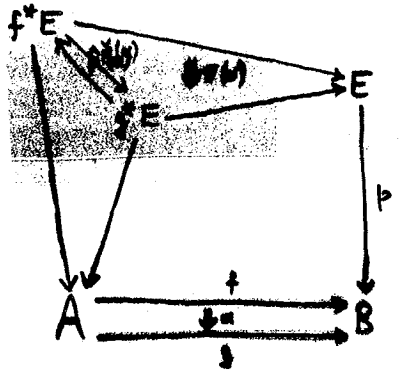
$$f(x_1, x_2, \ldots, x_n) = g(x_1, x_2, \ldots, x_n)$$

in which the same variables appear on each side in the same order (without repetition).

E.g. for TX = set of X-labelled trees, take
B = set of unlabelled trees (= $T1$)
$p^{-1}(b)$ = set of leaves (= vertices to be labelled) of b.

For partial products in a 2-category (like $\underline{\mathrm{Top}}$), we need to be able to 'lift' 2-cells from $\mathcal{B}$ to $\mathcal{E}$

either covariantly or contravariantly

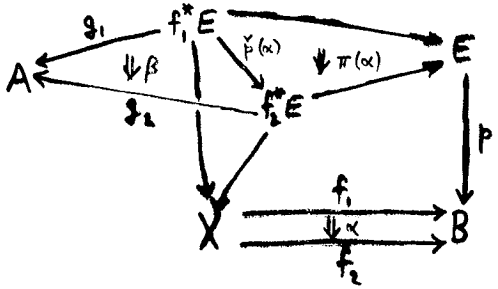


We call p an opfibration (a fibration) if this occurs (in a 'functorial' way).

Suppose p is an opfibration. We define the covariant partial product $P_!(p,A)$ to be a representing object (if it exists) for the functor $F_! : \underline{\mathcal{K}}^{op} \longrightarrow \underline{\mathrm{Cat}}$ given by

$$\mathrm{ob}\, F_!(X) = \{ (f,g) \mid f : X \longrightarrow \mathcal{B},\ g : \mathcal{E} \times_{\mathcal{B}} X \longrightarrow A \}$$

morphisms of $F_!(X)$ are pairs (α, β) as in



Similarly for the contravariant partial product $P^*(p,A)$

Warning: even if $P_!(p,A)$ and $P^*(p,A)$ both exist, they may not coincide.

However,

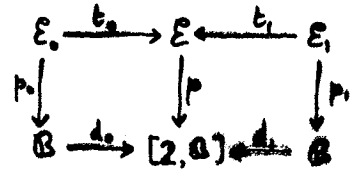
Lemma Suppose p is both an opfibration and a fibration, and suppose

$$\left(\check{p}(\alpha) \dashv \hat{p}(\alpha)\right)$$

for all α. Then $P_!(p,A)$ coincides with $P^*(p,A)$ if either exists.

This happens in all examples of interest (at least in $\underline{\mathrm{Top}}$):

Theorem Let $\mathcal{P}$ be a class of bounded geometric morphisms which is stable under pullback, and such that whenever $(p : \mathcal{E} \longrightarrow [2,\mathcal{B}]) \in \mathcal{P}$



and the above squares are pullbacks, the composite $t_0^* t_{1!}$ is the inverse (direct) image of a geometric morphism over $\mathcal{B}$. Then the members of $\mathcal{P}$ are opfibrations (fibrations). If further $(p : \mathcal{E} \longrightarrow \mathcal{B}) \in \mathcal{P}$ is representable in $\underline{\mathrm{BTop}}/\mathcal{B}$, then $P_!(p, \mathcal{A})$ $(P^*(p, \mathcal{A}))$ exists in $\underline{\mathrm{BTop}}/\mathcal{S}$ for any bounded $\mathcal{S}$-topos $\mathcal{A}$, provided $\mathcal{B}$ is bounded over $\mathcal{S}$.

Examples: $[\mathbb{C}, \mathcal{B}] \longrightarrow \mathcal{B}$ for any internal category $\mathbb{C}$ in $\mathcal{B}$.

$[\mathbb{C}^{op}, \mathcal{B}] \longrightarrow \mathcal{B}$ for any internal category $\mathbb{C}$ with finite limits.

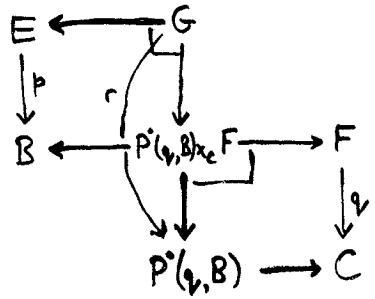
Any coherent topos over $\mathcal{B}$.

Plus (adjoint or coadjoint) retracts of the above.

When does $P(p,-)$ carry a monad structure?

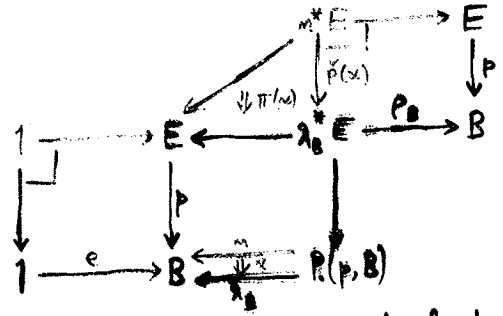
Lemma A composite of two partial product functors (of the same variance) is a partial product functor.

Proof

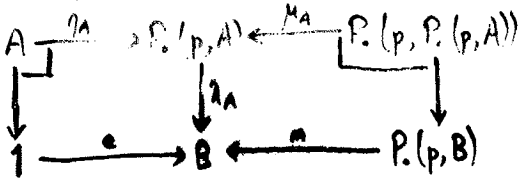


For any A, $P'(q, P'(p,A))$ and $P'(r,A)$ have the same universal property.

Suppose given $e : 1 \to B$, $m : P'(p,B) \to B$ and $\alpha : m \Longrightarrow \lambda_B$ s.t.



Then e and m are the 1-components of natural transformations $\eta : id \to P'(p,-)$, $\mu : P'(p, P'(p,-)) \to P'(p,-)$ obtained by



The monad identities for η and μ similarly reduce to diagrams involving e and m.

Examples in $\mathcal{B}\mathrm{Top}/\mathcal{S}$ (a) Covariant.

1 Take $\mathcal{B} = \mathcal{S}[\mathbb{C}]$ (the classifying topos for categories), $\mathcal{E} = [\mathbb{C}, \mathcal{B}]$ given by the terminal category $\mathbb{1}$, m by the Grothendieck construction. If $\mathcal{A}$ classifies a geometric theory $\mathbb{T}$, then $P'(p, \mathcal{A})$ classifies arbitrary (small) diagrams of $\mathbb{T}$-models.

2 Similarly for any quotient theory of categories whose models are closed under the Grothendieck construction:
e.g. posets, discrete categories, connected categories, filtered categories, ...
but not totally ordered sets.

3 Quasi-injective toposes (= coadjoint retracts of presheaf toposes) correspond to small categories + a left flat, idempotent profunctor comonad [Johnstone-Joyal 1982]. This is a geometric theory, so there's a generic one; and it defines a bagdomain functor.

(b) Contravariant:

1 (The upper bagdomain) Take $\mathcal{B} = \mathcal{S}[\mathbb{B}]$ (the classifying topos for Boolean algebras), $\mathcal{E} = \mathcal{S}h(\mathrm{spec}\,\mathbb{B})$ (= classifying topos for Boolean algebras with a distinguished prime filter). e given by the 2-element Boolean algebra, m by global sections.
Algebras for this monad are classifying toposes for theories whose models are preserved by global sections over Stone spaces.

2 More generally, take $\mathcal{B} = \mathcal{S}[\mathbb{D}]$ where $\mathbb{D}$ is the generic distributive lattice, and $\mathcal{E} = \mathcal{S}h(\mathrm{spec}\,\mathbb{D})$. (However, m is not given by global sections here.) Here 'bag of $\mathbb{T}$-models' = sheaf of $\mathbb{T}$-models on a coherent space.

3 (?) Can build a classifying topos for (distributive lattices equipped with an idempotent comonad on their spectrum): bags of $\mathbb{T}$-models will 'include' sheaves of $\mathbb{T}$-models on a compact Hausdorff space. Does this have a monad structure?

First Steps Towards an AXIOMATIZATION of DOMAINS in TYPE THEORY

T. Streicher
LMU, Munich (FRG)

main events in the SEMANTICAL branch of TCS

- DOMAIN THEORY
- CONSTRUCTIVE TYPE THEORIES

} denotational

- LINEAR LOGIC, GEOM. INTERACT.

} structural understanding of OPERATIONAL FEATURES

for the moment let us forget about operational features

GOAL : a unified (formal) framework for expressing

algorithmic (- MODULES (Programs)

descriptive (- SPECIFICATIONS

and proving relations between them

in order to express Types of modules

one needs strong sum Types

(e.g. $(\Sigma X : \text{Type})(X \to X) \times X$)

$\Rightarrow$ USE A SUFFICIENTLY STRONG THEORY of DEPENDENT TYES (eg. XCC + ITT)

a) use CONCEPT of DELIVERABLES in order to express SPECIFICATIONS

collections of modules are certain Types

specifications are pairs (S, P)

where S Type of structures/modules

$P : S \to \text{Prop}$ predicate selecting the admissible or intended modules

Type theory allows to

- manipulate specifications (functions)
- express relations between them (predicates)

} one may iterate this process : specifications of specifications...

6) the ALGORITHMIC COMPONENT

Tradit. TT can express only a

FRAGMENT of the

ALGORITHMIC WORLD

(excluded are: - nontermination
- infinite computations
(or e.g. streams))

a more important drawback:

higher order primitive recursive fcts.

can express

(almost) all terminating
algorithmic fcts.

BUT: often only in a CLUMSY way

(ex. primitive rec. def. of GGT

vs. the usual program
employing GENERAL RECURSION

Thus we WANT a TYPE THEORY
with a Type

$\mathbb{D}om$ of data domains

together with

$El\mathbb{D} : \mathbb{D}om \to Type$

turning domains as objects of $\mathbb{D}om$
into domains as Types

satisfying the minimum requirement
that there is a fixpoint operator

$$fix \in (\Pi D : \mathbb{D}om)(El\mathbb{D}(D) \to El\mathbb{D}(D)) \to El\mathbb{D}(D)$$

s.t. $D : \mathbb{D}om$, $f : El\mathbb{D}(D) \to El\mathbb{D}(D)$

$\vdash f(fix\,D\,f) = fix\,D\,f \in El\mathbb{D}(D)$

Surely that is not sufficient,
it does not capture the nature of domains.

BASIC QUESTION

- which notions are ASSUMED AXIOMATICALLY
- which notions are DERIVED

} comparison with the archetype LCF

FEATURES

- Horn logic, basic predicate $\sqsubseteq$
- least fixpoints + Scott induction

LCF is good for a lot of PRACTICAL PURPOSES

HOW CAN LCF be EXTENDED and IMPROVED

- STREGTHEN the LOGIC
- CHOOSE MORE ADEQUATE FUNDAMENTAL NOTIONS

THE PRAGMATIC WAY (?

formulate LCF inside ✗CC

→ the logic is much more expressive!
(HOL)

→ SOME EASY, BUT HELPFUL CHANGES

- axiomatise the domain

$$\Sigma : \begin{matrix} \top \\ | \\ \bot \end{matrix}$$

($\forall, \wedge$ + Phoa's AXIOM)

- approximation $\tilde{\sqsubseteq}$
expressed as observational inequivalence
- axioms expressing continuity and leastness of fixpoints

8

The several INDUCTION PRINCIPLES can be DERIVED

because

one can EXPRESS that a PREDICATE on a DOMAIN is ADMISSIBLE, i.e. is

, CLOSED under SUPs of ASCENDING CHAINS

whereas

in LCF the Logic is RESTRICTED in a way, s.t.

ONLY ADMISSIBLE PREDICATES ARE EXPRESSIBLE

(therefore: $\exists$, $\Rightarrow$ are excluded)

THE RADICAL APPROACH (9

AXIOMATIZE

SYNTHETIC DOMAIN THEORY

OUTLINE assume an impredicative collection or kind Type (e.g. Prop in XCC)

(here impredicative simply means CLOSURE under Π)

DOMAINS are SPECIFIED by a PREDICATE :

is_Dom : Type → Prop

EXPRESSING THAT

a type X is Σ-REPLETE (in the sense of

MORE EXPLICITELY: 10

- axiomatise the algebraic structure of Σ (i.e. $\wedge, \top, \bot, \forall$)
- PHOA's AXIOM

$$\begin{array}{ccc} \Sigma^{\Sigma} & \xrightarrow{\langle \lambda f.\, f\bot,\ \lambda f.\, f\top \rangle} & \Sigma \times \Sigma \\ & \searrow^{r} \quad \nearrow & \\ & \{(p,q) \in \Sigma\times\Sigma \mid p \trianglelefteq q\} & \end{array}$$

where $p \trianglelefteq q$

means $(\forall P : \Sigma \to \Sigma)$

$$P(p) = \top \to P(q) = \top$$

- SCOTT's AXIOM

$$(\forall \Phi : \Sigma^{N} \to \Sigma)\ \Phi(\lambda x.\top) = \top$$

$$\Rightarrow (\exists n \in N)\ \Phi(\lambda x < n.\top) = \top$$

CONSISTENCY or (11

the SEMANTICAL MOTIVATION

- consider the

ω-Set MODEL of TYPE THEORY

- interpret

Type as PER ω

and

is_Dom : Type → Prop

as the predicate

$(\lambda X : \text{Type})$

$(\forall Y, Z : \text{Type})\ (\forall f : Z \to Y)$

$[\, \Sigma^{f} \text{ iso} \Rightarrow X^{f} \text{ iso} \,]$

12

How should one interpret

$$\mathrm{Prop} \quad ?$$

usually one takes PERω

BUT THEREISA PROBLEM

in the PERω Model
we have STRONG SUMS
(or in other words:
sums of families of props indexed over
a prop are props again!)
but as $[\![\mathrm{Type}]\!] \neq [\![\mathrm{Prop}]\!]$
we get the

AXIOM of CHOICE / $A, B \in \mathrm{Type}$

$$(\forall x{:}A)(\exists y{:}B)\,P(x,y) \Rightarrow (\exists f{:}A\to B)(\forall x{:}A)\,P(x,fx)$$

13

If we apply the AXIOM of CHOICE
to SCOTT's AXIOM

then there were a function

$$M : (\Pi\,\Phi : \Sigma^N \to \Sigma)\; Eq^{\Sigma}(\Phi(\lambda x.\tau),\tau) \to N$$

such that

$$(\forall\,\Phi : \Sigma^N \to \Sigma)\; Eq^{\Sigma}(\Phi(\lambda x.\tau),\tau) \to Eq^{\Sigma}\big(\Phi(\lambda x{<}M(\Phi)),\tau\big)$$

by dependent type carrying
we then get from M a function

(14)

$$\mathcal{M} : \underbrace{\left[(\Sigma\,\Phi : \Sigma^N \to \Sigma)\ \xi^{\Sigma}_{\xi}(\Phi(\lambda x.\tau), \tau)\right]}_{\text{itself } \Sigma\text{-replete}} \longrightarrow N$$

as it is the equaliser of

$$\Sigma^{(\Sigma^N)} \underset{\lambda\Phi.\,\Phi(\lambda x.\tau)}{\overset{\lambda\Phi.\tau}{\rightrightarrows}} \Sigma$$

THUS as

- $\mathcal{M}$ is <u>monotonic</u> (it is a map between domains)
- the source of $\mathcal{M}$ has a <u>top</u> element
- N is <u>flat</u>

we have THAT: $\mathcal{M}$ is CONSTANT

15

Thus we can prove

$$(\exists n : N)(\forall \Phi : \Sigma^N \to \Sigma)$$

$$\Phi(\lambda x.\tau) = \tau \Rightarrow \Phi(\lambda x{<}n.\tau) = \tau$$

Therefore we get a CONTRADICTION

Let n_0 be such an n,

put $\boxed{\Phi \equiv \lambda p.\, p(n_0)}$

then $\Phi(\lambda x.\tau) = \tau$

and therefore also $\Phi(\lambda x{<}n_0.\tau) = \tau$

but, of course, also $\Phi(\lambda x{<}n_0.\tau) = \bot$

BUT: if $\bot = \tau$ then every domain contains at most one element

ANYWAY 16

SCOTT's AXIOM is VALID

in the MODEL where

$[\![Prop]\!] = \{ R \in PER_\omega \mid card(\omega/R) \leq 1 \}$

i.e. RESTRICTION to the

PROOF IRRELEVANCE MODEL

This collection of INTUITIONISTIC,

PROOF IRRELEVANT

PROPOSITIONS

is DEFINABLE

$$Prop_{int} \equiv (\Sigma P : Prop)\ (\forall p, q : P)\ Eq^P(p, q)$$

[$Prop_{class} \equiv (\Sigma P : Prop)\ \neg\neg P \to P$

! classifies regular subobjects !]

CONCLUSIONS & DISCUSSION (17)

- axiomatisation of domains in CTT is possible
- BUT one needs a careful distinction between
  - $Prop_{UI}$ (props whose proofs are computationally meaningfull)
  - $Prop_{int}$ (intuit. props. without computat. contents)
  - $Prop_{class}$ (2-valued classical logic)

  (⊆ full, reflective inclusions)
- equality between domain elements

  NOT as CONVERSION RULES

  BUT as AXIOM on the LEVEL of $Prop_{UI}$

```
Init XCC;

        (* Higher Order Logic Definitons *)

FALSE == {p : Prop} p ;

NOT == [p : Prop]  p -> FALSE ;

AND == [p : Prop] [q : Prop]  {r : Prop} (p -> q -> r) -> r ;

OR ==  [p : Prop] [q : Prop]  {r : Prop} (p -> r) -> (q -> r) -> r;

AEQ ==  [p : Prop] [q : Prop]  AND (p -> q) (q -> p) ;

ALL == [X : Type(0)] [p : X -> Prop]  {x : X} p x ;

EXIST ==  [X : Type(0)] [p : X -> Prop]  {q : Prop} ({x : X} ((p x) -> q)) -> q ;

eq == [X | Type(0)] [x , y : X] {P:X->Prop} (P x) -> (P y) ;

[ext : {X : Type(0)} {Y : X -> Type(0)} {f,g : {x : X} Y x}
          ({x:X} eq (f x) (g x))  ->  eq  f g ] ;


class == [p : Prop]  (NOT (NOT p)) -> p ;


                (* COMPREHENSION *)

[ Com : {X | Type(0)} {P : X -> Prop} Type(0) ] ;


[in : {X | Type(0)} {P | X -> Prop} (Com P) -> X ] ;


[in_pr : {X : Type(0)} {P : X -> Prop} {y : Com P} P (in y)] ;


[univ : {X | Type(0)} {P | X -> Prop} {Y | Type(0)}
          {f : Y -> X} {p : {y : Y} P (f y)}
          Y -> Com P                                ] ;


[univ_pr :  {X: Type(0)}  {P : X -> Prop}  {Y : Type(0)}
            {f : Y -> X}  {p : {y : Y} P (f y)}
           {y : Y} eq (f y) (in (univ f p y))       ] ;


[univ_uniq :  {X: Type(0)} {P : X -> Prop} {Y : Type(0)} {f , g : Y -> Com P}
              ({y : Y} eq (in (f y)) (in (g y)))  ->  eq f g                  ] ;
```



```
                (* Axiom of Unique Choice *)

EX_UNI == [X : Type(0)] [P : X -> Prop]
                AND (EXIST X P) ({x , y : X} (P x) -> (P y) -> eq x y)  ;

[ax_uni_choice :  {X , Y : Type(0)} {P : X -> Y -> Prop}
                  ({x:X} EX_UNI Y (P x))
                  -> EXIST (X->Y) ([f:X->Y] {x:X} P x (f x))   ] ;



        (* Natural Numbers *)

[N :Type(0)]  [zero : N]  [succ : N->N] ;

[R : {C | N -> Type(0)} {d : C zero} {e:{n : N} {y : C n} C (succ n)} {n:N} C n] ;

[  [C : N ->Type(0)] [d : C zero] [e : {n : N} {y : C n} C (succ n)] [n:N]
     R d e zero ==> d   ||   R d e (succ n)  ==> e n (R d e n)     ] ;

pred == R |([n:N]N)  zero  ([n:N] [m:N] n) ;



        (* SIGMA *)

[Sig : Type]  [top , bot : Sig] ;

def  == [x : Sig] eq top x  ;

[ false_pr  : NOT (def bot) ] ;

[and : Sig -> Sig -> Sig] ;

[ [x:Sig]  and top x ==> x   ||   and x top ==> x] ;

[and_pr :  ALL Sig ([x:Sig] ALL Sig ([y:Sig]  AEQ (def (and x y)) (AND (def x) (def y))))] ;
```

```
[or : Sig -> Sig -> Sig] ;

[ [x : Sig]  or top x ==> top  ||  or x top ==> top] ;

[or_pr :  ALL Sig ([x:Sig] ALL Sig ([y:Sig]  AEQ (def (or x y)) (OR (def x) (def y))))] ;


[join : (N -> Sig) -> Sig] ;

 shift == [p:N->Sig] [n:N]  p (succ n) ;

[[p:N->Sig]  join p  ==> or (p zero) (join (shift p)) ] ;

[join_pr :  ALL (N->Sig) ([p:N->Sig]
              AEQ (def (join p))(EXIST N ([n:N] def (p n))))] ;



                (* PHOA Axiom *)

[phoa : ALL (Sig->Sig) ([p:Sig->Sig] ALL (Sig->Sig) ([q:Sig->Sig]
           (eq (p bot) (q bot)) -> (eq (p top) (q top)) -> (eq p q)))] ;

extend == [x:Sig] [y:Sig] [z:Sig] or x (and y z) ;


                (*  SCOTT Axiom *)

step == R |([n:N] N-> Sig) ([n:N] bot) ([n:N] [f:N->Sig] [m:N] f (pred m)) ;

[scott : ALL ((N->Sig)->Sig) ([F:(N->Sig)->Sig] ALL (N->Sig)
         ([p:N->Sig]  (def (F p)) -> EXIST N ([n:N] def (F (step n)))  ))] ;


        (* Definitions of Domains *)

approx == [X | Type] [x , y :X] {p : X -> Sig} (def (p x)) -> def (p y) ;

LP == [X : Type(0)]  {x , y : X} (approx x y) -> (approx y x) -> eq x y  ;


epi_rel == [A : Type(0)] [X , Y | Type(0)] [f : X -> Y]

                ALL (X->A) ([p : X ->A]

                    EXIST (Y->A) ([q : Y -> A]{x : X} eq (p x) (q (f x))))  ;


mono_rel == [A : Type(0)] [X , Y | Type(0)] [f : X -> Y]

                  ALL (Y->A) ([p : Y->A] ALL (Y->A) ([q : Y->A]

                  ({x : X} eq (p (f x)) (q (f x)))  ->  eq p q ) ) ;
```



```
equ_rel  ==  [A : Type(0)] [X , Y | Type(0)] [f : X -> Y]   AND (epi_rel A f) (mono_rel A f)  ;

repl_rel  ==  [A : Type(0)]  [B : Type(0)]

                  {X , Y : Type(0)} {f : X -> Y}  (equ_rel A f) -> (equ_rel B f)  ;

Dom == [X : Type(0)] AND (LP X) (repl_rel Sig X) ;


          (* The Sigma Repletion *)

reg_pred == [X | Type(0)] [P : X -> Prop] {x : X} class (P x) ;

embed ==  [X | Type(0)] [x : X]  [p : X -> Sig]  p x ;

Sig_Repl  == [X : Type(0)]

               Com

              ([x : X]

                  EXIST (((X->Sig)->Sig)->Prop) ([P :((X->Sig)->Sig)->Prop]

                 {p : Prop}

                 ((reg_pred |((X->Sig)->Sig) P) -> {pr :{x:X} P (embed x)}

                        (mono_rel  Sig (univ (embed |X) pr)) -> p)

                  -> p ) ) ;


        (* Chain Domains *)

antitone == [f : N -> Sig]  {n : N} (def (f (succ n))) -> def (f n) ;

Omega_Inf  == Com antitone ;
```

Andrea Schalk

Power Locales

Abstract

As the category of locally compact locales is equivalent to the category of locally compact sober spaces we can view the monads given by the various power locale constructions as monads over topological spaces.

This setting makes it possible for us to determine the categories of algebras for some of these monads.

Every monotone endofunction of a cpo has a least fixed point intuitionistically.

Paul Taylor
Imperial College
pt@doc.ic.ac.uk

SERC "Foundational Structures in Computer Science"
ESPRIT/BRA "Categorical Logic in C.S."

Three well-known proofs of

	the existence of the transitive closure of a given binary relation $R \subseteq X \times X$	the least fixed point (Tarski's) theorem for $f : X \to X$
Down from above as meet in a lattice.	$TC(R) = \bigcap \{T : R \subseteq T \subseteq X \times X,\ T \text{ transitive}\}$	$LFP(f) = \bigwedge \{x : fx \leq x\}$
greatest element of least closed set	$TC(R) = \bigcup U \quad U \subseteq P(X \times X)$ where U is the least subset closed under: transitivity axioms + containing R	$LFP(f) = \bigvee U \quad U \subseteq P(X)$ joins: $\bot, \vee, \bigvee^{\uparrow}$ + application of f
from below: countable iteration of continuous (finitary) operation	$TC(R) = \bigcup_{n=0}^{\infty} U^n$	$LFP(f) = \bigvee_{n=0}^{\infty} f^n$
up		$LFP(f) = \bigvee_{\alpha \in On} f^{\alpha}$

FIRST ATTEMPT:

$U \subseteq X$ is smallest set closed under

$\bot,\ f,\ \bigvee^{\uparrow}$

so $U \ni \bot,\ f\bot,\ f^2\bot, \ldots, f^n\bot, \ldots\ f^{\omega}\bot \stackrel{\text{def}}{=} \bigvee_n^{\uparrow} f^n\bot$

if f is continuous,

- $f(\bigvee_n^{\uparrow} f^n\bot) = \bigvee_n^{\uparrow} f^{n+1}\bot \geq f^n\bot$
- this is all that U contains.
- $f^{\omega}\bot$ is the greatest element of U and the least fixed point of f.

if U is directed

then it has a greatest element and this is the least fixed point.

how to prove it's directed?

$\forall u, v \in U.\ \exists w \in U.\ u \leq w \wedge v \leq w$

"by simultaneous induction on the reason why $u, v \in U$"

(PROOFS)

$$\frac{}{\bot \in U} \qquad \frac{x \in U}{fx \in U} \qquad \frac{x_i \in U \ (i \in I, \text{directed})}{\bigvee x_i \in U}$$

free algebra of proofs, P

well-founded (immediate) subproof relation, $\prec$

but:
- size of P?
- $\bigvee$ is a partial operation
- choice of proofs, to show that $P \to U$ is closed under $\bigvee$
- choice of w

Maybe these aren't problems (for a proof theorist) but I'm not confident that they can be dealt with intuitionistically

① OBSERVE $\prec \longmapsto \leq$

$$P\left\{\begin{array}{c} \cdots \quad P'\left\{\begin{array}{c}\vdots \\ x' \in U\end{array}\right. \quad \cdots \\ \hline x \in U \end{array}\right.$$

$P' \prec P \Rightarrow x' \leq x$

because:
- nothing else says for $\bot$
- by induction $x \leq fx$ on U
- $x_i \leq \bigvee_{i \in I} x_i$

② OBSERVE WLOG $\prec$ IS "TRANSITIVE"

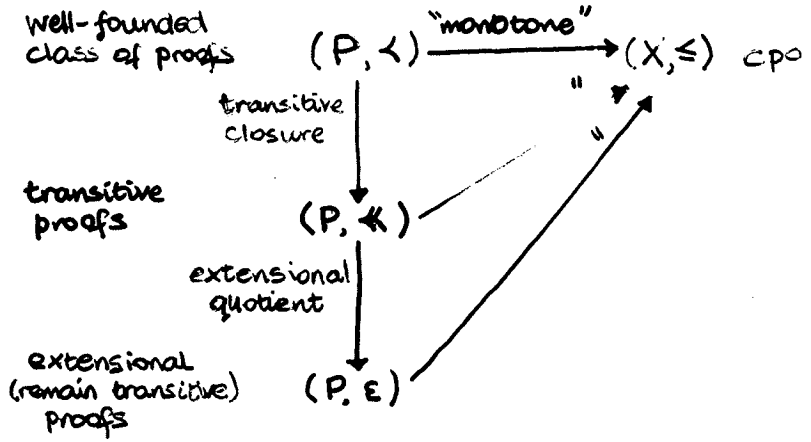
$$\frac{\cdots \ \dfrac{y \in U}{x_1 = fy \in U} \quad \dfrac{z_j \in U \ (j \in J)}{x_2 = \bigvee z_j \in U} \cdots}{\bigvee x_i \in U} \rightsquigarrow \frac{y \in U}{\cdots \ y \in U \ x_1 = fy \in U \cdots z_j \in U \ (j \in J) \cdots}{\bigvee \{x_i, y, z_j\} \in U}$$

INDEED $\dfrac{x_i \in U \ (i \in I)}{\bigvee fx_i \in U}$

③ OBSERVE WLOG $\prec$ IS "EXTENSIONAL"

$$\frac{\cdots \ x_1 = \bigvee_{y_j \in U} \ x_2 = \bigvee_{y_j \in U} \cdots}{\bigvee x_i \in U} \rightsquigarrow \frac{\cdots \ x_{(i)} = \bigvee x_j \in U \cdots}{\bigvee x_{(i)} \in U}$$

GENERAL PROCESS



must show:

1) transitive closure remains well-founded
2) how to construct extensional quotient
3) it remains (transitive &) well-founded.
4) directedness?

ORDINALE - FIRST DEFINITION.

Classically: Well-founded, Extensional, Transitive $\Leftrightarrow$ Well-founded, Trichotomous

$x<y \vee x=y \vee y<x$

[The proof of this equivalence uses double negation at every step. By far the **worst** (in this sense) result I know.]

So,

if $u \in \mathcal{U}$
this has a canonical proof (extensional + transitive) which is simply the ordinal

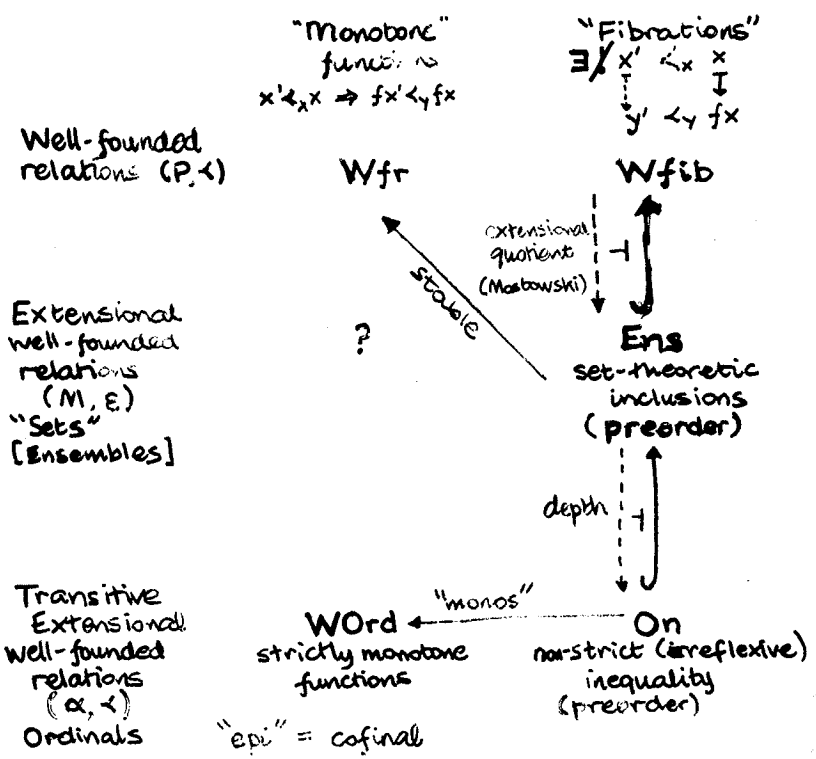
ie $P \vdash u \in \mathcal{U}$ (P extensional + transitive) $\Leftrightarrow$ $P = \alpha \in On$, $u = f^{\alpha}(\bot)$.

Since (classical) ordinals are trichotomous (both in themselves and collectively) they're directed so $\mathcal{U}$ has a greatest element as required.

- use of excluded middle (trichotomy)?
- size (when to stop iterating)?

SETS & ORDINALS — CATEGORICALLY.

Five interesting categories



Mostowski construction of extensional quotient:

$$\overline{P} = \{ \overline{P'} : P' < P \}$$

uses Axiom of Replacement

My construction (quotient by equivalence relation) doesn't.

PRESERVATION OF WELL-FOUNDEDNESS, ETC

Backward

$X \xrightarrow{\text{monotone}} Y$, well-founded

$\Rightarrow X$ well-founded

This is well-known, but people don't know they know it.

Forward

$X \xrightarrow{\text{monotone, surjective}} Y$, well-founded & surjective on predecessors

$\Rightarrow Y$ well founded

Also: transitive closure

$X \xrightarrow{\text{surj. on pred.}} Y$, extensional, well founded

$\Rightarrow$ liftings unique

$X \underset{g}{\overset{f}{\xrightarrow{\text{surj. on pred.}}}} Y$, extensional & well founded

$\Rightarrow f = g$

Corollary: ensembles have no nontrivial automorphisms.

Prop^n If X, Y are extensional & well-founded then there is at most one function $X \to Y$ which is surjective on predecessors, and then it's an inclusion of lower sets **characterise subset relation**

Extensional quotient of $(X, <)$ is $X/\sim$ where

$x \sim y$ if $(\forall x' <_X \exists y' <_Y y \sim y') \wedge (\forall y' <_Y \exists x' <_X x \sim y')$

Egli Milner

$X \longrightarrow X/\sim$ is universal surjective-on-predecessors f^n to an extensional well-founded rel^n

APPLICATIONS OF EXTENSIONAL QUOTIENT

Powerset given (X, ε)

form $X \mapsto P(X)$

with $x \prec y$ if $x \varepsilon y$

$x \prec U$ if $x \in U$

$[U \prec V$ never$]$

extensional quotient is $(P(X), \varepsilon)$

$x = \{x' : x' \varepsilon x\}$

von Neumann hierarchy

Indexed union given (X_i, ε_i) $(i \in I)$

form $\coprod_{i \in I} X_i$ with relation on each component

extensional quotient is $(\bigcup_{i \in I} X_i, \varepsilon)$

Later: **intuitionistic successor**:

$\alpha^+ \subset P(\alpha)$ = set of [directed] lower sets of α

hereditary join: $\alpha \vee \beta = \alpha \cup \beta \cup \{\alpha' \vee \beta' : \alpha' \in \alpha, \beta' \in \beta\}$

is extensional quotient of

$\alpha + \beta + \alpha \times \beta$

with $\alpha'' \prec \langle \alpha', \beta' \rangle$ if $\alpha'' \prec \alpha' \in \alpha$

$\beta'' \prec \langle \alpha', \beta' \rangle$ if $\beta'' \prec \beta' \in \beta$

$\alpha'' \prec \alpha'$ $\beta'' \prec \beta'$

SUCCESSOR ORDINALS

Classically, $\alpha^+ = \alpha \cup \{\alpha\}$

α^+ is the set of decidable lower subsets

Idea from synthetic domain theory:

$\alpha^+ = \sum^{\alpha}$ = set of all upper subsets

Definition for intuitionistic ordinals:

α^+ = set of all lower subsets

Hence:

$$\beta \in \alpha^+ \iff \beta \subseteq \alpha$$

But

$$\alpha \in \gamma \not\Rightarrow \alpha^+ \subseteq \gamma$$

because although $\beta \in \alpha \Rightarrow \beta \in \gamma$

$\beta \subseteq \alpha \not\Rightarrow \beta \in \gamma$

So the irreflexive relation is **not** obtained [in any simple way] from the **reflexive** one.

INDUCTION & DIRECTEDNESS

$0 = \emptyset$

$1 = \{\emptyset\} = P(\emptyset)$

$2 = P(\{\emptyset\}) = \Omega$

new symbol: $\mathbb{2}$ two-dum-omega

$3 =$ lower subsets of $\mathbb{2}$

$\alpha \in 3 \Leftrightarrow \begin{cases} \alpha \subset \mathbb{2} & \text{such that} \\ \forall p \in \mathbb{2}.\ p \in \alpha \to \bot \in \alpha \end{cases}$

such α need not be directed

so in $f^{\alpha}(\bot) \stackrel{\text{def}}{=} \bigvee \{ f(f^{\beta}(\bot)) : \beta \in \alpha \}$

the join is not directed

so cannot be formed in a cpo

FIRST DEFINITION OF ORDINALS DOESN'T WORK

How do we define (hereditarily) directed ordinals?

[B.T.W.: "directed" means in the binary sense

$\forall u,v\ \exists w.\ u,v \leq w$

only, so $\emptyset$ is directed. Better: "semidirected"]

$\alpha^+ =$ set of all lower directed subsets of α

[in SDT any $\alpha \to \Sigma$ automatically preserves $\wedge, \vee$]

HEREDITARILY DIRECTED ORDINALS

$(\alpha, <)$ set with binary relation such that

well founded $\forall \varphi.\ \dfrac{\forall x.[\forall x'.\ x'<x \to \varphi(x')] \to \varphi(x)}{\forall x.\ \varphi(x)}$

extensional $\forall x,y\ \dfrac{\forall z.\ z<x \leftrightarrow z<y}{x=y}$

(hereditarily) transitive $\forall x,y,z\ \dfrac{x<y \qquad y<z}{x<z}$

hereditarily directed $\forall x,y.\ \exists z.$

$\forall x'.\ \dfrac{x'<x}{x'<z}$

$\forall y'.\ \dfrac{y'<y}{y'<z}$

$\forall z'.\ \dfrac{x<z' \qquad y<z'}{z<z'}$

0, $\vee$ are constructed in the obvious way

α^+ is the set of $\subseteq$-directed lower sets of α

$\alpha \vee \beta$ is the hereditary join

WHEN TO STOP ITERATING?

Classically: Hartogs' Lemma

for any X,

$\mathcal{H}(X)$ is the ordinal of isomorphism classes of ordinal structures on subsets (subquotients) of X ($\mathbb{N} \times X^{\mathbb{N}}$)

Since no two lower subsets of an ensemble are isomorphic unless by the identity

$$\mathcal{H}(X) \longrightarrow X \quad \text{eg by} \quad \alpha \mapsto f^\alpha(\bot)$$

is not injective

Therefore with $x = f^{\mathcal{H}(X)}(\bot)$

$\neg\neg\ x = f(x)$

How do we make them genuinely equal?

RETURN TO THE COMPLETE LATTICE CASE

Suppose $\bar{X}$ has binary joins and $\bar{f}: \bar{X} \to \bar{X}$ preserves them

How to achieve this special case from the general one?

FREELY.

$$\bar{X} = \{ W \subset X : \ \bot \in W \ ;\ x \leq y \in W \to x \in W \ ;\ x_i \in W \to \bigvee x_i \in W \}$$

bottom, down-closed, Scott-closed

under inclusion

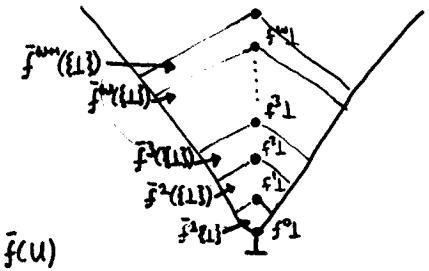
$$\begin{array}{ccc} \bar{X} & \xrightarrow{\bar{f}} & \bar{X} \\ {\scriptstyle x \mapsto \downarrow x}\uparrow & & \uparrow \\ X & \xrightarrow{f} & X \end{array}$$

$\bar{f}$ preserves $\vee$ and hence all joins

What about ordinal powers?

$$\bar{f}^\alpha(\{\bot\}) = \downarrow f^\alpha(\bot)$$

The iterates in $\bar{X}$ are principal lower sets generated by $f^\alpha(\bot)$



If $U = \bar{f}^\alpha(\{\bot\})$ is fixed $U = \bar{f}(U)$

then it has a greatest element, which is fixed by f.

THE JOIN-PRESERVING CASE :

APPLYING HARTOGS' LEMMA **POSITIVELY**

Classically, ordinals are trichotomous so the join $\alpha \vee \beta = \max\{\alpha, \beta\}$ is preserved automatically

$$\alpha \longmapsto f^{\alpha}(\bot)$$

INTUITIONISTICALLY the join is nontrivial

but $f^{\alpha\vee\beta}(\bot) = f^{\alpha}(\bot) \vee f^{\beta}(\bot)$

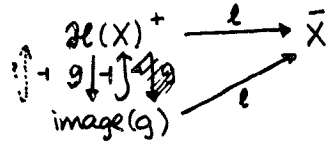
if $f : X \to X$ itself preserves binary joins

So we have $\mathcal{H}(X)^+ \underset{r}{\overset{l}{\rightleftarrows}} \bar{X}$ $\quad l(\alpha) = \bar{f}^{\alpha}(\{\bot\})$

This induces a closure operation on $\mathcal{H}(X)^+$

$$g(\alpha) = \text{largest } \beta \text{ with } l(\beta) \leq l(\alpha)$$

This commutes with l



The inherited order makes image(g) an ordinal, α_0

The restriction of l to α_0 satisfies the same defining equation as l on $\mathcal{H}(X)^+$ but is injective. so $\alpha_0 \in \mathcal{H}(X)$ and $l(\alpha_0^+) = l(g(\alpha_0^+)) = l(\alpha_0)$ so $l(\alpha_0)$ is the fixed point.

Postscript to "Tarski's theorem for cpos, constructively"

I am grateful to Thierry Coquand, Peter Freyd & André Joyal Paul Taylor. for their interesting and helpful comments.

1. Peter and André observed that $(On, \in)$ is the free (class) cpo equipped with successor satisfying $0 < 0^+$ (although it is not monotone: $\alpha < \beta \not\Rightarrow \alpha^+ < \beta^+$).

2. Thierry observed that the proof can be done in the Hoare powerdomain, without using ordinals at all.

   Put $L(X) = \{ a \subset X : \bot \in a,\ a \text{ is } \downarrow, \vee \text{ (ie Scott)-closed} \}$. this is the free complete $\vee$-semilattice on X qua cpo (ie $L \dashv U$ where $U : \mathbf{CSLat} \to \mathbf{CPO}_\bot$). NB $\bot_{L(X)} = \{\bot_X\}$

   Let $V \subset L(X)$ be the smallest subset such that $\bot_{L(X)} \in V$, V is closed under $\bar{f} : L(X) \to L(X)$ where $\bar{f}(a) = \overline{f[a]}$ (Scott-closure) and V is closed under $\vee$ (but not $\downarrow$)

   Then (1) $\forall a \in V.\ \exists x.\ a = \downarrow x$
   for put $W = \{ a : \exists x.\ a = \downarrow x \}$
   show that $\bot_{L(X)} \in W$, it's closed under $\bar{f}$ and $\vee$
   so $W \supset V$

   (2) $\forall a \in V.\ \forall b \in V.\ \exists c \in V.\ a, b \leq c$
   for put $W = \{ b : \exists c.\ a, b \leq c \in V \}$.
   Show that $\bot_{L(X)} \in W$, it's closed under $\bar{f}$ and $\vee$
   second uses:
   let $a, b \leq c$, then $a \vee \bar{f}b \leq \bar{f}a \vee \bar{f}b = \bar{f}(a \vee b) \leq \bar{f}c$

   Hence V (being directed and closed under $\vee$) has a greatest element $a = \downarrow x$
   whence one can show that $x = fx$ is the least fixed pt.

3. UNFORTUNATELY $\bar{f}$ does not preserve binary join because intuitionistically the union of two (Scott) closed sets need not be closed.
   This problem applies both to Thierry's proof and mine. but there is still a join-preserving function $L(\mathbf{On}) \to L(X)$. I don't know at the moment how to proceed from here.

4. Conjecture: the classical ordinals can be recovered as those which are hereditarily either successors or limits Define $\alpha^- = \{ \gamma : \exists \beta.\ \gamma \in \beta \in \alpha \}$. $\alpha^{+-} = \alpha \in \alpha^+$, $\lambda^- = \lambda$.

ISSN 0105-8517

Proceedings of the

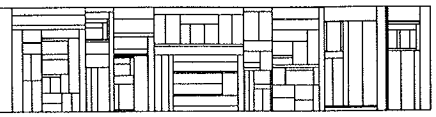
CLICS Workshop – Part II

Aarhus University, 23-27 March 1992

Glynn Winskel, ed.

DAIMI PB – 397-II
May 1992

COMPUTER SCIENCE DEPARTMENT
AARHUS UNIVERSITY
Ny Munkegade, Building 540
DK-8000 Aarhus C, Denmark



CLICS WORKSHOP
Aarhus University, Denmark

PROGRAMME

Monday:

		page
09.00-10.00	J. Y. Girard. On the geometry of interactions[1].	
10.05-10.50	S. Abramsky. New foundations for the geometry of interactions.	1 – 42
	Coffee	
11.20-12.05	P. Wadler. There's no substitute for linear logic.	43 – 70
12.15-13.00	G. Bierman. Type systems, linearity and functional languages.	71 – 92
	Lunch	
14.15-15.00	V. Pratt. Event spaces.	93 – 114
15.05-15.50	C. Barry Jay. Tail Recursion Through Universal Invariants.	115 – 138
	Tea	
16.20-17.05	G. Longo. A new challenge: The categorical semantics of ad hoc polymorphism.	139 – 160
17.15-18.00	R. Tennent. Semantics of local variables.	161 – 174
18.00-18.20	A. Joyal. Conway games.	175 – 182

Tuesday:

09.00-10.00	T. Coquand. Interaction sequences.	183 – 212
10.05-10.50	R. Crole. Recursive types via fixpoint objects.	213 – 224
	Coffee	
11.20-12.05	E. Moggi. Evaluation logic, an overview.	225 – 242
12.15-13.00	P. Dybjer. A construction of a Frege structure in predicative type theory.	243 – 260
	Lunch	
14.15-15.00	S. Vickers. Topical categories of domains.	261 – 274
15.05-15.50	P. T. Johnstone. Variations of the bagdomain theme.	275 – 282
	Tea	
16.20-17.05	T. Streicher. Towards an axiomatization of domain theory in type theory.	283 – 303
17.15-18.00	A. Schalk. Power locales.	304 – 304
18.00-18.45	P. Taylor. Tarski's theorem for cpo's, constructively.	305 – 322

[1]There is no contribution for this talk.

Wednesday:

		page
09.00-10.00	P. Freyd. Towards axiomatic domain theory.	323 – 328
10.05-10.50	W. Phoa. Synthetic domains and operational semantics.	329 – 340
	Coffee	
11.20-12.05	M. Fourman. Set-theoretic semantics for ML.	341 – 358
12.15- 13.00	R. Amadio. On adequacy of per-models.	359 – 380
	Lunch	
	Free afternoon.	

Thursday:

09.00-10.00	M. Felleisen. Observable sequentiality and full abstraction.	381 – 412
10.05-10.50	S. Brookes. Computational comonads and intensional semantics.	413 – 458
	Coffee	
11.20-12.05	A. Pitts. Coinduction for recursively defined domains.	459 – 480
12.15-13.00	P. Scott. Remarks on the Π calculus and linear logic.	481 – 514
	Lunch	
14.15-15.00	Y. Lafont. Genetic abstract machine.	515 - 532
15.05-15.50	E. Ritter. Categorical abstract machines for higher order typed λ-calculi.	533 – 545
	Tea	
16.20-17.05	B. Jacobs. Affine and material semantics.	546 – 546
17.15-18.00	V. de Paiva. Full intuitionistic linear logic.	547 – 570

Friday:

09.00-10.00	G. Plotkin. On compactness and CPO-enriched categories.	571 – 584
10.05-10.50	E. Robinson. A categorical account of parametricity and type abstraction.	585 – 598
	Coffee	
11.20-12.05	P. Francois Lamarche. Sequentiality, Games and Linear Logic.	599 – 608
12.15-13.00	J. Power. A logical framework based on categories with structure.	609 – 612

List of Participants 613 – 616

Notes by Philip Scott, March 25, 1992

Towards Axiomatic Domain Theory
P. Freyd

Algebraically Complete Category:
Every endofunctn (covariant) has initial algebra

This is inconsis w/usual completeness (i.e. e.g.'s = complete lattices are only things complete in both senses)

E.G's of AC cats:
- Countable sets
(1) · $\aleph_0$-dim. spaces.

In (1), Every T preserves monos.
$\exists$ embedding $TA \rightarrowtail A$ (take vs of max. dimension).

$\therefore$ get $Sub(A) \longrightarrow Sub(TA) \to Sub(A)$
Can use Tarski argument to

get least fixed pt. Then get F s.t. $TF \cong F$.

(above arg. still true in dual cat — although arg. false)

Consider
$$\begin{array}{ccc} TF & \xrightarrow{T\hat{b}} & TB \\ \downarrow & & \downarrow b \\ F & \xrightarrow{\hat{b}} & B \end{array}$$

Existence is hard part: Consider

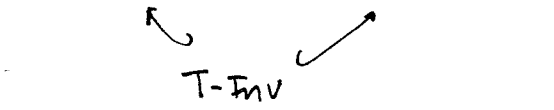
$$Par(F,B) \longrightarrow Par(TF,TB) \longrightarrow Par(F,B).$$

This gives a map $F \to B$.

Why false in dual category?
E.g. $TX = A \times X$.
IF alg. exists, must be A^N.

Dual cat. is reg. cat —
but $\sqrt{}$ partial maps not necessary
partial map. ∴ doesn't work.

Algebraically Compact Categories



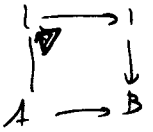
Unique map: Initial T-alg
→ Final T-coalg.
are canonically isomorphic.

Alg. compact categories are punctuated
(≈ ∴ like domains w/ strict maps)

$\mathbb{B} \hookrightarrow \mathbb{A}$ (= alg. compact)

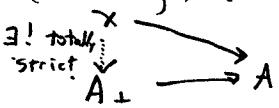
$\mathbb{B}$ subcat ⫰ totally strict maps

$A \longrightarrow B$ totally strict if

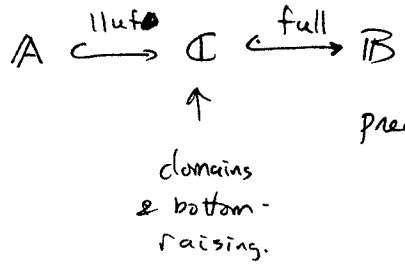


i.e. $X \longrightarrow A \longrightarrow B = \bot \Rightarrow X \longrightarrow A = \bot$.

Axiom: $\mathbb{B}$ coreflective in $\mathbb{A}$.
(usually, just lifting).



Can interpret $\mathbb{B}$ as "predomains"
(ignore totally strict maps' behaviors at bottom)



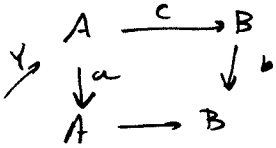
Thm: A lluf reflective subcat
$\mathbb{A} \hookrightarrow \mathbb{C} \Rightarrow \mathbb{C}$ has
fixed pt. ppty.

(Trivially $\mathbb{A}$ has fixed pt ppty)
In fact, ∃ clinat

$Y_A : (A, A) \longrightarrow (1, A)$.

IF $\mathbb{C}$ were cartesian closed, could
internalize : $Y_1 : A^1 \longrightarrow A$

satisfies Plotkin ppty:

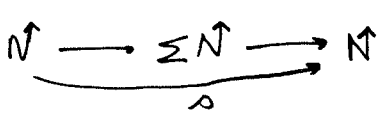


$Y(a)c = Y(\cdot$

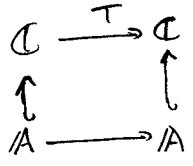
Define $\mathbb{N}^\uparrow$ — ascending nats

$\Sigma X = X_\bot$. ∃ map $\Sigma \mathbb{N}^\uparrow \longrightarrow \mathbb{N}^\uparrow$

By using reflector ppty:



IF have endofunctr. $T : \mathbb{C} \longrightarrow \mathbb{C}$,
~~then~~ induced by $\mathbb{A} \xrightarrow{T} \mathbb{A}$,



Final T-coalg (wrt A) is still final wrt $\mathbb{C}$.

IF $T(\mathbb{C}) \subset \mathbb{A}$, the initial T-alg (wrt $\mathbb{A}$) is still so wrt $\mathbb{C}$.

E.g. $\mathbb{A} = Rel(\text{Ctbl. Sets})$ — is alg. compact wrt functors that preserve ordering of rel's.

(∵ such a functr preserves converses $T(R^\circ) = T(R)^\circ$ ∴ T preserves partial maps & maps. ∴ these are invariant subcats).

∴ Initial alg for T is initial

w.r.t. maps.

What is $\mathbb{B}$ — totally strict in this case?

$\mathbb{B}$ = "luf" subcat of entire rel's

Reflection: $1+A \longrightarrow A$, $B \longrightarrow A$

Not all idempotents split in $\mathbb{B}$.

But take Karoubi($\mathbb{B}$):

$\exists$ max. $1+A \longrightarrow A$ s.t.

$1+A \longrightarrow A$, $1+A$ — this yields coreflection.

$\mathbb{B} = Rel(Sets_{fin})$

Karoubi($\mathbb{B}$) = join-preserving maps on finite distrib. lattices

Rel(all sets): cat of V pres. maps on completely dist. lattice

$$- \bigwedge_I \bigvee_J a_{ij} = \bigvee_{f \in \prod_I J} \bigwedge_I a_{i,f(i)}$$

What are idempotents in Rel?

$R^2 \subset R$; $R \subset R^2$

Rel(all sets) is alg. compact!

(Q: What is $\mathbb{Z}$ — qua T-alg?)

Thm: $\mathbb{A}$ alg. compact, $T : \mathbb{A}^\circ \times \mathbb{A} \longrightarrow \mathbb{A}$ has a canonically least inv. object:

Construe T as endofunctr $\mathbb{A}^\circ \times \mathbb{A}$

$\mathbb{A}^\circ \times \mathbb{A} \longrightarrow \mathbb{A}^\circ \times \mathbb{A}$

$(A, B) \longmapsto (TBA, TAB)$ } & find inv. objects.

Q: Is $\mathbb{R}$ a T-alg?

Consider Rel(Sets)

$TX = X \triangleleft 1 \triangleleft X$ gives $\mathbb{Q}$.

Initial T-alg. $[-\infty, \infty]$

Aside: $TX = 1 + X^N$ on Sets.

Initial T-alg = Trees w/ctbl branching (rooted) & well frded.

e.g. Ω = unctbl ordinals

$1 + \Omega^N$

$\downarrow$ $\langle \perp, \sup(\text{successors}) \rangle$

Ω

e.g. $1 + (N^N)^N \longrightarrow N^N$

$\cong$

$1 + N^{N \times N}$

Set-Theoretic Semantics for ML.

Wesley Phoa
John Longley
Michael Fourman

LFCS
Edinburgh

Work in Progress . . .

Semantics - Categories

Programs - Computer Science

Proof - Logic

NO NEW THEOREMS

2

Semantics

- Intended Semantics
- Mathematical Semantics
- Operational Semantics

Example A simple functional language L

Types A, B: unit, bool, products $A \times B$, Algebraic datatypes A list, B tree

Functions f, g: recursive functions $f : A \to B$ (à la ML)

Expressions e, b:
- Application: $f\ e$
- Conditional: if b then e_1 else e_2
- Case: case e of $pat_1 \Rightarrow e_1$ | $pat_2 \Rightarrow e_2$ | ...

Intended Semantics

$[\![A]\!]$ is a set

$[\![A\text{-list}]\!]$ is a free algebra

$[\![f : A \to B]\!]$ is a partial map $[\![A]\!] \to [\![B]\!]$

$[\![e : A]\!]$ is an element of $\tilde{[\![A]\!]}$

Recursive functions are computed as least fixed points, using restriction order $\leq$ on partial maps.

$$f = \bigvee_n F^n(\perp)$$

A Program Breadth-first Search

4

```
datatype Tree = L of int
              | N of int * Tree * Tree

fun search (forest : Tree list) =
case forest of
    []          => []
| (L i)::t      => i :: search t
| (N(i,l,r))::t => i :: search (t @ [l,r])

fun BFS t = search [t]
```

Naive set-theoretic reasoning in predicate logic allows us to prove properties

Eg "By induction on the structure of t
BFS t terminates."

A real functional language

ML - functional subset
(no refs or exceptions)

- first-class higher-order values
- recursive datatypes
- polymorphism (Hindley-Milner)
- modules

Claim
A set-theoretic interpretation of ML types and values is simple and consistent.

In place of Sets use Eff, the effective topos

$[\![A]\!]$	is a set (which is a predomain)
$[\![D]\!]$	is a solution to domain equation
$[\![f: A \to B]\!]$	is a partial map
$[\![A \to B]\!]$	is the set of (Σ) partial maps

Recursive functions are computed as least fixed points, using (intrinsic) predomain ordering. $\sqsubseteq$ $\quad f = \bigsqcup_n F^n(\bot)$

Logic

Higher-order Intuitionistic Logic

Equality Types are Decidable

Products are sets of pairs

Function spaces are extensional

Datatypes solve domain equations

- Algebraic Types are initial (induction)

Partial functions satisfy
defining (Kleene) equalities

- Functions with $\subseteq$ monotone definitions are initial.

NB Some logical types — eg Ω
are not computational.

"ADDED IN PROOF"
We can reason about "lazy lists" (ie streams) in this set-theoretic setting, without considering the domain ordering.

Another Program! BFS à la Tofte

```
datatype  Cont = C of Cont -> int list
                | D

fun apply (Cf) c = f c
|   apply  D  (Cf) = f D
|   apply  D   D  = []

fun search t a b =
case t of
     Li => i :: apply a b
|  N(i,l,r) =>
   i :: apply a (C (apply b o Cosearch l o Cosearch r))

fun BFS_c t = search t D D
```

Claim For all trees t

$$BFS_c\ t = BFS\ t$$

Define cont : Forest -> Cont -> Cont

```
fun cont [] = I
|   cont (t::rest) = C o search t o cont rest

fun contleft t = cont t D
fun contright [] = D
|   contright t = C (apply D o cont t)
```

lemma (Longley)

apply (contleft t1) (contright t2) = BFS (t1@t2)

Proof By induction

apply (contleft []) (contright []) = ... = []

apply (contleft []) (contright t2) = ...
= apply D (cont t2 D)
= apply (contleft t2) D

apply (cont (Li::rest) D) (contright t2) = ...
= i :: apply (contleft rest) (contright t2)

apply (cont (N(i,l,r)::rest) D) (contright t2)
= search (N(i,l,r)) (cont rest D) (contright t2)
= i :: apply (cont rest D) (C (apply (contright t2) o C o search l o C o search r) []) (by cases)
= i :: apply (contleft rest) (C (apply D o cont (t2@[l,r])))
= i :: apply (contleft rest) (contright (t2@[l,r]))

The cases

C (apply (contright t2) o C o ...)

t2 = [] C (apply D o C o ...)

t2 ≠ [] C (apply (C (apply D o cont t2)) o C o ...)
= C (apply D o cont t2 o C o ...)

Since apply (C cf) = cf

by extensionality

10

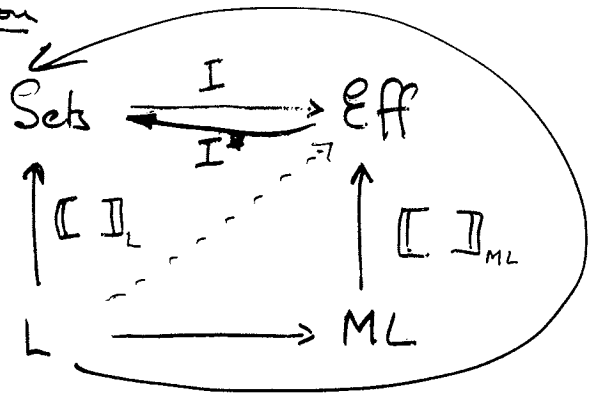
Lemma For algebraic datatypes solving domain equations gives free algebras For arbitrary objects of generators.

Lemma If $f \leq g \Rightarrow Ff \leq Fg$

then $\bigvee_n F^n(\bot) = \bigsqcup_n F^n(\bot)$.

A sufficient condition is that each occurrence of f is applied, on the RHS of its definition

Observation



The mathematical semantics for ML is a faithful extension of the intended semantics for L. $[\![e]\!]_L = I^* [\![e]\!]_{ML}$

We can extend this picture to cover Polymorphism and modules

```
fun Ix = x
    I : 'a --> 'a
    [] : 'a list
    rev : 'a list --> 'a list
```

rev = I : 'a list → 'a list
?

Not in our semantics !

We interpret 'a by extending $\mathcal{E}$ with a generic (computational) type

$\mathcal{E}['a]$

is a topos with predomains as before.

Type specialization is represented by logical inverse image functors. Eg

'a list ↦ int list

'a list $\in \mathcal{E}['a] \xrightarrow{\text{int}} \mathcal{E}$

'a ↦ int

A Simple Categorical Semantics for ML Polymorphism & Modules

Wesley Phoa, Mike Fourman
LFCS/Dept of CS, Edinburgh

I Polynomial Categories

II Let-Polymorphism

III Signatures, Sharing

"avoid mathematics;
use category theory instead"

WHAT DOES $\lambda x.x : \forall\alpha.\alpha\to\alpha$ MEAN, IN ML

(a) Model all types in a category $\mathbb{C}$,

so $1 \xrightarrow{[\![\lambda x.x]\!]} [\![\forall\alpha.\alpha\to\alpha]\!] \in \mathbb{C}$

$\mathbb{C}$ must be a "model of system F"

(b) (Fourman / Phoa / Walters / Vickers / ...)

A type scheme $\forall\alpha.\alpha\to\alpha$ does not have the same status as, e.g., $nat\to nat$,

so $1 \xrightarrow{[\![\lambda x.x]\!]} [\![\alpha\to\alpha]\!] = [X\to X] \in \mathbb{C}[X]$

$\mathbb{C}[X]$ = POLYNOMIAL CAT on $\mathbb{C}$

= $\mathbb{C}$ + "indeterminate object" X

$[\![\alpha]\!] = X$

OBSERVATIONS

- "$\forall\alpha$" not interpreted — just a marker that tells you when you can instantiate
- does not extend to full System F
- change of viewpoint:

  "polymorphism is not a feature of a particular model $\mathbb{C}$, but added on top of an arbitrary $\mathbb{C}$ via a general (2-) categorical construction"

GENERAL

works for any CCC, any distrib. cat.....

(& models of System F "restrict")

"ORTHOGONAL"

polymorphism separate from underlying lang.

I POLYNOMIAL CATEGORIES

CCC = (2-) category of CCC's

If $\mathbb{C}$ is a CCC, its polynomial cat (rel. to CCC) is a category

$$\mathbb{C} \xrightarrow{I} \mathbb{C}[X] \ni X \quad \text{"generic object (of } \mathbb{C}\text{)"}$$

s.t. for any CCC $\mathbb{D}$,

$$\frac{\mathbb{C} \longrightarrow \mathbb{D}\ ,\quad D \in \mathbb{D}}{\mathbb{C}[X] \to \mathbb{D} \qquad X \mapsto D \ (\text{up to} \cong)}$$

e.g. if $D \in \mathbb{C}$, $\exists!$ (up to $\cong$)

substitution functor

$$S_D : \mathbb{C}[X] \to \mathbb{C} \qquad S_D I \cong 1_{\mathbb{C}}$$
$$S_D X \cong C$$

CONSTRUCTION

$$\mathbb{C}[X] = \mathbb{C} + \langle X \rangle$$

where

"+" = (bi) coproduct in CCC

"$\langle X \rangle$" = free CCC on one object

These exist by general theory of cats-with-structure (Blackwell, Kelly, Power)

OR Given

$$F : \mathbb{B}^{op} \to \mathbf{CCC}$$

indexed CCC with generic object $G \in F(P)$

can, in $\mathrm{Im}(F)$, put

$$\mathbb{C} = F(1)$$

$$\mathbb{C}[X] = F(P) \qquad X = G$$

II LET-POLYMORPHISM

Common knowledge: can interpret Hindley-style typing $(\lambda x.x : \alpha \to \alpha)$ in any CCC $\mathbb{C}$.

Not clear how to extend this to Milner-style LET-polymorphism

let $x = \lambda z.z$ in $(xx)y$

Polynomial cats provide simple sol'n BUT a little cumbersome: if $x \in FV(e)$ might end up bound in a let, must keep track of each occurrence of x.

TYPE SYSTEM Damas-Milner → Tofte

$e ::= x \mid ee' \mid \lambda x.e \mid$ let $x = e$ in e' [...]

$\tau ::= \alpha \mid \tau \to \tau'$ [| int | bool | ...]

$\sigma ::= \forall \alpha_1 \ldots \alpha_n . \tau$

$\sigma > \tau'$ iff $\sigma = \forall \alpha_1, \ldots \alpha_n . \tau$ and $\exists$ types $\tau_1, \ldots \tau_n$ s.t. $\tau' = \tau[\alpha_i := \tau_i]$

If F is a set of type vars,

$Clos_F \tau = \forall \alpha_1 \ldots \alpha_k . \tau$

where $\{\alpha_1 \ldots \alpha_k\} = TV(\tau) \setminus F$

(There are variants, but this version is quite close to SML)

Typing judgements look like (Wes)

$$T; G; P \vdash e : \tau$$

where $T = \{\alpha_1, \ldots, \alpha_n\}$

$G = \{x_1 : \sigma_1, \ldots, x_p : \sigma_p\}$ for lets

$P = \{y_1 : \tau_1, \ldots, y_q : \tau_q\}$ for λ's

RULES

$$\frac{}{T; G; P, y:\tau \vdash y : \tau} \qquad \frac{\sigma \succ \tau}{T; G, x:\sigma; P \vdash x : \tau}$$

$$\frac{T; G; P \vdash e : \tau \to \tau' \quad T; G; P \vdash e' : \tau}{T; G; P \vdash e e' : \tau'}$$

$$\frac{T; G; P, y:\tau \vdash e : \tau'}{T; G; P \vdash \lambda y.e : \tau \to \tau'}$$

$$\frac{T; G; P \vdash e : \tau \quad T; G, x : Clos_{FTV(G;P)}\tau; P \vdash e' : \tau'}{T; G; P \vdash \text{let } x = e \text{ in } e' : \tau'}$$

SEMANTICS

Sps $\alpha_1, \ldots, \alpha_n; \{x_i : \sigma_i\}; \{y_j : \tau_j\} \vdash e : \tau$

$[\![e]\!] : ? \longrightarrow [\![\tau]\!]$ in $\mathbb{C}[X_1, \ldots, X_n]$

Let $x_i : \tau_{i1}, \ldots, x_i : \tau_{ir_i}$ $(\sigma_i \succ \tau_{ie})$ be the occurrences of x_i in e

$$[\![e]\!] : \prod_i \prod_{e=1}^{r_i} [\![\tau_{ie}]\!] \times \prod_j [\![\tau_j]\!] \longrightarrow [\![\tau]\!]$$

let handled using substitution functors

SOUNDNESS (special case)

If $T; ; \vdash e : \tau$ and $e \twoheadrightarrow e'$, e' in nf

then $T; ; \vdash e' : \tau$ and

$[\![e]\!] = [\![e']\!] : 1 \longrightarrow [\![\tau]\!]$

EXAMPLE

$\alpha_1, \alpha_2;\ x : \forall\alpha_2.\alpha_2 \to \alpha_2;\ y : \alpha_1 \vdash (xx)y : \alpha_1$

$[\![(xx)y]\!] : [[X_1 \to X_1] \to [X_1 \to X_1]]$

$\times [X_1 \to X_1] \times X_1 \longrightarrow X_1$ (evals)

$\alpha_1, \alpha_2;\ ;\ \vdash \lambda z.z : \alpha_2 \to \alpha_2$

$[\![\lambda z.z]\!] = \ulcorner 1_{X_2} \urcorner : 1 \to [X_2 \to X_2] \in \mathbb{C}[X_1, X_2]$

$\alpha_1, \alpha_2;\ ;\ y : \alpha_2 \vdash \text{let } x = \lambda z.z \text{ in } (xx)y : \alpha_1$

$\dashv\vdash \forall\alpha_2.\alpha_2 \to \alpha_2 \succ (\alpha_1 \to \alpha_1) \to (\alpha_1 \to \alpha_1)$

$\rightsquigarrow$ subst. functor $S : \mathbb{C}[X_1, X_2] \to \mathbb{C}[X_1, X_2]$

$X_1 \mapsto X_1 \quad X_2 \mapsto [X_2 \to X_2]$

Similarly for $\forall\alpha_2.\alpha_2 \to \alpha_2 \succ \alpha_1 \to \alpha_1$

so $[\![\text{let} \ldots]\!] : X_1 \xrightarrow{\langle \ulcorner 1_{[X_1 \to X_1]} \urcorner, \ulcorner 1_{[X_1]} \urcorner \rangle \times 1_{X_1}}$

$[[X_1 \to X_1] \to [X_1 \to X_1]] \times [X_1 \to X_1] \times X_1 \to X_1$

III MODULES, SIGNATURES, SHARING

As well as adjoining an indeterminate object X to $\mathbb{C}$, we can adjoin an indeterminate element x to an existing object $C \in \mathbb{C}$ (easier!)

$\mathbb{C}[x:C]$ has the universal prop. that

$$\frac{\mathbb{C} \xrightarrow{F} \mathbb{D} \qquad 1 \xrightarrow{d} FC \in \mathbb{D}}{\mathbb{C}[x:C] \longrightarrow \mathbb{D} \qquad (x \mapsto d)}$$

e.g. have $\mathbb{C} \xrightarrow{I} \mathbb{C}[X, x:X] \ni 1 \xrightarrow{x} X$

and functors $S : \mathbb{C}[X, x:X] \to \mathbb{C}$ s.t. $SI \cong 1_{\mathbb{C}}$ correspond to pairs

$\langle C \in \mathbb{C},\ 1 \xrightarrow{c} C \in \mathbb{C} \rangle$

call this a functor *under* $\mathbb{C}$

SIGNATURES W/OUT SUBSTRUCTURES

```
signature SIGA =
  sig
    type t;
    val f: t → nat
  end;
```

"Structures" in $\mathbb{C}$ are pairs

$$\langle C \in \mathbb{C},\ 1 \xrightarrow{g} [C \to N] \in \mathbb{C} \rangle = S$$

Let $[\![SIGA]\!] = \mathbb{C}[X, x: [X \to N]\rangle$

– then

$$\frac{\text{"structures" in } \mathbb{C}}{[\![SIGA]\!] \to \mathbb{C} \quad \text{under } \mathbb{C}}$$

$[\![SIGA]\!]$ "classifies" structures of signature SIGA; S is generic structure

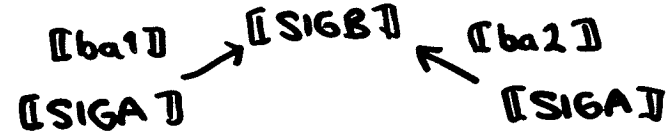
or something...

SIGNATURES W/ONLY SUBSTRUCTURES

```
signature SIGB =
  sig
    structure ba1 : SIGA;
    structure ba2 : SIGA
  end;
```

pictured by ML people as

Form the coproduct



Structures $b: SIGB$ give us

$$[\![b]\!] : [\![SIG\,B]\!] \to \mathbb{C}$$

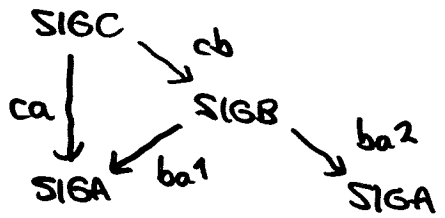
and $[\![b.ba1]\!] = [\![b]\!] \circ [\![ba1]\!]$

$[\![b.ba2]\!] = [\![b]\!] \circ [\![ba2]\!]$

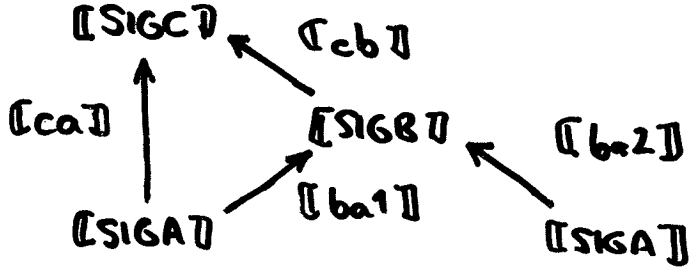
SIGNATURES WITH SHARING

```
signature SIGC =
  sig
    structure ca : SIGA;
    structure cb : SIGB
    sharing   ca = cb.ba1
  end
```

pictured as



To get ⟦SIGC⟧, form the colimit



Commutativity enforces

⟦c.ca⟧ = ⟦c.cb.ba1⟧ ↙ to coherent ≅ on the nose...

FUNCTORS

```
functor F(sb : SIGB) : SIGC =
  struct
    structure ca = sb.ba1;
    structure cb = sb
  end;
```

should give us

⟦F⟧ : ⟦SIGC⟧ ⟶ ⟦SIGB⟧

i.e. if b : SIGB, ⟦b⟧ : ⟦SIGB⟧ → ℂ

then ⟦F(b)⟧ = ⟦b⟧ ∘ ⟦F⟧ : ⟦SIGC⟧ → ℂ

Construct ⟦F⟧ by using the fact that ⟦SIGC⟧ is a colimit.

ALSO POSSIBLE GENERALLY

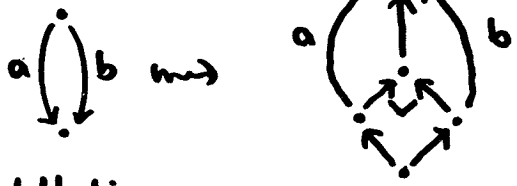
- functors with sharing specs,
- sharing specs involving admissification
- external sharing

etc...

WARNING

Rigorous treatment will involve careful handling of identifier/name bindings

tricky!



-other subtleties...

ALSO POSSIBLE SOMETIMES

- equality types
- type classes (?)
- Extended ML specifications

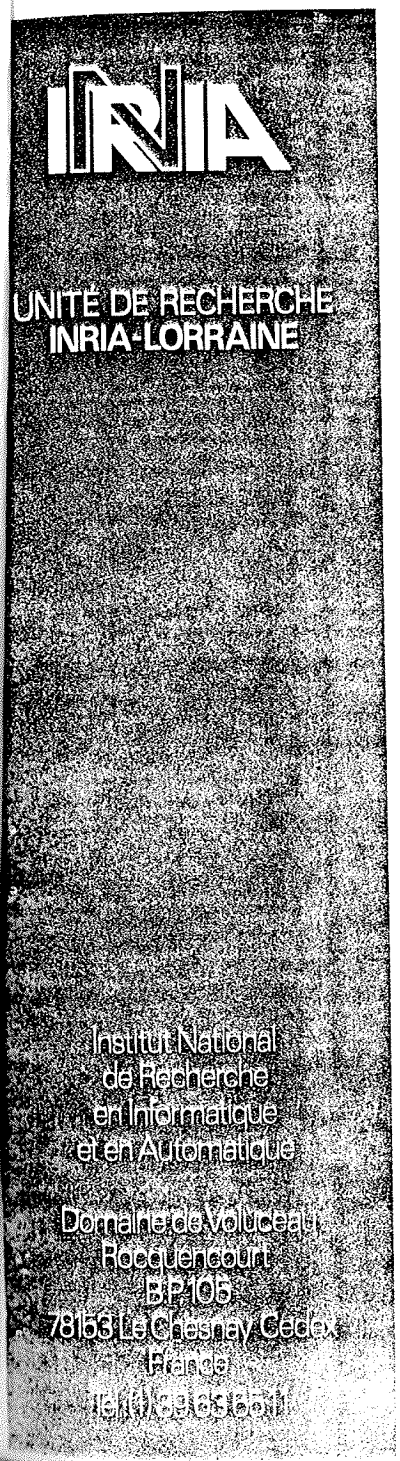
Phoa & Ed Kazmierczak are exploring consequences for EML; has clarified proof rules

MAYBE POSSIBLE

recursive modules

higher-order modules

THE END



Rapports de Recherche

N° 1579

Programme 2
Calcul Symbolique, Programmation
et Génie logiciel

ON THE ADEQUACY OF PER MODELS

Roberto M. AMADIO

Janvier 1992

★RR.1579★

On the Adequacy of Per Models

Roberto M. Amadio
CRIN-CNRS, Nancy[1]

Abstract

In this note we consider a fixed point extension of the second order lambda calculus equipped with a call by value evaluation mechanism. We interpret the language in a partial cartesian closed category of "directed complete" partial equivalence relations (pers) over a domain theoretic model of a type-free, call-by-value, lambda calculus. Our main result is that the notions of "syntactic" and "semantic" convergence coincide.

Résumé

Dans cette note nous considérons une extension de point fixe du lambda calcul du deuxième ordre avec un mécanisme d'évaluation par valeur. Nous interprétons ce calcul dans une catégorie cartésienne partialement fermée de relations partiels d'équivalence. Le résultat principal est l'équivalence des notions syntaxique et sémantique de convergence.

[1] Group "Prograis", URA 262, CRIN-CNRS, B.P. 239, 54506, Vandoeuvre-lès-Nancy, FRANCE. E-mail: amadio@loria.crin.fr. "Prograis" is also a joint project with INRIA-Lorraine.

Introduction

Recently the research in the area of "synthetic domain theory" (see, e.g., Hyland[91]) has especially addressed the problem of discovering subcategories of partial equivalence relations (pers) over a partial combinatory algebra (pca) that enjoy good completeness properties, and that admit certain constructions typical of domain theory, such as the solution of recursive function and domain equations.

Our concern here does not lie in the construction of models, or in the categorical abstraction of such construction, but in an attempt at connecting such models to issues arising in the design and semantics of, say, typed functional languages. In particular we connect a certain per-interpretation to a call-by-value evaluation discipline that corresponds to current implementations of higher order typed functional languages with a static type checker (see, e.g., Cardelli[89]).

The *main result* establishes the equivalence of the syntactic and semantic notions of convergence. This is a classical "adequacy" result for domain theoretic interpretations, as sketched for example in Plotkin[85], after Martin-Löf[83]. However, as far as we know, no results were available, in the case of per-interpretations.

There are two additional points we wish to emphasize:
- In order to prove such an adequacy theorem *very little structure* on the per model is needed, in particular we do not require working with an O-category.
- The adequacy of the per interpretation is largely *independent* from the adequacy of the underlying pca.

Section 5 is the technical core of this paper. The proof of adequacy w.r.t. standard domain theoretic interpretations, requires the combination of "admissible predicates" techniques and "reducibility candidates" techniques. In the proof we propose here, there are two additional twists that are due to the presence of second order types, and to the interplay between the typed and the type-free structures. In particular the key of the result lies in the definition of *adequacy relation* (5.1), and *in the way one associates an adequacy relation to a type* (5.2).

In order to make the central result quickly accessible we have delayed the more or less standard proofs of the first four sections to appendix A. Such sections contain respectively: (1) the definition of a fixed point extension of the second order lambda calculus; (2) the definition of the evaluation mechanism of such language; (3) the description of the basic properties of the partial cartesian closed category of "directed complete" pers over a cpo model of call-by-value, type-free lambda calculus; (4) the interpretation of the language in the semantic structure.

1. Language

Types and raw terms are defined by the following BNFs:

Type Variables:	$tv ::= t \mid s \mid \ldots$
Types:	$\alpha ::= 1 \mid tv \mid (\alpha \to \alpha) \mid (\forall tv.\alpha)$
Term Variables:	$v ::= x \mid y \mid \ldots$
Terms:	$M ::= * \mid v \mid (\lambda v{:}\alpha.M) \mid (MM) \mid (\lambda tv.M) \mid (M\alpha) \mid (Y_\alpha M)$

In the following we will feel free to spare on parentheses, and to omit the type label in the Y combinator. All types and terms are considered up to α-redenomination.

A *well formed context* Γ is given by a list of pairs, v: α, in which all variables are distinct. We write Γ, v: α to evidentiate the last element of the list, we write $v{:}\alpha\in\Gamma$ to state that v:α occurs in Γ, and we denote with ftv(Γ) the collection of type variables free in types occurring in Γ. Note that in the calculus presented here type variables contexts are left implicit.

As usual a *substitution*, say σ, is a function that associates variables, say v, v_1, ..., to formal expressions, say exp, exp_1, ..., . The domain of a substitution is $\{v \mid \sigma(v)\neq v\}$. We assume that such domain is always finite. We denote with $[exp_1/v_1,..., exp_n/v_n]$ a substitution whose domain is contained in $\{v_1,..., v_n\}$, and that associates exp_i to v_i, for i=1,...,n. If σ is a substitution and exp is an expression then σexp denotes the expression resulting from the application of the substitution to the expression, according to the standard rules that take care of bounded variables. We abbreviate an iteration of substitutions, say $\sigma_1(..(\sigma_n\ exp)..)$, with $\sigma_1..\sigma_n$ exp, so, for example, [r/s][s/t] (t→s) = (r→r).

In a formal system the symbol "⇒" separates, in an inference rule, the premisses from the conclusion. If "J" is a judgment of the formal system then we write "⊢J", if such judgment is derivable.

A *typing judgment* is of the shape $\Gamma\supset M{:}\alpha$ where Γ is always a well formed context. Derivable typing judgment are specified by the following formal system:

(*)	$\Rightarrow \Gamma\supset *: 1$
(asmp)	$x{:}\alpha\in\Gamma \Rightarrow \Gamma\supset x{:}\alpha$
(→I)	$\Gamma,x{:}\alpha\supset M{:}\beta \Rightarrow \Gamma\supset(\lambda x{:}\alpha.M){:}(\alpha\to\beta)$
(→E)	$\Gamma\supset M{:}(\alpha\to\beta)\ ,\ \Gamma\supset N{:}\alpha \Rightarrow \Gamma\supset(MN){:}\beta$
(∀I)	$\Gamma\supset M{:}\alpha\ \ t\notin ftv(\Gamma) \Rightarrow \Gamma\supset(\lambda t.M){:}(\forall t.\alpha)$
(∀E)	$\Gamma\supset M{:}(\forall t.\alpha) \Rightarrow \Gamma\supset(M\beta){:}[\beta/t]\alpha$
(Y)	$\Gamma\supset M{:}(1\to\alpha)\to\alpha \Rightarrow \Gamma\supset Y_\alpha M{:}\alpha$

This language is intended to represent a second order lambda calculus with a fixed point combinator over lifted types (one can think of the constant Y_α as having type $((\alpha)_\perp\to(\alpha)_\perp)\to(\alpha)_\perp$, where: $(\alpha)_\perp \equiv (1\to\alpha)$, and $(\alpha\to\beta)\equiv(\alpha\to(\beta)_\perp)$. The type (α→β) should be thought as the type of the *partial* functions from α to β.

2. Evaluation

The *canonical forms* are the well-typed terms without free variables, but possibly with free type variables, that are generated by the following grammar:

C::= * | (λv: α. M) | (λtv. C)

The evaluation "↦" is specified as a relation between terms without free variables and canonical forms. If M↦C then M and C have the same type, so one may also think of "↦" as a family of relations indexed over types. The definition of the evaluation relation proceeds by induction on the structure of a well-typed closed

term.

(*)	$\Rightarrow *\mapsto *$
(asmp)	"we never evaluate a free variable"
(→I)	$\Rightarrow \lambda x{:}\alpha.M\mapsto\lambda x{:}\alpha.M$
(→E)	$M\mapsto\lambda x{:}\alpha.M',\ N\mapsto C',\ [C'/x]M'\mapsto C \Rightarrow MN\mapsto C$
(∀I)	$M\mapsto C \Rightarrow \lambda t.M\mapsto\lambda t.C$
(∀E)	$M\mapsto\lambda t.C \Rightarrow M\alpha\mapsto[\alpha/t]C$
(Y)	$M(\lambda x{:}1.YM)\mapsto C \Rightarrow YM\mapsto C$ (for x fresh variable)

We write $M\downarrow$ if $\vdash M\mapsto C$, for some canonical form C, and $M\uparrow$ otherwise. Note that the definition of "↦" gives directly a deterministic procedure to reduce, if possible, a closed term to a canonical form. Hence each term can reduce at most to a canonical form. Canonical forms always reduce to themselves.

Observe that we *evaluate under type abstraction*. On one hand this corresponds to the fact that in actual implementations of the language the type-checker is static, hence no information about type abstraction and type application appears at run time. On the other hand, as it will become clear in section 4, this choice is important in capturing the behavior of the interpretation of second order types as intersections.

3. Semantic Structure

In the presentation of the per-model we take a minimalist approach, by presenting only those properties and constructions that are needed in the proof. We refer to Amadio[90] for more information about the relevance and the context of the structures discussed below.

Conventions *(category of dcpos)*

A set X is *directed* in the poset (P, ≤) if $\emptyset\neq X\subseteq P$, and $\forall x,y\in X.\exists z\in X.(x\leq z\ \wedge\ y\leq z)$. A poset is *directed complete* (dcpo) if it has joins of its directed subsets. Two mathematical expressions including partial operations, say e_1, e_2, are *Kleene equivalent*, written $e_1\cong e_2$, if either they are both defined and they are equal, or they are both undefined. We also write $e\Downarrow$ ($e\Uparrow$) if a mathematical expression is defined (undefined). A *partial (Scott-)continuous function* between two dcpos, $h{:}(D,\leq_D)\to(E,\leq_E)$, is a partial function between the dcpos D and E such that for any directed set X in D, $f(\bigsqcup X)\cong\bigsqcup f(X)$ (whenever we take the join of an indexed set of mathematical expressions, such join is defined if the join of the *defined* mathematical expressions is defined). We denote with **dcpo** the category of dcpos and partial continuous functions. This category is (equivalent to) a *partial cartesian closed category* in the sense of Moggi[88]. Given two dcpos, D, E, we denote with D→E, the partial exponent, that is the collection of partial continuous functions pointwise ordered.

3.1 Realizability Structure

We assume to have an object D in the category of **dcpo** that has its partial functional space as a retract, i.e. i: (D→D)→D, j: D→(D→D), j∘i = $id_{D\to D}$ in **dcpo**. We define a partial operation of application over D as: de ≜ j(d)(e). From this operation one can define, as usual, continuous operations of pairing, < , >:D×D→D, and projection π_i:D→D, (i=1,2) such that π_1<d,d'> = d, and π_2<d,d'> = d'.

More Conventions *(category of ppers)*

A partial equivalence relation over D (per) is a binary relation over D that is symmetric and transitive. We denote with A, B,... pers over D. We write: dAe for (d,e)∈ A, $[d]_A$ for {e∈ D | dAe}, | A | for {d∈ D | dAd}, [A] for {$[d]_A$ | d∈ | A |}. A partial morphism between pers, f: A→B, is a map f: [A]→[B] such that

$$\exists h\in D.\ \forall d\in |A|.\ (f([d]_A)\Downarrow \wedge hd\in f([d]_A)) \vee (f([d]_A)\Uparrow \wedge hd\Uparrow)).$$

We denote with **pper** the category of pers and partial morphisms of pers. Such category is (equivalent to) a pccc where terminal, product (in the related category of total morphisms), and partial exponent pers are defined as follows:

$$\mathbf{1} \triangleq D\times D, \qquad dA\times Be \Leftrightarrow \pi_1 dA\pi_1 e \wedge \pi_2 dB\pi_2 e,$$
$$h\ pexp(A,B)\ k \Leftrightarrow \forall d,e.\ dAe \Rightarrow (hdBke \vee (hd\Uparrow \wedge ke\Uparrow)).$$

The interpretation of the language is based on the following category of directed complete pers and partial maps. One may think of this category as a loose analogous of the category of dcpos and partial continuous maps. We will see that it retains the basic properties of the category of ppers, and moreover it has a fixed point combinator over "lifted" objects. The proofs of these results follow Amadio[89].

3.2 Definition *(directed complete pers)*

A per A is directed complete (dcper) if for any directed set X in D×D, if X⊆A then $\bigsqcup_{D\times D} X\in A$. We define **dcpper** as the full subcategory of **pper** whose objects are dcpers.

3.3 Proposition *(basic properties of the semantic structures)*

(1) **dcpper** is a pccc.

(2) dcpers are closed under arbitrary intersections.

(3) **dcpper** is reflective in **pper**, that is the inclusion functor from **dcpper** to **pper** has a left adjoint.

(4) **dcpper** has fixpoints over objects of the shape pexp(**1**,A).

4. Interpretation

In this section we define an interpretation of the language based on the semantic structures just introduced. By convention if τ: V→W is a partial function from a collection of variables, v, to a set of values, W, then for v∈V, and a∈W we define: τ[a/v](v') ≅ if v' ≡ v then a else τ(v').

Types. The interpretation of a type, given a type assignment η: tv→dcper, is a dcper, defined by induction as follows:

$$[\![1]\!]\eta = \mathbf{1}$$
$$[\![t]\!]\eta = \eta(t)$$
$$[\![\alpha\to\beta]\!]\eta = pexp([\![\alpha]\!]\eta, [\![\beta]\!]\eta)$$
$$[\![\forall t.\alpha]\!]\eta = \bigcap_{A\ dcper} [\![\alpha]\!]\eta[A/t]$$

Terms. An assignment is a partial function ρ: v→$\bigcup_{A\ dcper}$[A]. A type assignment η is compatible with an assignment ρ, w.r.t. a well-formed context Γ, if for any x:α∈Γ, ([[α]]η = ∅ ⇒ ρ(x)⇑) ∧ ([[α]]η ≠ ∅ ⇒ ρ(x)∈[[α]]η]); we shortly write this as $\eta\uparrow_\Gamma\rho$. The interpretation of a judgment ⊢Γ ⊃ M: α, given η, ρ, such that $\eta\uparrow_\Gamma\rho$, is either undefined or an element in [[α]]η] (equivalently it is a partial map from the terminal object to [[α]]η). Such interpretation is defined by induction on the length of the typing judgment. Observe that some clause may fail to be defined, hence the use of Kleene equivalence. Suppose ⊢∅ ⊃ M: α, we write M⇓ if for any type assignment η, and any ρ, [[∅⊃M:α]]ηρ ⇓, and M⇑ otherwise.

(*) $[\![\Gamma \supset *: \mathbf{1}]\!]\eta\rho = [d]_1$, for d∈D .

(asmp) $[\![\Gamma \supset x_i: \alpha_i]\!]\eta\rho \cong \rho(x_i)$

(→I) $[\![\Gamma \supset \lambda x{:}\alpha.M{:}\alpha\to\beta]\!]\eta\rho = \{h\in D \mid \forall d\in |A|.\ (f(d)\Downarrow \Rightarrow hd\in f(d)) \wedge (f(d)\Uparrow \Rightarrow hd\Uparrow)\}$
where: A = [[α]]η , f(d) ≅ [[Γ,x:α ⊃ M:β]]ηρ[$[d]_A$/x] .

(→E) $[\![\Gamma\supset MN:\beta]\!]\eta\rho \cong [\![\Gamma\supset M:\alpha\to\beta]\!]\eta\rho \cdot [\![\Gamma\supset N:\alpha]\!]\eta\rho$
where: A ≜ [[α]]η, B = [[β]]η, $[h]_{pexp(A,B)} \cdot [d]_A \cong [hd]_B$.

(∀I) $[\![\Gamma\supset\lambda t.M:\forall t.\alpha]\!]\eta\rho \cong$ *if* ∃A.(f(A)⇑) *then* ⇑ *else* {h∈D | ∀A dcper. h∈f(A)}
where: f(A) ≅ [[Γ⊃ M:α]]η[A/t]ρ .

(∀E) $[\![\Gamma\supset M\beta:[\beta/t]\alpha]\!]\eta\rho \cong [\![\Gamma\supset M:\forall t.\alpha]\!]\eta\rho \cdot [\![\beta]\!]\eta$
where: F = λA.[[α]]η[A/t], $\bigcap F = \bigcap_{A\ dcper} F(A)$, $[h]_{\bigcap F}\cdot B \cong [h]_{F(B)}$.

(Y) $[\![\Gamma\supset YM:\alpha]\!]\eta\rho \cong [\ \bigsqcup_{n<\omega} k(n)\ ]_A$
where: $B = [\![(1\to\alpha)\to\alpha]\!]\eta$, $[k]_B \cong [\![\Gamma\supset M:(1\to\alpha)\to\alpha]\!]\eta\rho$,
k(0) ≅ ⇑, k(n+1) ≅ k i(λd∈D.k(n)) .

Note: We retain the attention of the reader on three points:

(1) Something has to be done in order to show in the clauses (→I) and (∀I) that certain collections of realizers are not empty.

(2) When we apply a term to a type (clause (∀E)), we keep the same realizer, this connects with the choice of evaluating under type abstraction.

(3) In the (Y) clause the existence of the fixed point combinator, which was announced in proposition 3.3.(4), takes a concrete shape. Its construction takes advantage of an iterator one can build in the realizability structure.

The following is proved by connecting the interpretation of a typed term in the per-model to the interpretation of its underlying type free-term in the realizability structure.

4.1 Proposition *(Typing Soundness)*
If $\vdash\Gamma \supset M{:}\alpha$ then, for any η and ρ such that $\eta \uparrow_\Gamma \rho$, we have:
$[\![\Gamma \supset M{:}\alpha]\!]\eta\rho \Downarrow \Rightarrow [\![\Gamma \supset M{:}\alpha]\!]\eta\rho \in [\,[\![\alpha]\!]\eta\,]$.

Proviso *(on type substitution)*
In the following $\sigma, \tau, \ldots$, denote substitutions of types for type variables. We now specify top-down what it means to apply such substitutions to a typing judgment. As usual one has to treat cautiously bound variables.

Judgment: $\sigma(\Gamma \supset M{:}\alpha) = \sigma\Gamma \supset \sigma M{:}\sigma\alpha$

Types: $\sigma 1 = 1$; $\sigma t = \sigma(t)$; $\sigma(\alpha\to\beta) = \sigma\alpha\to\sigma\beta$;
$\sigma(\forall t.\alpha) = \forall r.\sigma[r/t]\alpha$, where r is a fresh variable.

Contexts: $\sigma(x_1{:}\alpha_1,\ldots, x_n{:}\alpha_n) = x_1{:}\sigma\alpha_1,\ldots, x_n{:}\sigma\alpha_n$

Terms: $\sigma* = *$; $\sigma x = x$; $\sigma(\lambda x{:}\alpha.M) = (\lambda x{:}\sigma\alpha.\sigma M)$; $\sigma(MN) = \sigma(M)\,\sigma(N)$;
$\sigma(\lambda t.M) = \lambda r.\sigma[r/t]M$, where r is a fresh variable; $\sigma(M\alpha) = \sigma M\sigma\alpha$;
$\sigma(Y_\alpha M) = Y_{\sigma\alpha}\sigma M$

Having defined the notion of type substitution in a typing judgment the next thing to verify is that provability is invariant under type substitution, and that type and term substitutions commute with the respective semantic substitutions. The following lemmas are proved by induction on the length of the typing.

4.2 Lemma *(Type Substitution)*
Suppose $\vdash\Gamma \supset M{:}\alpha$. Then:
(1) If σ is a type substitution then $\vdash \sigma(\Gamma \supset M{:}\,\alpha)$.
(2) For any type-assignment η, for any type substitution σ, and for any assignment ρ such that $\eta \uparrow_{\sigma\Gamma} \rho$, we have:
$[\![\,\sigma(\Gamma \supset M{:}\,\alpha)\,]\!]\eta\rho \cong [\![\,\Gamma \supset M{:}\alpha\,]\!]\eta'\rho$
where: $\eta'(t) \triangleq [\![\,\sigma t\,]\!]\eta$.

4.3 Lemma *(Term Substitution)*
If $\vdash\Gamma,x{:}\alpha \supset M{:}\beta$, and $\vdash\Gamma \supset N{:}\alpha$ then
(1) $\vdash\Gamma \supset [N/x]M{:}\beta$
(2) For any type-assignment η, for any assignment ρ such that $\eta \uparrow_\Gamma \rho$,
$[\![\Gamma \supset N{:}\alpha]\!]\eta\rho \Downarrow \Rightarrow [\![\Gamma \supset [N/x]M{:}\beta]\!]\eta\rho \cong [\![\Gamma,x{:}\alpha \supset M{:}\beta]\!]\eta\rho'$
where: $\rho' \triangleq \rho[\,[\![\Gamma \supset N{:}\alpha]\!]\eta\rho\ /x\,]$.

The following is proved by induction on the structure of C.

4.4 Lemma *(Canonical Forms are Defined)*
If $\vdash\varnothing \supset C{:}\alpha$, where C is a canonical form, then for any η and ρ, $[\![\varnothing \supset C{:}\alpha]\!]\eta\rho \Downarrow$.

The following is proved by induction on the deduction of the evaluation judgment.

4.5 Lemma *(Invariance under Evaluation)*
If $\vdash M\mapsto C$ then, for any η and ρ, $[\![\varnothing \supset M{:}\alpha]\!]\eta\rho = [\![\varnothing \supset C{:}\alpha]\!]\eta\rho$.

5. Adequacy

We want to prove that given a well typed closed term M, $M\Downarrow$ iff $M\downarrow$. It is easy to show that if $M\downarrow$ then $M\Downarrow$, as the interpretation is invariant under evaluation (4.5), and the interpretation of a canonical form is defined (4.4). In the other direction an iterated attempt of generalizing the induction hypothesis leads to the following

5.1 Definition *(adequacy relation)*
Let η be a type assignment. A relation $S \subseteq |\,[\![\alpha]\!]\eta\,| \times \Lambda^o{}_\alpha$ is an adequacy relation of type α, w.r.t. the type assignment η, if it satisfies the following conditions:

(C.1) $h\,S\,M \Rightarrow M\downarrow$.
(C.2) $\{h_n\}_{n<\omega}$ directed in D $\wedge$ $\forall n.\ h_n S\,M \Rightarrow (\sqcup_{n<\omega} h_n)\,S\,M$.
(C.3) $(h\,S\,M \wedge \vdash M\mapsto C \wedge \vdash M'\mapsto C) \Rightarrow h\,S\,M'$.
(C.4) $(h\,S\,M \wedge h\,[\![\alpha]\!]\eta\,h') \Rightarrow h'\,S\,M$.

We denote with $AR(\alpha,\eta)$ the collection of adequacy relations of type α, w.r.t. the type assignment η. Observe that, for any type α, the empty set is an adequacy relation of type α.

If one thinks of h as an element in the equivalence class corresponding to the interpretation of the term M, then condition (C.1) corresponds to what we need to prove, i.e. $M\Downarrow \Rightarrow M\downarrow$. Condition (C.2) says that an adequacy relation is a kind of admissible predicate, in that it is closed under directed sets. Condition (C.3) says that an adequacy relation is invariant w.r.t terms that reduce to the same canonical form. The formulation of this condition seems to be new; it has the advantage of being simple and of not requiring a finer analysis of the evaluation relation as the

closure of a "one-step" reduction relation. Condition (C.4) says that an adequacy relation is invariant w.r.t. the equivalence induced by the corresponding per. This condition comes from the choice of representing adequacy relations as relations over the field of a per rather than over the collection of its equivalence classes .

We wish to assign to each type α an adequacy relation, parametrically in a type assignment η. In order to do this correctly we need to introduce two further parameters:

(i) A *type substitution* σ.

(ii) An *adequacy relation assignment*, θ: $tv \to \cup_\alpha type AR(\alpha, \eta)$, (henceforth ar-assignment) that depends on η.

Let η be a type assignment. A substitution σ and an ar-assignment θ are *compatible*, w.r.t η, if $\theta(t) \in AR(\sigma t, \eta)$, for any t. We write this fact as: $\sigma \uparrow_\eta \theta$.

Proviso *(on bounded variables and substitution)*

In order to simplify the notation, in the following definition and proofs, whenever we bring a substitution under a bound variable we assume that the bound variable has been suitably redenominated so that it does not interact with the substitution.

5.2 Definition *(associating adequacy relations to types)*

Let α be any type. For any type assignment η, any substitution σ, and any ar-assignment θ, such that $\sigma \uparrow_\eta \theta$, we define a relation

$R(\alpha, \sigma, \theta) \subseteq |[\![\sigma\alpha]\!]\eta| \times \Lambda^o_{\sigma\alpha}$

by induction on the structure of α, as follows:

(1) $R(1, \sigma, \theta) \triangleq \{(h, M) \in D \times \Lambda^o_1 \mid M\downarrow\}$

(tv) $R(t, \sigma, \theta) \triangleq \theta(t)$

($\rightarrow$) $R(\alpha \to \beta, \sigma, \theta) \triangleq \{(h, M) \in |[\![\sigma(\alpha \to \beta)]\!]\eta| \times \Lambda^o_{\sigma(\alpha \to \beta)} \mid$
$M\downarrow \wedge (d\ R(\alpha, \sigma, \theta)\ N \Rightarrow (hd\ R(\beta, \sigma, \theta)\ MN \vee hd\Uparrow))\}$

($\forall$) $R(\forall t.\alpha, \sigma, \theta) \triangleq \{(h, M) \in |[\![\sigma(\forall t.\alpha)]\!]\eta| \times \Lambda^o_{\sigma(\forall t.\alpha)} \mid$
$M\downarrow \wedge \forall \beta.\ \forall S \in AR(\beta, \eta).\ h\ R(\alpha, [\beta/t]\sigma, \theta[S/t])\ M\beta\}$

5.3 Lemma *(coherence of the definition)*

Let α be any type. For any type assignment η, any substitution σ, and any ar-assignment θ, such that $\sigma \uparrow_\eta \theta$, we have: $R(\alpha, \sigma, \theta) \in AR(\sigma\alpha, \eta)$.

Proof

By induction on the structure of the type α we verify that $R(\alpha, \sigma, \theta)$ is well defined and belongs to $AR(\sigma\alpha, \eta)$ as it satisfies (C.1-4).

(1) $R(1, \sigma, \theta) \in AR(1, \eta)$, as one can easily verify that $\{(h, M) \in D \times \Lambda^o_1 \mid M\downarrow\}$ satisfies (C.1-4).

(tv) $R(t, \sigma, \theta) = \theta(t) \in AR(\sigma t, \eta)$, by the assumption: $\sigma \uparrow_\eta \theta$.

($\rightarrow$) By definition $R(\alpha \to \beta, \sigma, \theta) \subseteq |[\![\sigma(\alpha \to \beta)]\!]\eta| \times \Lambda^o_{\sigma(\alpha \to \beta)}$, and satisfies (C.1).

(C.2): Suppose $\{h_n\}_{n<\omega}$ directed, and $\forall n.\ h_n R(\alpha \to \beta, \sigma, \theta)\ M$. Then:
(i) $\sqcup_{n<\omega} h_n \in |[\![\sigma(\alpha \to \beta)]\!]\eta|$, because $[\![\sigma(\alpha \to \beta)]\!]\eta \in$ dcper.
(ii) For any $d \in D$, $(\sqcup_{n<\omega} h_n)d \cong \sqcup_{n<\omega} h_n d$, and $\{h_n d \mid h_n d \Downarrow\}_{n<\omega}$ is directed, if not empty.
(iii) Suppose $d\ R(\alpha, \sigma, \theta)\ N$. There are two possibilities:
(a) $(\sqcup_{n<\omega} h_n)d \Uparrow$, and we are done.
(b) $(\sqcup_{n<\omega} h_n)d \Downarrow$, that implies $h_m d \Downarrow$, for some m, and therefore $h_n d\ R(\beta, \sigma, \theta)\ MN$, for all $n \geq m$. We can then apply the ind. hyp. on β to conclude $(\sqcup_{n<\omega} h_n d)\ R(\beta, \sigma, \theta)\ MN$, i.e. by (i, ii) $(\sqcup_{n<\omega} h_n)d\ R(\beta, \sigma, \theta)\ MN$.
(iv) By combining cases (iii.a-b) we have: $\sqcup_{n<\omega} h_n R(\alpha \to \beta, \sigma, \theta)\ M$.

(C.3): Suppose $(h\ R(\alpha \to \beta, \sigma, \theta)\ M \wedge \vdash M \mapsto C \wedge \vdash M' \mapsto C)$. Then $h\ R(\alpha \to \beta, \sigma, \theta)\ M'$ because:
(i) $M' \in \Lambda^o_{\sigma(\alpha \to \beta)}$ and $M'\downarrow$ (as the evaluation preserves the type).
(ii) Given N, $\vdash MN \mapsto C'$ iff $\vdash M'N \mapsto C'$.
(iii) Suppose $d\ R(\alpha, \sigma, \theta)\ N$. By (ii): $hd\Uparrow \vee (hd\Downarrow \wedge MN\downarrow \wedge M'N\downarrow)$. In the first case we are done, in the latter one shows: $hd\ R(\beta, \sigma, \theta)\ M'N$ by applying the inductive hypothesis on β, and (C.3) with (ii).

(C.4): Suppose $(h\ R(\alpha \to \beta, \sigma, \theta)\ M \wedge h\ [\![\sigma(\alpha \to \beta)]\!]\eta\ h')$.
Then, by definition of pexp: $d\ [\![\sigma\alpha]\!]\eta\ d' \Rightarrow (hd\ [\![\sigma\beta]\!]\eta\ h'd' \vee (hd\Uparrow \wedge h'd\Uparrow))$.
To show $h'\ R(\alpha \to \beta, \sigma, \theta)\ M$ we have to verify:
$d\ R(\alpha, \sigma, \theta)\ N \Rightarrow (h'd\ R(\beta, \sigma, \theta)\ MN \vee h'd\Uparrow)$.
But: $h'd\Uparrow \Leftrightarrow hd\Uparrow$, and $h'd\Downarrow \Rightarrow hd\Downarrow \Rightarrow (hd\ R(\beta, \sigma, \theta)\ MN)$
$\Rightarrow (h'd\ R(\beta, \sigma, \theta)\ MN)$, the last implication by ind. hyp. on β and (C.4).

($\forall$) By definition $R(\forall t.\alpha, \sigma, \theta) \subseteq |[\![\sigma(\forall t.\alpha)]\!]\eta| \times \Lambda^o_{\sigma(\forall t.\alpha)}$ and satisfies (C.1).
Observe that for any type β: $(\sigma \uparrow_\eta \theta \wedge S \in AR(\beta, \eta)) \Rightarrow [\beta/t]\sigma \uparrow_\eta \theta[S/t]$.

(C.2) Suppose $\{h_n\}_{n<\omega}$ directed, and $\forall n.\ h_n R(\forall t.\alpha, \sigma, \theta)\ M$. Then,
$\forall \beta.\ \forall S \in AR(\beta, \eta).\ h_n\ R(\alpha, [\beta/t]\sigma, \theta[S/t])\ M\beta$
Hence, by induction hypothesis on α, and (C.2):
$\forall \beta.\ \forall S \in AR(\beta, \eta).\ \sqcup_{n<\omega} h_n\ R(\alpha, [\beta/t]\sigma, \theta[S/t])\ M\beta$
and this implies by definition of $R(\forall t.\alpha, \sigma, \theta)$:
$\sqcup_{n<\omega} h_n\ R(\forall t.\alpha, \sigma, \theta)\ M$

(C.3): Suppose $(h\ R(\forall t.\alpha, \sigma, \theta)\ M \wedge \vdash M \mapsto C \wedge \vdash M' \mapsto C)$.
Then $h\ R(\forall t.\alpha, \sigma, \theta)\ M'$ because:
(i) $M' \in \Lambda^o_{\sigma(\forall t.\alpha)}$ and $M'\downarrow$.
(ii) For any β, $\vdash M\beta \mapsto C'$ iff $\vdash M'\beta \mapsto C'$.
(iii) For any β, for any $S \in AR(\beta, \eta)$, we can apply (C.3) on α (using (ii) and ind. hyp.), with substitution $[\beta/t]\sigma$, and ar-assignment $\theta[S/t]$ to conclude:
$h\ R(\alpha, [\beta/t]\sigma, \theta[S/t])\ M'\beta$. Hence: $h\ R(\forall t.\alpha, \sigma, \theta)\ M'$.

(C.4): Suppose $(h\ R(\forall t.\alpha, \sigma, \theta)\ M \wedge h\ [\![\sigma(\forall t.\alpha)]\!]\eta\ h')$.
Then, by definition: $\forall B$ dcper. $h\ [\![\sigma\alpha]\!]\eta[B/t]\ h'$.
To show $h'\ R(\forall t.\alpha, \sigma, \theta)\ M$ we have to verify:

$\forall\beta.\ \forall S\in AR(\beta,\eta).\ h'\ R(\alpha,[\beta/t]\sigma,\theta[S/t])\ M\beta$.
But for $B = [\![\beta]\!]\eta$, we have $[\![\sigma\alpha]\!]\eta[B/t] = [\![[\beta/t]\sigma\alpha]\!]\eta$, by the type susbstitution lemma. Hence $h\ [\![[\beta/t]\sigma\alpha]\!]\eta\ h'$, and we can apply ind. hyp. on α, and (C.4). □

5.4 Theorem *(semantic and syntactic convergence coincide)*
Suppose $\vdash (x_1{:}\ \alpha_1),\ldots,(x_n{:}\ \alpha_n) \supset M{:}\ \alpha$. Then for any type assignment η, any substitution σ, and any ar-assignment θ, such that $\sigma\uparrow_\eta\theta$, we have:

if $d_i\ R(\alpha_i,\sigma,\theta)\ C_i$, for i=1,..,n, and $[\![\sigma(\Gamma \supset M{:}\ \alpha)]\!]\eta[d/x] = [h]_A$
then $h\ R(\alpha,\sigma,\theta)\ [C/x]\sigma M$.

where: $\Gamma \equiv (x_1{:}\ \alpha_1),\ldots,(x_n{:}\ \alpha_n)$; $[d/x] \equiv [\ [d_1]_{A_1}/x_1,\ldots,[d_n]_{A_n}/x_n]$; $A_i \equiv [\![\sigma\alpha_i]\!]\eta$ for i=1,..,n; $A \equiv [\![\sigma\alpha]\!]\eta$; $[C/x] \equiv [C_1/x_1,\ldots,C_n/x_n]$.

Proof

In order to have a quick understanding of the statement observe, as an instance: if $\vdash \varnothing \supset M{:}\ \alpha$ where M and α have no free type variables, and $[\![\varnothing \supset M{:}\ \alpha]\!]\eta\rho = [h]_A$ then $hR(\alpha, id, \theta)M$, for arbitrary η, ρ, θ. Hence, by (C.1), we have: $M\Downarrow \Rightarrow M\downarrow$.

We now proceed with the proof, by induction on the length of the typing.

(*) Then $[\![\sigma\Gamma \supset *{:}\ 1]\!]\eta[d/x] = [d]_1$, and $d\ R(1,\sigma,\theta)\ *$, by def. of $R(1,\sigma,\theta)$.

(asmp) Then $[\![\sigma\Gamma \supset x_i{:}\ \sigma\alpha_i]\!]\eta[d/x] = [d_i]_{A_i}$, and $d_i\ R(\alpha_i,\sigma,\theta)\ C_i$, by assumption.

(→I) Suppose $h\in[\![\sigma(\Gamma \supset \lambda x{:}\alpha.M{:}\alpha\to\beta)]\!]\eta[d/x]$. Then by the interpretation definition: $\forall d\in |A|.\ (f(d)\Downarrow \Rightarrow hd\in f(d)) \wedge (f(d)\Uparrow \Rightarrow hd\Uparrow)\}$ where: $A = [\![\sigma\alpha]\!]\eta$,
$f(d) = [\![\sigma(\Gamma,x{:}\alpha \supset M{:}\beta)]\!]\eta[d/x][[d]_A/x]$.
We have to show: $h\ R(\alpha\to\beta,\sigma,\theta)\ [C/x]\sigma(\lambda x{:}\alpha.M)$, that is:
(i) $[C/x]\sigma(\lambda x{:}\alpha.M)\downarrow$, that holds because an abstraction always converges.
(ii) $d\ R(\alpha,\sigma,\theta)\ N \Rightarrow (hd\ R(\beta,\sigma,\theta)\ [C/x]\sigma(\lambda x{:}\alpha.M)\ N) \vee (hd\Uparrow)$.
Observe: $[C/x]\sigma(\lambda x{:}\alpha.M) \equiv \lambda x{:}\sigma\alpha.[C/x]\sigma M$.
Also: $\vdash (\lambda x{:}\sigma\alpha.[C/x]\sigma M)N\mapsto C$ iff $\vdash [C'/x][C/x]\sigma M\mapsto C$ and $\vdash N\mapsto C'$.
Suppose $hd\Downarrow$, we can apply the inductive hypothesis on the provable judgment $\Gamma,x{:}\alpha \supset M{:}\beta$ to get $hd\ R(\beta,\sigma,\theta)\ [C'/x][C/x]\sigma M$. Hence by (C.1) $[C'/x][C/x]\sigma M\downarrow$, *and by (C.3)* we get (ii).

(→E) Suppose $d\in[\![\sigma(\Gamma\supset MN{:}\beta)]\!]\eta[d/x]$ then we have to show $d\ R(\beta,\sigma,\theta)\ [C/x]\sigma(MN)$.
By the definition of the interpretation we have $h\in[\![\sigma(\Gamma\supset M{:}\alpha\to\beta)]\!]\eta[d/x]$, and $d'\in[\![\sigma(\Gamma\supset N{:}\alpha)]\!]\eta[d/x]$, such that $hd'[\![\sigma\beta]\!]\eta\ d$.
By induction hypothesis we have: $h\ R(\alpha\to\beta,\sigma,\theta)\ [C/x]\sigma(M)$, and
$d'\ R(\alpha,\sigma,\theta)[C/x]\sigma(N)$. *By definition of* $R(\alpha\to\beta,\sigma,\theta)$ we can conclude:
$hd'\ R(\beta,\sigma,\theta)\ [C/x]\sigma(MN)$, and by (C.4) $d\ R(\beta,\sigma,\theta)\ [C/x]\sigma(MN)$.

(∀I) Suppose $h\in[\![\sigma(\Gamma\supset\lambda t.M{:}\forall t.\alpha)]\!]\eta[d/x]$ then we have to show:
(i) $[C/x]\sigma(\lambda t.M)\downarrow$
(ii) $\forall\beta.\ \forall S\in AR(\beta,\eta).\ h\ R(\alpha,[\beta/t]\sigma,\theta[S/t])\ [C/x]\sigma(\lambda t.M)\beta$.
Observe: $[C/x]\sigma(\lambda t.M) \equiv \lambda t.[C/x]\sigma M$. By induction hypothesis applied to $\Gamma\supset M{:}\ \alpha$, we have, $\forall\beta.\ \forall S\in AR(\beta,\eta) : h\ R(\alpha,[\beta/t]\sigma,\theta[S/t])\ [C/x][\beta/t]\sigma M$.
Observe: $\vdash [C/x]\sigma(\lambda t.M)\beta \mapsto C'$ iff $\vdash [C/x][\beta/t]\sigma M \mapsto C'$, *and apply (C.3).*

(∀E) Suppose $h\in[\![\sigma(\Gamma\supset M\beta{:}[\beta/t]\alpha)]\!]\eta[d/x]$, then we have to show:
$h\ R([\beta/t]\alpha,\sigma,\theta)\ [C/x]\sigma(M\beta)$.
By induction hypothesis applied to $\Gamma\supset M{:}\ \forall t.\alpha$, and the *definition of* $R(\forall t.\alpha,\sigma,\theta)$ we have: $h\ R(\alpha,[\sigma\beta/t]\sigma,\theta[S/t])\ [C/x]\sigma M\sigma\beta$, and observe $\sigma[\beta/t]\alpha = [\sigma\beta/t]\sigma\alpha$.

(Y) Suppose $[\ \sqcup_{n<\omega} k(n)\ ]_A = [\![\sigma(\Gamma\supset YM{:}\alpha)]\!]\eta[d/x]$, where: $k\in[\![\sigma(\Gamma\supset M{:}(1\to\alpha)\to\alpha)]\!]\eta[d/x]$ for some k. We show that for any n big enough $k(n)\ R(\alpha,\sigma,\theta)\ [C/x]\sigma YM$. We conclude *by (C.2)*: $\sqcup_{n<\omega} k(n)\ R(\alpha,\sigma,\theta)\ [C/x]\sigma YM$. By induction hypothesis on $\Gamma\supset M{:}\ (1\to\alpha)\to\alpha$ we know: $k\ R((1\to\alpha)\to\alpha,\sigma,\theta)\ [C/x]\sigma M$. Observe:
(i) $i(\lambda d\in D.\Uparrow)\ R((1\to\alpha),\sigma,\theta)\ \lambda x{:}1.[C/x]\sigma YM$.
(ii) $i(\lambda d\in D.k(n))\ R((1\to\alpha),\sigma,\theta)\ \lambda x{:}1.[C/x]\sigma YM \Rightarrow$
$k\ i(\lambda d\in D.k(n))\ R(\alpha,\sigma,\theta)\ [C/x]\sigma M\ (\lambda x{:}1.[C/x]\sigma YM) \vee k(n+1)\Uparrow$.
Use (C.3) to conclude $k(n)\ R(\alpha,\sigma,\theta)\ [C/x]\sigma YM$, for n big enough. □

Note *(on the generalization of the inductive hypothesis)*

We have emphasized in italic the points in the proof above where one discovers the need to generalize the inductive hypothesis, and to adapt the definition of R.

(1) The need of working with a judgment, $\Gamma\supset M{:}\alpha$, where Γ may be a non empty context, already appears at the (→I) case. Here also appears the need for some condition of closure under expansions, this problem also arises in the cases (∀I), and (Y). How to formalize this condition is a tricky point, the simplest solution we could find is formalized as condition (C.3).

(2) In the case (→E) it becomes evident that an adequacy relation associated to an → type should satisfy certain functional properties. This leads to clause (→) of definition 5.2. A similar problem arises when dealing with second order types. Here however we hit the problem of resolving a potential circularity in definition 5.2. Hence the need to introduce an abstract notion of adequacy relation and the notion of ar-assignment. Substitution are introduced in 5.2 to ensure that the definition is "well-typed". We emphasize that similar problems arise in the context of proofs of (strong) normalization of Girard system F (see, e.g., Girard&al[89], chpt. 14).

(3) Finally the way the fixed point is computed in the interpretation naturally suggests the introduction of (C.2) in order to deal with clause (Y).

6. Conclusion

(1) *Relating model-theoretic and operational pre-orders*
We write $\Gamma\supset M\leq_{obs}N{:}\ \alpha$, if $\vdash\Gamma\supset M{:}\ \alpha$, $\vdash\Gamma\supset N{:}\ \alpha$, and for any "context" $C[\ ]$ such that $\vdash\varnothing\supset C[M]{:}\ \beta$, and $\vdash\varnothing\supset C[N]{:}\ \beta$, we have: $C[M]\downarrow \Rightarrow C[N]\downarrow$.
This defines an *operational* preorder on terms. In the standard domain theoretic case it is easy to prove, as a corollary of the correspondence between syntactic and semantic convergence, that the pre-order induced by the model (definition left to the reader) is contained in the operational preorder defined above.

When considering per-models there is a natural *intrinsic* way (Rosolini[86]) to order the equivalence classes:

Let A be a per over D. We define the intrinsic preorder $\leq_A$ over [A] as follows: $x \leq_A y$ if $\forall h: A \to 1.\ (h(x)\Downarrow \Rightarrow h(y)\Downarrow)$

How does the intrinsic pre-order relate to the operational pre-order ? We expect that the former is included in the latter, but to have a proof as simple as in the domain-theoretic case some additional model-theoretic information seems needed. In particular one needs to show that contexts induce monotone operators w.r.t. the intrinsic pre-order, and this seems to depend on the fact that products, used in the intepretation of second order types, have a pointwise intrinsic preordering. We do not have such a property available for the model described here. There are other per models that enjoy this property, e.g. the model of complete extensional pers over Kleene partial combinatory algebra (Freyd&al.[90]). We expect that the "architecture" of the proof of our main result (5.4) can be applied to such per-models. Hence we suspect that for such models a standard proof of the theorem relating intrinsic and operational pre-ordering will go through.

(2) *Call-by-name, Subtyping, and Recursive Types*

It seems worthwhile to recall that a call-by-name version of the calculus can be easily coded in the calculus presented here with the standard idea that the call-by-name functional space, say $\alpha \to_n \beta$, is coded as $(1 \to \alpha) \to \beta$. We also recall that per-models have been used as a semantic foundation for typed functional languages with a notion of *subtyping*(Cardelli&Longo[90]). Our result suggests that this is an *adequate* approach.

We expect that our main result extends to recursive types once we take as semantic structure the collection of complete "uniform" pers over a D-infinity model, as presented in Amadio[89]. The basic idea is to "stratify" the definition of the adequacy relation associated to a type. Formally one introduces an intermediary family of adequacy relation $R(n, \alpha, \sigma, \theta)$, where $n \in \omega$. The adequacy relation $R(\alpha, \sigma, \theta)$ is obtained by a process of completion of the sequence $\{R(n, \alpha, \sigma, \theta) \mid n \in \omega\}$. Roughly $R(n, \alpha, \sigma, \theta)$ represents $R(\alpha, \sigma, \theta)$ cut at the n-th level of the construction of the underlying D-infinity structure. We refrain from going into this point since the development of the model requires a certain number of rather ad hoc conditions that would only obscure the main ideas we have discussed.

(3) *Independence from the Adequacy of the Realizability Structure*

It is well know that to every, say closed, term M of type α one can associate its "erasure", i.e. a type free term er(M) such that: $[\![M]\!]^{per} \cong [\,[\![er(M)]\!]^D]_{[A]}$ (a).
In Amadio[90] we suggested that a "cheap" adequacy theorem could be obtained by the following schema of implications:

$$[\![M]\!]^{per}\Downarrow \Rightarrow_{(1)} [\![er(M)]\!]^D\Downarrow \Rightarrow_{(2)} er(M)\downarrow \Rightarrow_{(3)} M\downarrow$$

where (1) follows by a result of type (a), (2) follows by the adequacy of the realizability structure, and (3) follows by a comparison of the evaluations. The weak point of this chain of implications is (2). As a matter of fact we have shown that the adequacy of the per model is *independent* from the adequacy of the

realizability structure w.r.t. the related type-free language. Following Baeten&Boerboom[79] one can build a realizability structure D that is *not* adequate in that $[\![\Omega]\!]^D\Downarrow$, where $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$. Of course one can still hope to show (2) for terms coming from the erasure of a typed term. But then we have basically to reprove the same result presented here following a "type-assignment" style.

References

Amadio R. [1989] "Recursion over realizability structures", Info.&Comp., 91, 1, (55-85), also appeared as TR 1/89 Dipartimento di Informatica, Università di Pisa.

Amadio R. [1990] "Domains in a Realizability Framework", in Proc. CAAP91, Brighton, SLNCS 493, Abramsky S., Maibaum T. (eds.), (241-263), full version appeared as Liens TR 19-90, Paris.

Amadio R., Cardelli L. [1990] "Subtyping Recursive Types", in Proc. ACM-POPL91, Orlando, full version appeared as DEC-SRC TR #62, Palo Alto.

Baeten J, Boerboom B. [1979] "Ω can be anything it shouldn't be", Indag. Math., 41, (111-120).

Cardelli L. [1989] "The Quest language and system", pre-print DEC-SRC, Palo Alto.

Cardelli L., Longo G.[1990] "A semantic basis for Quest", in Proc. ACM-Lisp and Functional Programming 90, Nice.

Freyd P., Mulry P., Rosolini G., Scott D. [1990] "Extensional Pers", in Proc. 5th IEEE-LICS, Philadelphia.

Girard J.Y., Lafont Y., Taylor P. [1989] "Proofs and Types", Cambridge University Press.

Hyland M. [1991] "First steps in synthetic domain theory", in Proc. Category Theory 90, Carboni&al. (eds.), Springer-Verlag, (to appear).

Martin-Löf [1983] "The domain interpretation of type theory", in Proc. Workshop on the Semantics of Programming Languages, Dybier&al (eds.), Chalmers University, Göteborg.

Moggi E. [1988] "Partial morphisms in categories of effective objects", Info.&Comp., 76, (250-277).

Plotkin G. [1985] "Denotational semantics with partial functions", lecture notes, CSLI, Stanford 1985.

Rosolini G. [1986] "Continuity and effectiveness in Topoi", PhD Thesis, Oxford University.

Nancy, Wed, Dec 11, 1991.

Appendix A

In this appendix we present the proofs of the results stated in the first four sections of the paper.

Properties Semantic Structure *(Proposition 3.3)*

(1) **dcpper** is a pccc.

(2) dcpers are closed under arbitrary intersections.

(3) **dcpper** is reflective in **pper**.

(4) **dcpper** has fixpoints over objects of the shape pexp(**1**,A). Such fixpoints are the least ones (up to equivalence) w.r.t. the intrinsic pre-order.

Proof

We sketch the main ideas of the techniques described in Amadio[89], Amadio[90].

(1) It is enough to verify that (i) **1** is a dcper and that (ii) A×B, A→B are dcpers if A, and B are dcpers.

(2) Immediate.

(3) This means that the inclusion functor from **dcper** to **pper** has a left adjoint, say L: **pper**→**dcpper**. Define $L(A) = \bigcap \{B \in \text{dcper} \mid A \subseteq B\}$. This is well defined because of (2). Next show: $\forall A \in \text{per}.\ \forall B \in \text{dcper}.\ \text{pexp}(A, B) = \text{pexp}(L(A), B)$. Hint: this can be proved by induction once it has been observed that L(A) can be characterized as the closure of a certain operator G over A.

Namely define: $G(A') = TC(Sup(A'))$, for A' per

where: $Sup(A') = \{\sqcup X \mid X \subseteq A', X \text{ directed}\}$, and TC(A') is the transitive closure of A'.

Then inductively: $G_0(A) = A$, $G_{\alpha+1}(A) = G(G_\alpha(A))$,

$G_\lambda(A) = \bigcup_{\alpha<\lambda} G_\alpha(A)$ (for λ limit ordinal).

Hence $L(A) = G_\beta(A)$ for some β, and one shows $\text{pexp}(A, B) = \text{pexp}(G_\alpha(A), B)$ by induction.

(4) Define $Fix = i(\lambda k \in D.\ \sqcup_{n<\omega} k(n))$, where $k(0) \cong \Uparrow$, $k(n+1) \cong k\ i(\lambda d \in D.\ k(n))$.

One shows: for any **dcpper** A, $Fix \in |\text{pexp}(B, A)|$,

where B = pexp(pexp(**1**,A),A).

Observe $Fix\ k \cong \sqcup_{n<\omega} k(n)$. Show by induction on n that:

$k\ B\ k' \Rightarrow k(n)\ A\ k'(n) \vee (k(n)\Uparrow \wedge k'(n)\Uparrow)$.

Since A is a dcper conclude: $\sqcup_{n<\omega}k(n)\ A\ \sqcup_{n<\omega}k'(n) \vee (\sqcup_{n<\omega}k(n)\Uparrow \wedge \sqcup_{n<\omega}k'(n)\Uparrow)$. □

Typing Soundness *(Proposition 4.1)*

If $\vdash \Gamma \supset M:\alpha$ then, for any η and ρ such that $\eta \uparrow_\Gamma \rho$, we have:

$[\![\Gamma \supset M:\alpha]\!]\eta\rho \Downarrow \Rightarrow [\![\Gamma \supset M:\alpha]\!]\eta\rho \in [\,[\![\alpha]\!]\eta\,]$.

A direct proof of this result by induction on the length of the typing judgment does not seem to work as we hit the problem of showing in the case (→I), and (∀I) that certain collections of realizers are non-empty. Hence, it appears that a natural argument to prove the existence of such realizers is to consider the interpretation in the realizability structure of an underlying type-free lambda term.

The function *er* (erasure) takes a typed term and returns a type-free term in the language generated by the following BNF (as usual v stands for the variables):

$P ::= * \mid v \mid (\lambda v.P) \mid (PP) \mid (Y_V\ P)$

It is defined by induction on the structure as follows:

$er(*) = *$; $er(x) = x$; $er(\lambda x{:}\alpha.M) = (\lambda x.er(M))$; $er(MN) = (er(M)er(N))$;

$er(\lambda t.M) = er(M)$; $er(M\beta) = er(M)$; $er(YM) = (Y_V er(M))$

Given a realizability structure (D, i, j) as in 3.1 we can define a standard interpretation of the lambda calculus introduced above. We denote with τ a partial function from variables to D. Define:

(*) $[\![*]\!]\tau = d$ for some fixed $d \in D$ (D supposed non-empty !)

(asmp) $[\![x]\!]\tau \cong \tau(x)$

(abs) $[\![\lambda x.P]\!]\tau = i(\lambda d \in D.[\![P]\!]\tau[d/x])$.

(apl) $[\![PQ]\!]\tau \cong j([\![P]\!]\tau)([\![Q]\!]\tau)$

(Y_V) $[\![Y_V P]\!]\tau \cong \sqcup_{n<\omega} h(n)$

where: $h \cong [\![P]\!]\tau$, $h(0) \cong \Uparrow$, $h(n+1) \cong h\ i(\lambda d \in D.h(n))$.

Given a context Γ, a type assignment η, an assignment ρ s.t. $\eta \uparrow_\Gamma \rho$, and an assignment τ, we say that ρ is compatible with τ w.r.t. Γ, and write $\rho \uparrow_\Gamma \tau$, if for any variable x in Γ: $(\rho(x)\Downarrow \Leftrightarrow \tau(x)\Downarrow) \wedge (\rho(x)\Downarrow \Rightarrow \tau(x) \in \rho(x))$.

We can now state a proposition stronger than 4.1.

Proposition *(erasure)*

Suppose $\vdash \Gamma \supset M:\alpha$. Then for any η, for any ρ s.t. $\eta \uparrow_\Gamma \rho$, for any τ s.t. $\rho \uparrow_\Gamma \tau$:

(1) $[\![\Gamma \supset M:\alpha]\!]\eta\rho \Downarrow \Leftrightarrow [\![er(M)]\!]\tau \Downarrow$.

If $[\![\Gamma \supset M:\alpha]\!]\eta\rho \Downarrow$ then:

(2) $[\![er(M)]\!]\tau \in |A|$, where $A = [\![\alpha]\!]\eta$.

(3) $[\![\Gamma \supset M:\alpha]\!]\eta\rho = [\,[\![er(M)]\!]\tau\,]_A$.

Proof

By induction on the length of the typing. We refer to the notation in the interpretation definition.

(*) (1) $[\![\Gamma \supset *: 1]\!]\eta\rho = [d]_1$, $[\![er(M)]\!]\tau = d$. (2) $d \in D$, $D = |[\![1]\!]\eta|$. (3) $d \in [d]_1$.

(asmp) (1-3) By definition of $\rho \uparrow_\Gamma \tau$.

(→I) (1) Both interpretations are always defined.

(2) Use ind. hyp. on Γ, $x{:}\alpha \supset M:\beta$ to show:

$d\ A\ e \Rightarrow ([\![er(M)]\!]\tau[d/x]\Uparrow \wedge [\![er(M)]\!]\tau[e/x]\Uparrow) \vee ([\![er(M)]\!]\tau[d/x]\ B\ [\![er(M)]\!]\tau[e/x])$.

(3) Let $x = \{h \in D \mid \forall d \in |A|.\ (fd\Downarrow \Rightarrow hd \in f(d)) \wedge (fd\Uparrow \Rightarrow hd\Uparrow)\}$. We show:

(i) $h, h' \in x \Rightarrow h\ [\![\alpha{\to}\beta]\!]\eta\ h'$, and (ii) $h\ [\![\alpha{\to}\beta]\!]\eta\ h' \wedge h \in x \Rightarrow h' \in x$. Apply ind. hyp. on f(d). (i) Observe: $d\ [\![\alpha]\!]\eta\ d' \Rightarrow (f(d)\Downarrow \Rightarrow hd, hd' \in f(d)) \wedge (f(d)\Uparrow \Rightarrow hd\Uparrow \wedge hd'\Uparrow)$.

(ii) Observe: $\forall d \in |A|.\ (f(d)\Downarrow \Rightarrow hd, h'd \in f(d)) \wedge (f(d)\Uparrow \Rightarrow hd\Uparrow \wedge h'd\Uparrow)$.

Finally prove: $[\![\lambda x.er(M)]\!]\tau \in [\![\Gamma \supset \lambda x{:}\alpha.\ M{:}\beta]\!]\eta\rho$.

(→E) Left to the reader.

(∀I) (1) $[\![\Gamma \supset \lambda t.M:\forall t.\alpha]\!]\eta\rho\Downarrow \Leftrightarrow \forall A.(f(A)\Downarrow) \Leftrightarrow [\![er(M)]\!]\tau\Downarrow \Leftrightarrow [\![er(\lambda t.M)]\!]\tau\Downarrow$.
(2) If $\forall A.\ [\![er(M)]\!]\tau \in |[\![\alpha]\!]\eta[A/t]|$ then $[\![er(M)]\!]\tau \in |\cap_{A\ dcper}[\![\alpha]\!]\eta[A/t]|$.
(3) Let $x = \{h \in D \mid \forall A\ dcper.\ h \in f(A)\}$. We show:
(i) $h, h' \in x \Rightarrow h \cap_{A\ dcper}[\![\alpha]\!]\eta[A/t]\ h'$. (ii) $h \cap_{A\ dcper}[\![\alpha]\!]\eta[A/t]\ h', h \in x \Rightarrow h' \in x$.
(i) Since by ind. hyp., for any A dcper, $h, h' \in f(A) \in [\![\alpha]\!]\eta[A/t]]$. (ii) Since, for any A dcper, $h\ [\![\alpha]\!]\eta[A/t]\ h', h \in f(A) \Rightarrow h' \in f(A)$.

(∀E) Left to the reader.

(Y) Left to the reader, this requires an inductive argument of the type already seen for the (Y) clasuse. □

Note

Of course we could define a combinator Y_V directly in the type-free calculus as follows:

$$Y_V \equiv \lambda f.\ f\ (\lambda z.(\omega_V(f)\ \omega_V(f))),\ \ \omega_V(f) \equiv \lambda x.\ f(\lambda z.\ xx).$$

In order to have an intuition of Y_V behavior let us β-reduce $(Y_V\ f)$:

$$(Y_V\ f) \to_\beta f\ (\lambda z.(\omega_V(f)\ \omega_V(f))), \text{ and}$$
$$(\omega_V(f)\ \omega_V(f)) \equiv (\lambda x.\ f(\lambda z.\ xx)))\ \omega_V(f) \to_\beta f(\lambda z.\ (\omega_V(f)\ \omega_V(f)))$$

One may think of $Y_V\ f$ as: $f\ (\lambda z.f(\lambda z.\ f(\lambda z.\ f(\lambda z.\ ....))))$. The problem is that there is no reason why the interpretation of Y_V should behave like the interpretation given in the clause (Y_V) above, and we need such an intepretation to prove the "erasure" proposition! For instance taking the realizability structure described in Appendix B we have: $[\![\ Y_V\ (\lambda f.\ f *)\ ]\!]\tau\Downarrow$, whereas we wish: $[\![\ Y\ (\lambda f: 1\to\alpha.\ f *)\ ]\!]\eta\rho \Uparrow$. □

Type Substitution *(Lemma 4.2)*

Suppose $\vdash\Gamma \supset M:\alpha$. Then:

(1) If σ is a type substitution then $\vdash \sigma(\Gamma \supset M:\alpha)$.

(2) For any type-assignment η, for any type substitution σ, and for any assignment ρ such that $\eta\uparrow_{\sigma\Gamma}\rho$, we have:

$$[\![\ \sigma(\Gamma \supset M:\alpha)\ ]\!]\eta\rho \cong [\![\ \Gamma \supset M:\alpha\ ]\!]\eta'\rho$$

where: $\eta'(t) \triangleq [\![\ \sigma t\ ]\!]\eta$.

Proof

Both assertions are proved by induction on the length of the typing.

(1) (*) If $\vdash\Gamma \supset *:1$ then $\vdash\sigma\Gamma \supset *:1$, since σΓ is a context.

(asmp) If $x:\alpha \in \Gamma$ then $x:\sigma\alpha \in \sigma\Gamma$.

(→I) If $\vdash\Gamma,x:\alpha \supset M:\beta$ then $\vdash\sigma\Gamma,x:\sigma\alpha \supset \sigma M:\sigma\beta$, by induction hypothesis. Hence: $\vdash\sigma\Gamma \supset \lambda x:\sigma\alpha.\ \sigma M:\sigma\alpha\to\sigma\beta \equiv \sigma(\Gamma \supset \lambda x:\alpha.\ M:\alpha\to\beta)$.

(→E) If $\vdash\Gamma \supset M:(\alpha\to\beta)$ and $\vdash\Gamma \supset N:\alpha$ then $\vdash\sigma\Gamma \supset \sigma M:(\sigma\alpha\to\sigma\beta)$ and $\vdash\sigma\Gamma \supset \sigma N:\sigma\alpha$. Hence: $\vdash\sigma\Gamma \supset (\sigma M\sigma N):\sigma\beta \equiv \sigma(\Gamma \supset (MN):\beta)$.

(∀I) Redenominate t so that it does not interfere with the substitution. If $\vdash\Gamma \supset$. and $t \notin ftv(\Gamma)$ then $\vdash \sigma(\Gamma \supset M:\alpha)$, by induction hypothesis. Hence $\vdash \sigma\Gamma \supset \lambda t.\sigma M:\forall t.\sigma\alpha$.

(∀E) If $\vdash \Gamma \supset M:(\forall t.\alpha)$ then $\vdash \sigma\Gamma \supset \sigma M:\forall t.\sigma\alpha$ for suitable t, by induction hypothesis. Hence: $\vdash \sigma\Gamma \supset \sigma M\sigma\beta:[\sigma\beta/t]\sigma\alpha \equiv \sigma(\Gamma \supset (M\beta):[\beta/t]\alpha)$.

(Y) If $\vdash \Gamma \supset M:(1\to\alpha)\to\alpha$ then $\vdash \sigma\Gamma \supset \sigma M:(1\to\sigma\alpha)\to\sigma\alpha$, by induction hypothesis. Hence: $\vdash \sigma\Gamma \supset Y_{\sigma\alpha}\sigma M:\sigma\alpha \equiv \sigma(\Gamma \supset Y_\alpha M:\alpha)$.

(2) Observe $\eta\uparrow_{\sigma\Gamma}\rho \Rightarrow \eta'\uparrow_\Gamma\rho$, as for any $(x:\alpha)$ in Γ: $[\![\sigma\alpha]\!]\eta = [\![\alpha]\!]\eta'$.

(*) $[\![\ \sigma(\Gamma \supset *:1)\ ]\!]\eta\rho = [d]_1 = [\![\ \Gamma \supset *:1\ ]\!]\eta'\rho$.

(asmp) $[\![\ \sigma(\Gamma \supset x:\alpha)\ ]\!]\eta\rho \cong \rho(x) \cong [\![\ \Gamma \supset x:\alpha\ ]\!]\eta'\rho$.

(→I) Observe $\sigma(\Gamma,x:\alpha \supset M:\beta) = \sigma\Gamma,x:\sigma\alpha \supset \sigma M:\sigma\beta$. For any $d \in |[\![\sigma\alpha]\!]\eta| = |[\![\alpha]\!]\eta'|$, we have by ind. hyp.: $[\![\sigma\Gamma,x:\sigma\alpha \supset \sigma M:\sigma\beta]\!]\eta\rho[[d]_A/x] \cong [\![\Gamma,x:\alpha \supset M:\beta]\!]\eta'\rho[[d]_A/x]$. Hence by the definition of the interpretation:

$$[\![\sigma(\Gamma \supset \lambda x:\alpha.M:\alpha\to\beta)]\!]\eta\rho = [\![\Gamma \supset \lambda x:\alpha.M:\alpha\to\beta]\!]\eta'\rho.$$

(→E) Immediate application of the inductive hypothesis.

(∀I) As usual we assume that the bound variable t has been suitably redenominated. Then for $t \notin ftv(\Gamma)$ we have: $\eta[A/t]\uparrow_{\sigma\Gamma}\rho$, for any A. Hence by ind. hyp.: $[\![\sigma\Gamma \supset \sigma M:\sigma\alpha]\!]\eta[A/t]\rho \cong [\![\Gamma \supset M:\alpha]\!]\eta'[A/t]\rho$. By the definition of the interpretation: $[\![\sigma(\Gamma \supset \lambda t.M:\forall t.\alpha)]\!]\eta\rho \cong [\![\Gamma \supset \lambda t.M:\forall t.\alpha]\!]\eta'\rho$.

(∀E) Apply ind. hyp. observing: $[\![\sigma([\beta/t]\alpha)]\!]\eta = [\![\ [\beta/t]\alpha\ ]\!]\eta'$.

(Y) Again an easy application of the ind. hyp. . □

Term Substitution *(Lemma 4.3)*

If $\vdash\Gamma,x:\alpha \supset M:\beta$, and $\vdash\Gamma \supset N:\alpha$ then

(1) $\vdash\Gamma \supset [N/x]M:\beta$

(2) For any type-assignment η, for any assignment ρ such that $\eta\uparrow_\Gamma\rho$,

$$[\![\Gamma \supset N:\alpha]\!]\eta\rho\Downarrow \Rightarrow [\![\Gamma \supset [N/x]M:\beta]\!]\eta\rho \cong [\![\Gamma,x:\alpha \supset M:\beta]\!]\eta\rho'$$

where: $\rho' \triangleq \rho[\ [\![\Gamma \supset N:\alpha]\!]\eta\rho\ /x\ ]$.

Proof

Both assertions are proved by induction on the length of the typing.

(1) (*) $[N/x] * \equiv *$, and $\vdash\Gamma \supset *:1$.

(asmp) Say: $y:\beta \in \Gamma \Rightarrow \Gamma,\ x:\alpha \supset y:\beta$. If $x \equiv y$ then $[N/x]y \equiv N$, and $\vdash\Gamma \supset N:\alpha$; otherwise $\vdash\Gamma \supset y:\beta$, by (asmp).

(→I) First we observe that a rule of exchange is derived in the sense that: $\vdash\Gamma,\ x:\alpha,\ y:\alpha' \supset M:\beta$ implies $\vdash\Gamma,\ y:\alpha',\ x:\alpha \supset M:\beta$ with a proof of the same length. Next, say: $\Gamma,x:\alpha,\ y:\beta \supset M:\beta' \Rightarrow \Gamma,x:\alpha \supset \lambda y:\beta.M:\beta\to\beta'$. Then we apply the ind. hyp. on $\Gamma,\ y:\beta,\ x:\alpha \supset M:\beta'$ to conclude: $\Gamma,\ y:\beta \supset [N/x]M:\beta'$. By (→I) we conclude: $\vdash\Gamma \supset \lambda y:\beta.[N/x]M:\beta\to\beta'$.

(→E) Direct application of ind. hyp. .

(∀I) Say: $\Gamma,x:\alpha \supset M:\alpha\ \ t \notin ftv(\Gamma,x:\alpha) \Rightarrow \Gamma,x:\alpha \supset (\lambda t.M):(\forall t.\alpha)$. By ind. hyp. $\vdash\Gamma \supset$

(→E) Left to the reader.

(∀I) (1) $[\![\Gamma \supset \lambda t.M:\forall t.\alpha]\!]\eta\rho\Downarrow \Leftrightarrow \forall A.(f(A)\Downarrow) \Leftrightarrow [\![er(M)]\!]\tau\Downarrow \Leftrightarrow [\![er(\lambda t.M)]\!]\tau\Downarrow$.
(2) If $\forall A.\ [\![er(M)]\!]\tau \in |[\![\alpha]\!]\eta[A/t]|$ then $[\![er(M)]\!]\tau \in |\cap_{A\ dcper}[\![\alpha]\!]\eta[A/t]|$.
(3) Let $x = \{h \in D \mid \forall A\ dcper.\ h \in f(A)\}$. We show:
(i) $h, h' \in x \Rightarrow h \cap_{A\ dcper}[\![\alpha]\!]\eta[A/t]\ h'$. (ii) $h \cap_{A\ dcper}[\![\alpha]\!]\eta[A/t]\ h', h \in x \Rightarrow h' \in x$.
(i) Since by ind. hyp., for any A dcper, h, $h' \in f(A) \in [\![\alpha]\!]\eta[A/t]]$. (ii) Since, for any A dcper, $h\ [\![\alpha]\!]\eta[A/t]\ h', h \in f(A) \Rightarrow h' \in f(A)$.

(∀E) Left to the reader.

(Y) Left to the reader, this requires an inductive argument of the type already seen for the (Y) clasuse. □

Note

Of course we could define a combinator Y_V directly in the type-free calculus as follows:

$$Y_V \equiv \lambda f.\ f\ (\lambda z.(\omega_V(f)\ \omega_V(f))),\ \ \omega_V(f) \equiv \lambda x.\ f(\lambda z.\ xx).$$

In order to have an intuition of Y_V behavior let us β-reduce $(Y_V\ f)$:

$$(Y_V\ f) \to_\beta f\ (\lambda z.(\omega_V(f)\ \omega_V(f))), \text{ and}$$
$$(\omega_V(f)\ \omega_V(f)) \equiv (\lambda x.\ f(\lambda z.\ xx)))\ \omega_V(f) \to_\beta f(\lambda z.\ (\omega_V(f)\ \omega_V(f)))$$

One may think of $Y_V\ f$ as: $f\ (\lambda z.f(\lambda z.\ f(\lambda z.\ f(\lambda z.\ ....))))$. The problem is that there is no reason why the interpretation of Y_V should behave like the interpretation given in the clause (Y_V) above, and we need such an intepretation to prove the "erasure" proposition! For instance taking the realizability structure described in Appendix B we have: $[\![\ Y_V\ (\lambda f.\ f *)\ ]\!]\tau\Downarrow$, whereas we wish: $[\![\ Y\ (\lambda f: 1\to\alpha.\ f *)\ ]\!]\eta\rho\Uparrow$. □

Type Substitution *(Lemma 4.2)*

Suppose $\vdash\Gamma \supset M:\alpha$. Then:

(1) If σ is a type substitution then $\vdash \sigma(\Gamma \supset M:\alpha)$.

(2) For any type-assignment η, for any type substitution σ, and for any assignment ρ such that $\eta\uparrow_{\sigma\Gamma}\rho$, we have:

$$[\![\ \sigma(\Gamma \supset M:\alpha)\ ]\!]\eta\rho \cong [\![\ \Gamma \supset M:\alpha\ ]\!]\eta'\rho$$

where: $\eta'(t) \triangleq [\![\ \sigma t\ ]\!]\eta$.

Proof

Both assertions are proved by induction on the length of the typing.

(1) (*) If $\vdash\Gamma \supset *: 1$ then $\vdash\sigma\Gamma \supset *: 1$, since σΓ is a context.

(asmp) If $x:\alpha \in \Gamma$ then $x:\sigma\alpha \in \sigma\Gamma$.

(→I) If $\vdash\Gamma,x:\alpha \supset M:\beta$ then $\vdash\sigma\Gamma,x:\sigma\alpha \supset \sigma M:\sigma\beta$, by induction hypothesis. Hence: $\vdash\sigma\Gamma \supset \lambda x:\sigma\alpha.\ \sigma M:\sigma\alpha\to\sigma\beta \equiv \sigma(\Gamma \supset \lambda x:\alpha.\ M:\alpha\to\beta)$.

(→E) If $\vdash\Gamma \supset M:(\alpha\to\beta)$ and $\vdash\Gamma \supset N:\alpha$ then $\vdash\sigma\Gamma \supset \sigma M:(\sigma\alpha\to\sigma\beta)$ and $\vdash\sigma\Gamma \supset \sigma N:\sigma\alpha$. Hence: $\vdash\sigma\Gamma \supset (\sigma M\sigma N):\sigma\beta \equiv \sigma(\Gamma \supset (MN):\beta)$.

(∀I) Redenominate t so that it does not interfere with the substitution. If $\vdash\Gamma \supset M:\alpha$ and $t \notin ftv(\Gamma)$ then $\vdash \sigma(\Gamma \supset M:\alpha)$, by induction hypothesis. Hence $\vdash \sigma\Gamma \supset \lambda t.\sigma M:\forall t.\sigma\alpha$.

(∀E) If $\vdash\Gamma \supset M:(\forall t.\alpha)$ then $\vdash\sigma\Gamma \supset \sigma M:\forall t.\sigma\alpha$ for suitable t, by induction hypothesis. Hence: $\vdash\sigma\Gamma \supset \sigma M\sigma\beta:[\sigma\beta/t]\sigma\alpha \equiv \sigma(\Gamma \supset (M\beta):[\beta/t]\alpha)$.

(Y) If $\vdash\Gamma \supset M:(1\to\alpha)\to\alpha$ then $\vdash\sigma\Gamma \supset \sigma M:(1\to\sigma\alpha)\to\sigma\alpha$, by induction hypothesis. Hence: $\vdash\sigma\Gamma \supset Y_{\sigma\alpha}\sigma M:\sigma\alpha \equiv \sigma(\Gamma \supset Y_\alpha M:\alpha)$.

(2) Observe $\eta\uparrow_{\sigma\Gamma}\rho \Rightarrow \eta'\uparrow_\Gamma\rho$, as for any $(x:\alpha)$ in Γ: $[\![\sigma\alpha]\!]\eta = [\![\alpha]\!]\eta'$.

(*) $[\![\ \sigma(\Gamma \supset *: 1)\ ]\!]\eta\rho = [d]_1 = [\![\ \Gamma \supset *: 1\ ]\!]\eta'\rho$.

(asmp) $[\![\ \sigma(\Gamma \supset x:\alpha)\ ]\!]\eta\rho \cong \rho(x) \cong [\![\ \Gamma \supset x:\alpha\ ]\!]\eta'\rho$.

(→I) Observe $\sigma(\Gamma,x:\alpha \supset M:\beta) = \sigma\Gamma,x:\sigma\alpha \supset \sigma M:\sigma\beta$. For any $d \in |[\![\sigma\alpha]\!]\eta| = |[\![\alpha]\!]\eta'|$, we have by ind. hyp.: $[\![\sigma\Gamma,x:\sigma\alpha \supset \sigma M:\sigma\beta]\!]\eta\rho[[d]_A/x] \cong [\![\Gamma,x:\alpha \supset M:\beta]\!]\eta'\rho[[d]_A/x]$. Hence by the definition of the interpretation:

$$[\![\sigma(\Gamma \supset \lambda x:\alpha.M:\alpha\to\beta)]\!]\eta\rho = [\![\Gamma \supset \lambda x:\alpha.M:\alpha\to\beta]\!]\eta'\rho.$$

(→E) Immediate application of the inductive hypothesis.

(∀I) As usual we assume that the bound variable t has been suitably redenominated. Then for $t \notin ftv(\Gamma)$ we have: $\eta[A/t]\uparrow_{\sigma\Gamma}\rho$, for any A. Hence by ind. hyp.: $[\![\sigma\Gamma \supset \sigma M:\sigma\alpha]\!]\eta[A/t]\rho \cong [\![\Gamma \supset M:\alpha]\!]\eta'[A/t]\rho$. By the definition of the interpretation: $[\![\sigma(\Gamma \supset \lambda t.M:\forall t.\alpha)]\!]\eta\rho \cong [\![\Gamma \supset \lambda t.M:\forall t.\alpha]\!]\eta'\rho$.

(∀E) Apply ind. hyp. observing: $[\![\sigma([\beta/t]\alpha)]\!]\eta = [\![\ [\beta/t]\alpha\ ]\!]\eta'$.

(Y) Again an easy application of the ind. hyp. . □

Term Substitution *(Lemma 4.3)*

If $\vdash\Gamma,x:\alpha \supset M:\beta$, and $\vdash\Gamma \supset N:\alpha$ then

(1) $\vdash\Gamma \supset [N/x]M:\beta$

(2) For any type-assignment η, for any assignment ρ such that $\eta\uparrow_\Gamma\rho$,

$$[\![\Gamma \supset N:\alpha]\!]\eta\rho\Downarrow \Rightarrow [\![\Gamma \supset [N/x]M:\beta]\!]\eta\rho \cong [\![\Gamma,x:\alpha \supset M:\beta]\!]\eta\rho'$$

where: $\rho' \triangleq \rho[\ [\![\Gamma \supset N:\alpha]\!]\eta\rho\ /x\ ]$.

Proof

Both assertions are proved by induction on the length of the typing.

(1) (*) $[N/x] * \equiv *$, and $\vdash\Gamma \supset *: 1$.

(asmp) Say: $y:\beta \in \Gamma \Rightarrow \Gamma,\ x:\alpha \supset y:\beta$. If $x \equiv y$ then $[N/x]y \equiv N$, and $\vdash\Gamma \supset N:\alpha$; otherwise $\vdash\Gamma \supset y:\beta$, by (asmp).

(→I) First we observe that a rule of exchange is derived in the sense that: $\vdash\Gamma,\ x:\alpha,\ y:\alpha' \supset M:\beta$ implies $\vdash\Gamma,\ y:\alpha',\ x:\alpha \supset M:\beta$ with a proof of the same length. Next, say: $\Gamma,x:\alpha,\ y:\beta \supset M:\beta' \Rightarrow \Gamma,x:\alpha \supset \lambda y:\beta.M:\beta\to\beta'$. Then we apply the ind. hyp. on $\Gamma,\ y:\beta,\ x:\alpha \supset M:\beta'$ to conclude: $\Gamma,\ y:\beta \supset [N/x]M:\beta'$. By (→I) we conclude: $\vdash\Gamma \supset \lambda y:\beta.[N/x]M:\beta\to\beta'$.

(→E) Direct application of ind. hyp. .

(∀I) Say: $\Gamma,x:\alpha \supset M:\alpha$ $t \notin ftv(\Gamma,x:\alpha) \Rightarrow \Gamma,x:\alpha \supset (\lambda t.M):(\forall t.\alpha)$. By ind. hyp. $\vdash\Gamma \supset$

$[N/x]M: \alpha$, and by ($\forall$I) we conclude: $\Gamma \supset (\lambda t.[N/x]M): (\forall t.\alpha)$.

($\forall$E) Direct application of ind. hyp. .

(Y) Direct application of ind. hyp.

(2) Left to the reader. □

Canonical Forms are Defined *(Lemma 4.4)*

If $\vdash \varnothing \supset C:\alpha$ where C is a canonical form, then for any η and ρ, $[\![\varnothing \supset C:\alpha]\!]\eta\rho \Downarrow$.

Proof

We recall: C::= * | (λv: α. M) | (λtv. C). Hence we proceed by induction on the structure of C.

(C ≡ *) Then $[\![\varnothing \supset *:1]\!]\eta\rho \Downarrow$.

(C ≡ λx: α. M) Then $[\![\varnothing \supset \lambda x{:}\alpha.\, M{:}\, \alpha\to\beta]\!]\eta\rho \Downarrow$.

(C ≡ λt. C) Then, for any A, $[\![\varnothing \supset C{:}\, \alpha]\!]\eta[A/t]\rho \Downarrow$ by inductive hypothesis, and therefore $[\![\varnothing \supset \lambda t.C{:}\, \forall t\, \alpha]\!]\eta\rho \Downarrow$. □

Invariance under Evaluation *(Lemma 4.5)*

If $\vdash M \mapsto C$ then, for any η and ρ, $[\![\varnothing \supset M:\alpha]\!]\eta\rho = [\![\varnothing \supset C:\alpha]\!]\eta\rho$.

Proof

By induction on the deduction of the evaluation judgment.

(*), ($\to$I) Trivial.

($\to$E) By applying the ind. hyp. and the term substitution lemma we have:

$[\![\varnothing \supset MN : \beta]\!]\eta\rho = [\![\varnothing \supset \lambda x{:}\alpha.M' : \alpha\to\beta]\!]\eta\rho\ [\![\varnothing \supset C' : \alpha]\!]\eta\rho =$

$[\![x{:}\alpha \supset M' : \beta]\!]\eta\rho[\, [\![\varnothing \supset C' : \alpha]\!]\eta\rho\ /x\,] = [\![\varnothing \supset [C'/x]M' : \beta]\!]\eta\rho = [\![\varnothing \supset C : \beta]\!]\eta\rho$.

($\forall$I) By ind. hyp. .

($\forall$E) By applying the ind. hyp. and the type substitution lemma we have:

$[\![\varnothing \supset M\alpha : [\alpha/t]\beta]\!]\eta\rho = [\![\varnothing \supset (\lambda t.C) : \forall t.\beta]\!]\eta\rho\ [\![\alpha]\!]\eta\rho =$

$[\![\varnothing \supset C : \beta]\!]\eta[[\![\alpha]\!]\eta/t]\rho = [\![\varnothing \supset [\alpha/t]C : [\alpha/t]\beta]\!]\eta\rho$.

(Y) By ind. hyp. and $[\![\varnothing \supset M(\lambda x{:}1.YM) : \alpha]\!]\eta\rho = [\![\varnothing \supset (YM) : \alpha]\!]\eta\rho$. □

Appendix B

In this section we build a realizability structure D such that $[\![\Omega]\!]^D\Downarrow$, where $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$. The construction is a straightforward adaptation of the Pω model construction, and of certain observations in Baeten&Boerboom[79] on the surprising effects that certain codings can have on the interpretation.

Consider the collection of subsets of natural numbers ordered by inclusion, say Pω, as an object of the category **dcpo**. Suppose we are given two *bijective* codings: $<\,,\,>: \omega\times\omega\to\omega$, $e: \omega\to P_{fin}\omega$. We define $\mathrm{Graph}_v: (P\omega\to P\omega)\to P\omega$, and $\mathrm{Fun}_v: P\omega\to(P\omega\to P\omega)$ as follows:

$\mathrm{Graph}_v(f) = \{<n, m> \mid f(e_n)\Downarrow \wedge\ m\in f(e_n)\}$

$\mathrm{Fun}_v(X) = \lambda Y.$ if $Z\neq\varnothing$ then Z else $\Uparrow$,

where: $Z = \{m \mid \exists n.(<n, m>\in X \wedge e_n\subseteq Y)\}$

Verify the $\mathrm{Fun}_v \circ \mathrm{Graph}_v = \mathrm{id}_{(P\omega\to P\omega)}$. We set: $XY \equiv \mathrm{Fun}_v(X)(Y)$, and we denote with $P\omega^v$ the realizability structure resulting from this construction.

Next we expand the definition of $[\![\Omega]\!]^{P\omega^v}$.

$XX \cong \{m \mid \exists n.(<n, m>\in X \wedge e_n\subseteq X)\}$ if not-empty, $\Uparrow$ o.w.

For $\omega \equiv \lambda x.xx$, $[\![\omega]\!]^{P\omega^v} = \mathrm{Graph}_v(\lambda X.\, XX) = \{<n, m> \mid e_ne_n\Downarrow \wedge\ m\in e_ne_n\} =$
$\{<n, m> \mid m\in e_ne_n\} = \{<n, m> \mid \exists x.(<x, m>\in e_n \wedge e_x\subseteq e_n)\}$.

Hence $[\![\Omega]\!]^{P\omega^v} \cong \{z \mid \exists w.(<w, z>\in [\![\omega]\!]^{P\omega^v} \wedge e_w\subseteq [\![\omega]\!]^{P\omega^v})\}$ if not-empty, $\Uparrow$ o.w.

Let us now consider codings[2] such that:

$<1, 0> = 0$, and $e_1 = \{0\}$.

We claim: $0\in [\![\Omega]\!]^{P\omega^v}$. Take w=1. Then:

(i) $<1, 0>\in [\![\omega]\!]^{P\omega^v}$ iff $\exists x.(<x, 0>\in e_1 \wedge e_x\subseteq e_1)$.

But take x=1 and we have: $(<1, 0> = 0\in e_1 \wedge e_1\subseteq e_1)$.

(ii) $e_1\subseteq [\![\omega]\!]^{P\omega^v}$ iff $0\in e_1e_1$ iff $\exists n.(<n, 0>\in e_1 \wedge e_n\subseteq e_1)$.

But take n=1 and we have: $(<1, 0>\in e_1 \wedge e_1\subseteq e_1)$.

Conclusion, for such codings: $[\![\Omega]\!]^{P\omega^v}\Downarrow$. □

[2] Such codings are effective and can be easily built.

Observable Sequentiality

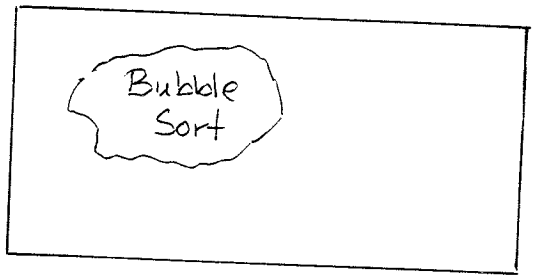
and

Full Abstraction for PCF

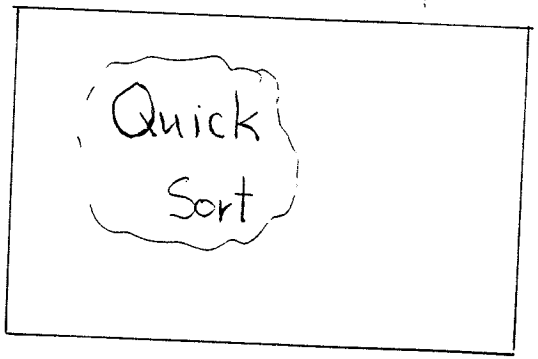
Robert Cartwright & Matthias Felleisen

Rice University

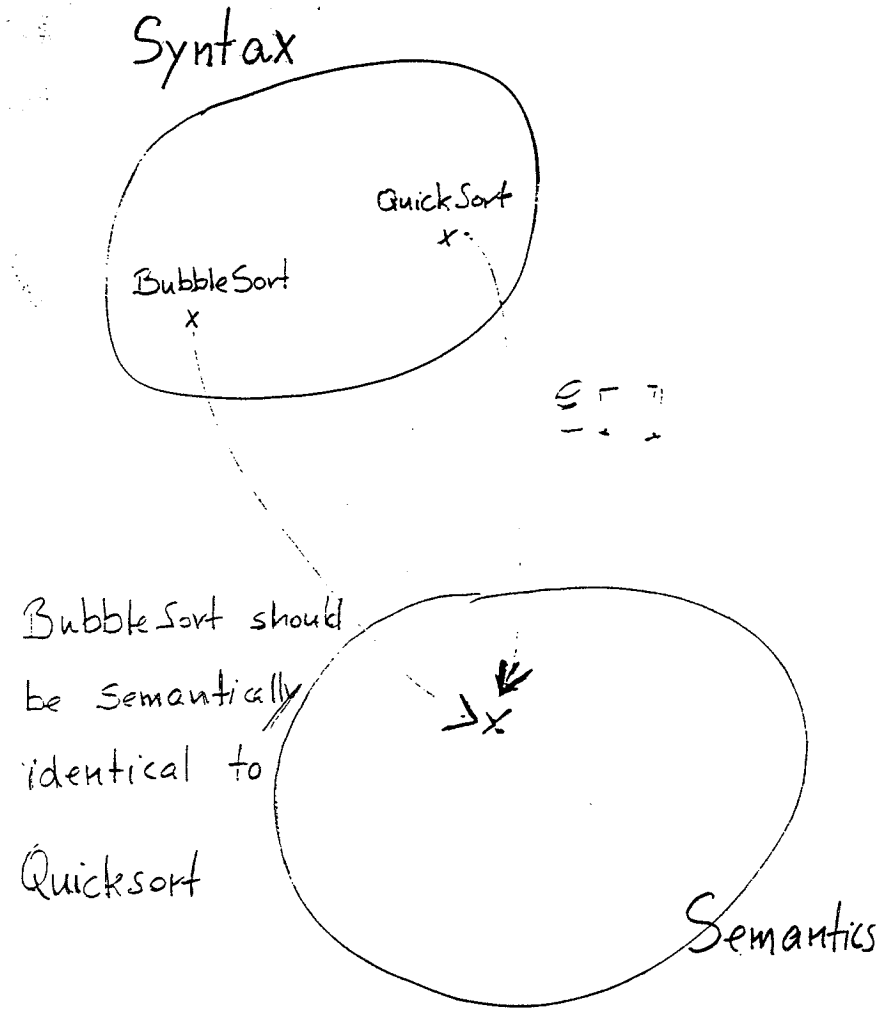
Program 1:



Program 2:



BubbleSort is observationally equivalent to Quicksort



The Full Abstraction Problem

Find a mathematical semantics for sequential programming languages that identifies all observationally equivalent program phrases.

I.

PCF:

The Essence of Sequentiality

Syntax

Types: $t ::= \text{int} \mid \underbrace{t \to t \to \ldots \to t}_{n \geq 1}$

Terms: $M ::=$

$k_n \mid x^t \mid$

$\text{add1} \mid (\lambda x^t.\, M) \mid$

$\text{sub1} \mid (M\ M) \mid$

$\text{if0} \mid Y^{(t \to t) \to t}$

for all t

Semantics

Types: $[\![\text{int}]\!] = \mathbb{N}_\perp$; $[\![s \to t]\!] = [\![s]\!] \to_c [\![t]\!]$

Terms: $P ::= x \mid S \mid K \mid (P\ P) \mid Y$

$\mid k_n \mid \text{add1} \mid \text{sub1} \mid \text{if0}$

$\lambda^* : (x, M) \longmapsto P$

$[\![S]\!](x, y, z) = (x(z))(y(z))$

$[\![K]\!](x, y) = x$

Full Abstraction

(for all programs $\mathcal{P}$,

$$[\![\mathcal{P}(M)]\!] = [\![\mathcal{P}(N)]\!])$$

if and only if

$$[\![M]\!] = [\![N]\!]$$

Example: Left+ ≡ Right+

Left+ $\stackrel{\text{df}}{=}$ $\lambda xy.$ (ifo x y (add1 (Left+ (sub1 x) y)))

Right+ $\stackrel{\text{df}}{=}$ $\lambda xy.$ (ifo y x (add1 (Right+ x (sub1 y)))

And the lack thereof

$P_n = \lambda p^{\text{int} \to \text{int} \to \text{int}}.$

if $p(\Omega, 1)$ then Ω

if $p(1, \Omega)$ then Ω

if $p(0, 0)$ then n

else Ω

for all $k \neq m$, for all programs R,

$$[\![R(P_k)]\!] = [\![R(P_m)]\!]$$

but

$$[\![P_k]\!]_s(\curvearrowright_0) = k \neq m = [\![P_m]\!]_s(\curvearrowright_0)$$

PCF is Sequential

Assume

1) $[\![P(M_1, \dots, M_m)]\!]_\phi = n \in \mathbb{N}$

2) $[\![P(\Omega, \dots, \Omega)]\!]_\phi = \bot$

Then $\exists i,\ 1 \leq i \leq m$

$$[\![P(M_1, \dots, M_{i-1}, \Omega, M_{i+1}, \dots, M_m)]\!]_\rho = \bot$$

↑
position i

A Brief History

1972	Vuillemin	Sequentiality
1977	Milner	Quotient Models
1977	Plotkin	PCF & Parallelism
1978	Kahn – Plotkin	Concrete Data Structures & S
1979	Berry	Stable Functions
1982	Berry – Curien	Sequential Algorithms
1986	Mulmuley	Recasting Quotients via T
1991	Jim – Meyer	Stable Functions cannot solve the problem

II.

Making PCF Realistic

Thesis 1: PCF lacks erro

$$[\![\text{sub1}(0)]\!]\rho \stackrel{?}{=} \bot$$

$$[\![\text{Left-+}]\!]\rho \stackrel{?}{=}$$

$$[\![\text{Right-+}]\!]\rho$$

A Programmer can Observe Sequentiality

$$[\![\text{Left-+}(\underline{\text{error}_1}, \underline{\text{error}_2})]\!]\rho = [\![\underline{\text{error}_1}]\!]\rho$$

$$[\![\text{Right-+}(\underline{\text{error}_1}, \underline{\text{error}_2})]\!]\rho = [\![\underline{\text{error}_2}]\!]\rho$$

Thesis 2: PCF lacks control

$$\Pi = \lambda l.\ \boxed{\text{catch}\ (\lambda\varepsilon.}$$
$$Y(\lambda L.\ (\lambda l.$$
$$(\text{cond}$$
$$((\text{null?}\ l)\quad 1)$$
$$((\text{zero?}\ (\text{car}\ l))\quad (\text{throw}\ \varepsilon$$
$$(t\ (*\ (\text{car}\ l)$$
$$(L\ (\text{cdr}\ l)))))))$$

$$(\Pi\ '(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 0)) \hookrightarrow 0$$

A Program can Observe Sequentiality

$$[\![(\text{catch}\ (\lambda x.\ \boxed{\text{Left+}}\ (x\ 0)\ (x\ 1)))]\!]\rho = 0$$

$$[\![(\text{catch}\ (\lambda x.\ \boxed{\text{Right+}}\ (x\ 0)\ (x\ 1)))]\!]\rho = 1$$

Conclusion:

In "Realistic PCF",
the programmer and
programs can observe the
sequential order of
evaluation(s).

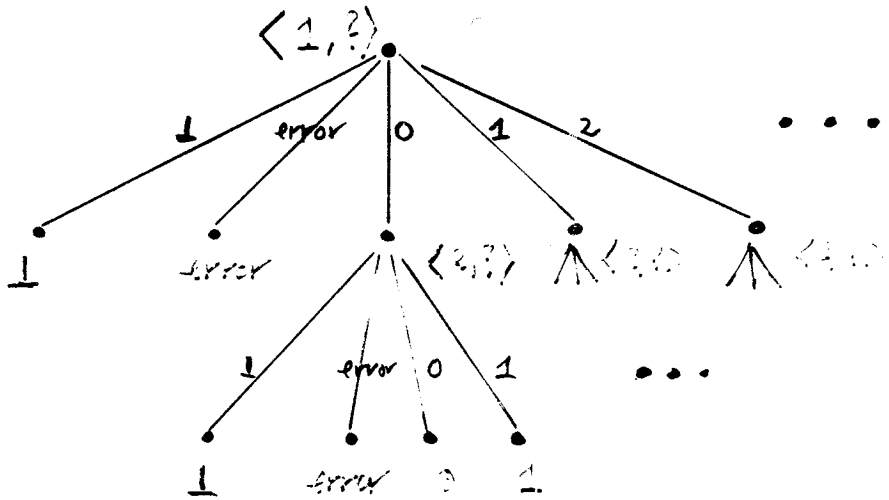
III.

Procedures as Decision Trees

Adding Ordering Information to Functions

$[\![\text{Left}^{-} +]\!]\rho =$

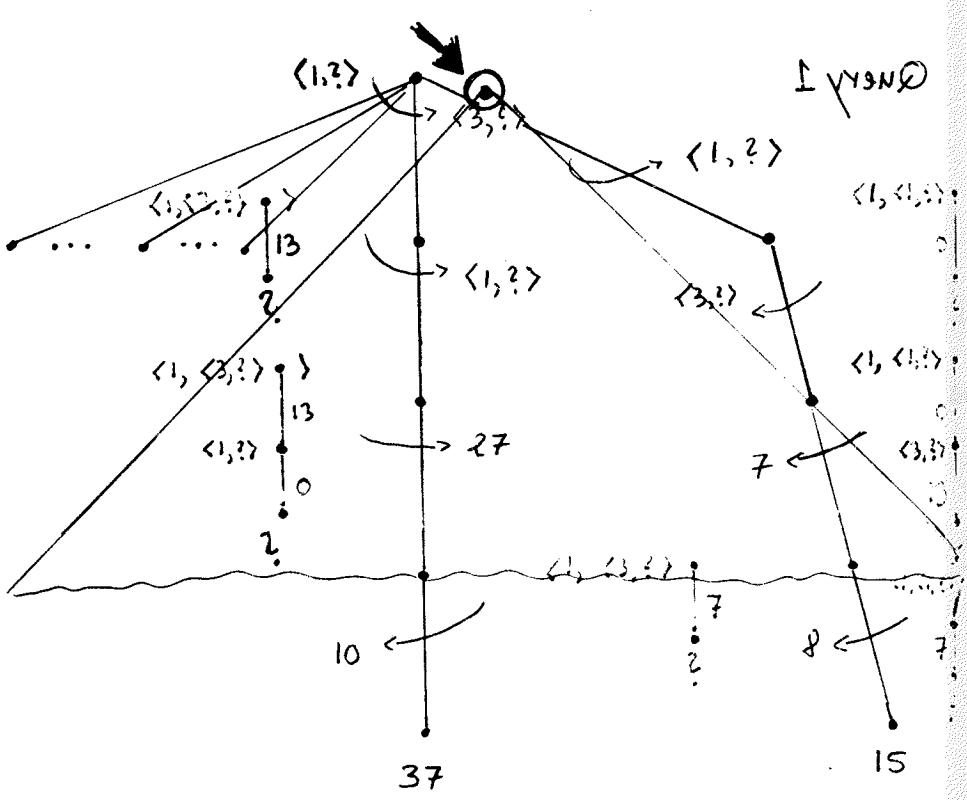
$$\left\langle \begin{matrix} \uparrow \\ -1 \end{matrix}, \begin{cases} \bot \longmapsto \bot \\ \text{error}_i \longmapsto \text{error}_i \\ \\ \\ 0 \longmapsto \langle \lambda \begin{cases} \bot \longmapsto \bot \\ \text{error}_i \longmapsto \text{error}_i \\ 0 \longmapsto 0 \\ 1 \longmapsto 1 \\ \vdots \end{cases} \\ \\ \vdots \end{cases} \right.$$

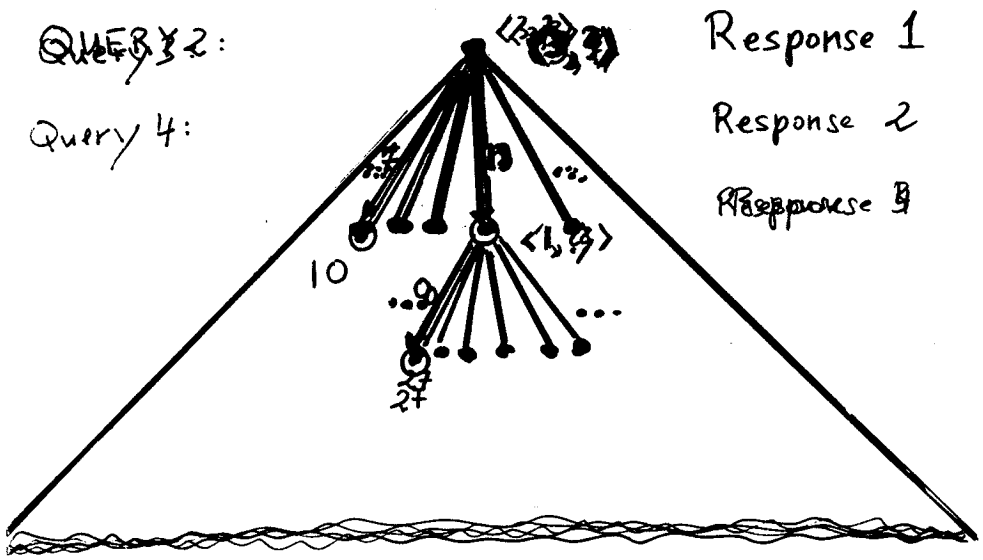


Decision Trees

Representing Higher-Order Procedures

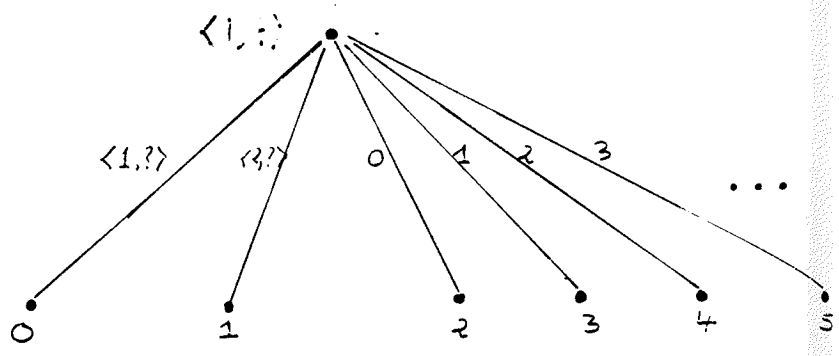
$(\lambda f^{i\times i\to i} : (\text{LEFT} + (f\ 0\ 0\ 13)\ (f\ 0\ 1\ 7)))$





Exploiting Tree Structures

catch: $((\text{int} \times \text{int}) \to \text{int}) \to \text{int}$



IV.

The Results:

Fully Abstract Models

For Sequential Extensions

of PCF

The Model:

1. Fix a set of error values: $\mathbb{E} \supseteq \emptyset$

2. Build decision tree domains over $\mathbb{N}_\perp \cup \mathbb{E}$

3. Define

$$\text{apply}: \mathbb{T}^{\pm \to s} \times \mathbb{T}^{\pm} \longrightarrow \mathbb{T}^{s}$$

4. Define

S; K; add1; suc1; ifc & catch

The Model (Revisited)

(Curien)

1. Fix a set of error values: $\mathbb{E} \supseteq \emptyset$

2. Build Concrete Data Structures (Domain) (over $\mathbb{N}_\perp \cup \mathbb{E}$)

3. Define application, currying, pairing, projecting

4. Prove The above forms a CCC!

(for all sets $\mathbb{E}$, incl. $\mathbb{E}=\emptyset$)

The Theorem:

Let $|\mathrm{Error}| \geq \emptyset$.

The decision tree model over Error is fully abstract for PCF + catch + Error.

- $|\mathrm{Error}| = \emptyset$: intensional model $\sim$ CDS + SA
- $|\mathrm{Error}| = 1$: extensional model
- $|\mathrm{Error}| \geq 2$: order-extensional model

$$f \sqsubseteq g \iff (\forall x)\ \mathrm{apply}(f,x) \sqsubseteq \mathrm{apply}(g,x)$$

Proof Ideas: (1)

$|\mathrm{Error}| \geq 1$:

$$f \sqsubseteq g \overset{\text{Prop.}}{\iff} (\forall F \in \mathbb{D}^{\tau\to o}_{\text{finite}} : (Ff) \sqsubseteq (Fg))$$

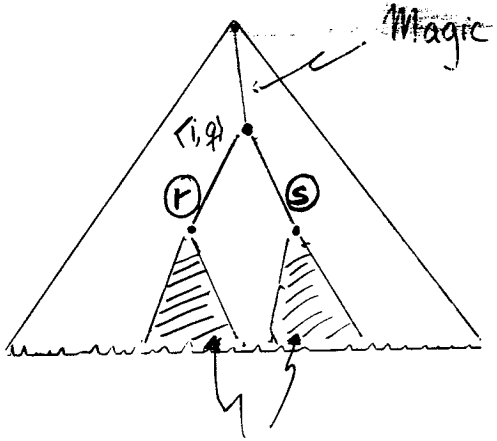
for $f, g \in \mathbb{D}^{\tau}$

$|\mathrm{Error}| > 1$:

$$f \sqsubseteq g \overset{\text{Prop.}}{\iff} \forall x \in \mathbb{D}^{\tau}_{\text{finite}} : \mathrm{apply}(f,x) \sqsubseteq \mathrm{apply}(g,x)$$

for $f, g \in \mathbb{D}^{\tau\to\delta}$

(2) $\tau = \tau_1 \to \cdots \to \tau_n \to o$



$r = q[?/\langle i, q'\rangle]$

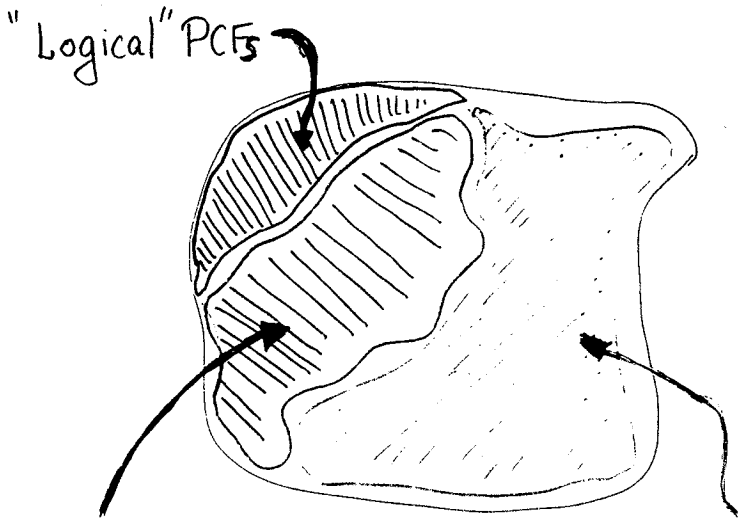
$s = q[?/\langle m_i, q''\rangle]$

$\to \lambda x_1 \cdots x_n.\ \underline{\underline{\ }}\ (x_i\ A_1\ A_2)\ \underline{\underline{\ }}$
(↑ ↑ function of q)

$\to \lambda x_1 \cdots x_n. \text{—} \lambda rs.\ (x_i\ A_1(r)\ A_2(s)) \text{—}$

$\to \lambda x_1 \cdots x_n. \text{—} \boxed{\text{catch}}\ (\lambda rs.\ (x_i\ A_1(r)\ A_2(s))''')$

Perspective



Error-Sensitive

Observably-Sequential

PCFs

PCFs w/
[illegible]
↑
[illegible]

PCF + Control Delimiters

$$[\![(\mathcal{F}\ \mathrm{error}_i)]\!] = [\![i]\!]$$

$$\mathrm{sym}+ \stackrel{df}{=} \lambda xy.\ (\underline{\mathrm{left}+}\ (\mathcal{F}\ x)\ (\mathcal{F}\ y))$$

$$= \lambda xy.\ (\underline{\mathrm{Right}+}\ (\mathcal{F}\ x)\ (\mathcal{F}\ y))$$

sym+ is error-insensitive

Puzzle: Define sym+ such that

error = (sym+ error *) = (sym+ * error

yet s.t. sym+ does not reveal order of evaluation

COMPUTATIONAL COMONADS and INTENSIONAL SEMANTICS

Stephen Brookes
Shai Geva

Carnegie Mellon University
School of Computer Science
Pittsburgh, Pa 15213

EXTENSIONAL SEMANTICS

Examples

- state transformation semantics for while-programs
- Scott model of simply typed lambda calculus

Advantages

- ignores computation details
- models only input-output behavior
- supports reasoning about extensional properties
  - Hoare logic for partial correctness
  - LCF
  - correctness-preserving program transformations

Disadvantages

- cannot distinguish between programs for the same function
- does not support reasoning about intensional properties

INTENSIONAL SEMANTICS

Examples

- Berry-Curien sequential algorithms on concrete data structures
- Brookes-Geva parallel algorithms on generalized concrete data structures

Advantages

- models computation strategy
- relates sensibly to extensional semantics
- may support reasoning about intensional properties
  - order of evaluation
  - degree of parallelism
  - efficiency-improving program transformations

Disadvantages

- intensional models tend to be more complex
- full abstraction only if can observe computation strategy

RESEARCH AIMS

Develop a theory of intensional semantics

- allow a variety of semantic models at differing levels of abstraction
- establish relationships between different models for the same programming language
- show relationship to existing intensional and extensional models
- use intensional semantics to reason about efficiency

THESIS

Category theory provides a general framework:

- datatypes as objects of a category
- comonads can represent notions of computation
- Kleisli categories can serve as intensional models
- intensional meanings as morphisms of Kleisli category
- algorithm = function + computation strategy
- recover extensional model from intensional by taking quotient
- weak or lax forms of cartesian closure seem appropriate in intensional categories

THIS TALK

- comonads and Kleisli categories
- examples of comonads for computation over domains
  - increasing sequences
  - strictly increasing sequences
  - timed data
- Kleisli categories of "parallel algorithms"
- computational comonads and extensional quotients
- computational pairing and lax exponentiation
- conditions for preservation of (weak or lax) cartesian closure
- generalization of earlier models (Berry-Curien, Brookes-Geva)

COMONADS

A *comonad* over a category $\mathcal{C}$ is a (co-)triple (T, ϵ, δ) where

- $T : \mathcal{C} \to \mathcal{C}$ is a functor
- $\epsilon : T \dot{\to} I_{\mathcal{C}}$ is a natural transformation
- $\delta : T \dot{\to} T^2$ is a natural transformation

and for every object A the following associativity and identity laws hold:

$$\begin{aligned} T(\delta_A) \circ \delta_A &= \delta_{TA} \circ \delta_A \\ \epsilon_{TA} \circ \delta_A &= T(\epsilon_A) \circ \delta_A = \mathsf{id}_{TA}. \end{aligned}$$

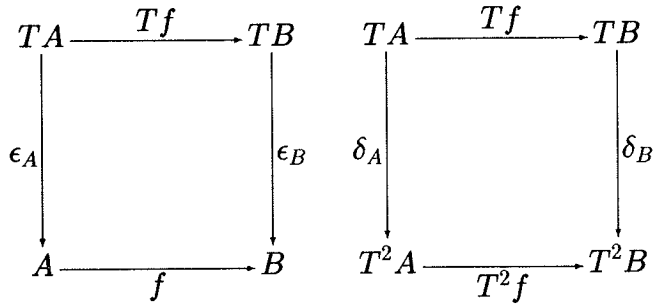
KLEISLI CATEGORIES

Given a comonad (T, ϵ, δ) over $\mathcal{C}$, the *Kleisli category* $\mathcal{C}_T$ is defined by:

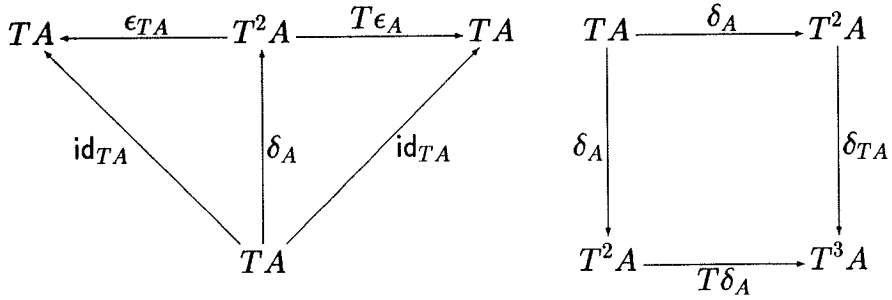
- The objects of $\mathcal{C}_T$ are the objects of $\mathcal{C}$.
- The morphisms from A to B in $\mathcal{C}_T$ are the morphisms from TA to B in $\mathcal{C}$.
- The identity morphism $\widehat{\mathsf{id}}_A$ on A in $\mathcal{C}_T$ is $\epsilon_A : TA \to^{\mathcal{C}} A$.
- The composition in $\mathcal{C}_T$ of $a : A \to^{\mathcal{C}_T} B$ and $a' : B \to^{\mathcal{C}_T} C$ is

$$a' \bar{\circ}\, a = a' \circ Ta \circ \delta_A.$$

COMONAD DIAGRAMS



Naturality



Identity and associativity laws

OPERATIONAL MODEL

- Objects of $\mathcal{C}$ represent datatypes.
- TA represents datatype of computations over A.
- An algorithm from A to B is a morphism from TA to B.
- Algorithms operate in coroutine-like manner (Kahn-MacQueen)
  - execution is lazy, demand-driven
  - algorithm responds to a request for output in B by performing input computation over A until it has enough information to determine what output value to produce
- When a and a' are algorithms from A to B and B to C, $a' \bar{\circ} a$ is an algorithm from A to C, and behaves like a' using a to generate its input.
- A similar, but sequential, operational model is used by Berry and Curien.

COMPUTATIONAL COMONADS

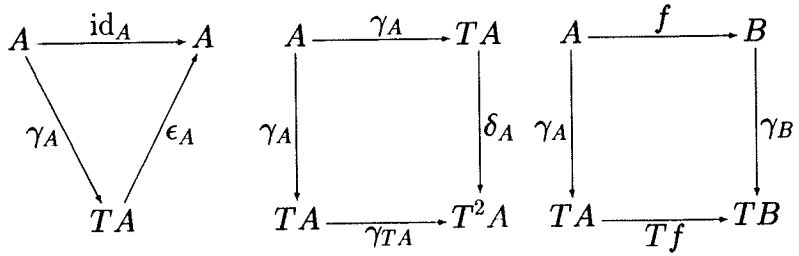
A *computational comonad* over a category $\mathcal{C}$ is a quadruple

$$(T, \epsilon, \delta, \gamma)$$

where (T, ϵ, δ) is a comonad over $\mathcal{C}$ and $\gamma : I_{\mathcal{C}} \dot{\to} T$ is a natural transformation such that, for every object A,

- $\epsilon_A \circ \gamma_A = \mathsf{id}_A$
- $\delta_A \circ \gamma_A = \gamma_{TA} \circ \gamma_A$.

As a corollary, ϵ_A is epi and γ_A is mono, for every object A.



Properties of a computational comonad

RELATING ALGORITHMS AND FUNCTIONS

Let $(T, \epsilon, \delta, \gamma)$ be a computational comonad.

Definition

The functors alg and fun between $\mathcal{C}$ and $\mathcal{C}_T$ are given by:

- $\mathsf{alg} : \mathcal{C} \to \mathcal{C}_T$ is the identity on objects;
- $\mathsf{alg}(f) = f \circ \epsilon_A$, for every $f : A \to^{\mathcal{C}} B$;
- $\mathsf{fun} : \mathcal{C}_T \to \mathcal{C}$ is the identity on objects;
- $\mathsf{fun}(a) = a \circ \gamma_A$, for all $a : A \to^{\mathcal{C}_T} B$.

Terminology

- $\mathsf{fun}(a)$ is the extensional morphism (function) computed by a.
- $\mathsf{alg}(f)$ is a (canonical) algorithm computing f.

SOME PROPERTIES

- fun induces an *extensional equivalence* relation $=^e$ on $\mathcal{C}_T$:

$$a_1 =^e a_2 \iff \mathsf{fun}(a_1) = \mathsf{fun}(a_2).$$

- This relation is a congruence: for all $a_1, a_2 : A \to^{\mathcal{C}_T} B$ and $a_1', a_2' : B \to^{\mathcal{C}_T} C$,

$$a_1 =^e a_2 \;\&\; a_1' =^e a_2' \quad \Rightarrow \quad a_1' \,\bar{\circ}\, a_1 =^e a_2' \,\bar{\circ}\, a_2.$$

- The quotient category of $\mathcal{C}_T$ by $=^e$ is isomorphic to $\mathcal{C}$.
- For all $f : A \to^{\mathcal{C}} B$, $\mathsf{fun}(\mathsf{alg}\ f) = f$.
- For all $a : A \to^{\mathcal{C}_T} B$, $\mathsf{alg}(\mathsf{fun}\ a) =^e a$.
- fun and alg are natural transformations:

$$\mathsf{fun} : \mathcal{C}(T(-), -) \dot{\to} \mathcal{C}(-, -)$$
$$\mathsf{alg} : \mathcal{C}(-, -) \dot{\to} \mathcal{C}(T(-), -).$$

For all $f : A' \to^{\mathcal{C}} A$ and $g : B \to^{\mathcal{C}} B'$,

$$\mathsf{fun} \circ \mathcal{C}(Tf, g) = \mathcal{C}(f, g) \circ \mathsf{fun}$$
$$\mathcal{C}(Tf, g) \circ \mathsf{alg} = \mathsf{alg} \circ \mathcal{C}(f, g).$$

COMPUTATION OVER DOMAINS

Background

- A domain is a directed-complete, bounded-complete, algebraic partial order with a least element.
- The category $\mathcal{C}$ of domains and continuous functions is cartesian closed.

Comonads over domains

We introduce three main examples:

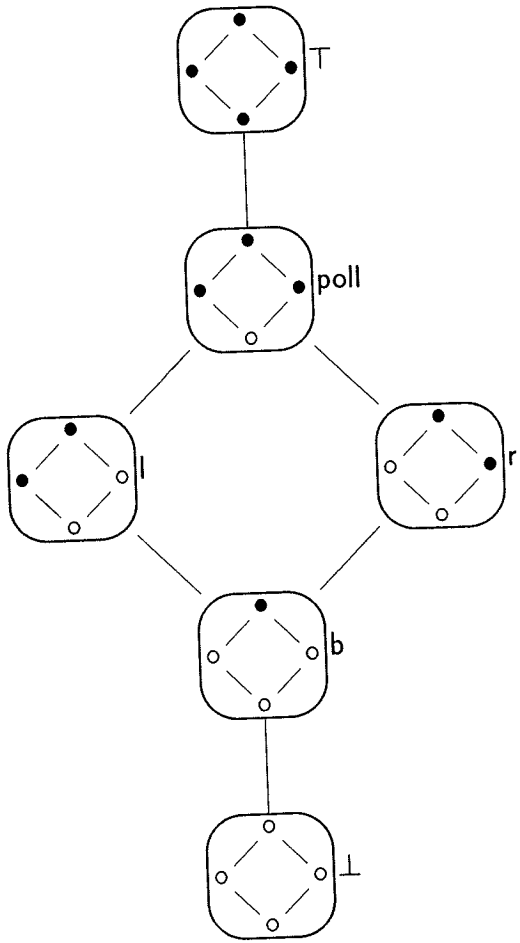
- increasing paths
- strictly increasing paths
- timed data

Generality

These comonads can be adapted to a variety of settings, including:

- stable functions on dI-domains
- sequential functions on concrete domains
- strict continuous functions on domains
- computable functions on effectively given domains

CONTINUOUS FUNCTIONS from 2×2 to 2



Continuous functions from 2×2 to 2, ordered pointwise.

INCREASING PATHS

Definition

- Let $T_1 D$ be the set of increasing sequences over D, ordered componentwise.
- For a continuous function $f : D \to D'$, let $T_1 f : T_1 D \to T_1 D'$ be the function that applies f componentwise:

$$(T_1 f)\langle d_n \rangle_{n=0}^{\infty} = \langle f d_n \rangle_{n=0}^{\infty}.$$

- For $t \in T_1 D$ let $\epsilon_D t$ be the least upper bound of t.
- For $t = \langle d_n \rangle_{n=0}^{\infty}$ let $\delta_D t$ be the sequence of prefixes of t: for all $k \geq 0$,

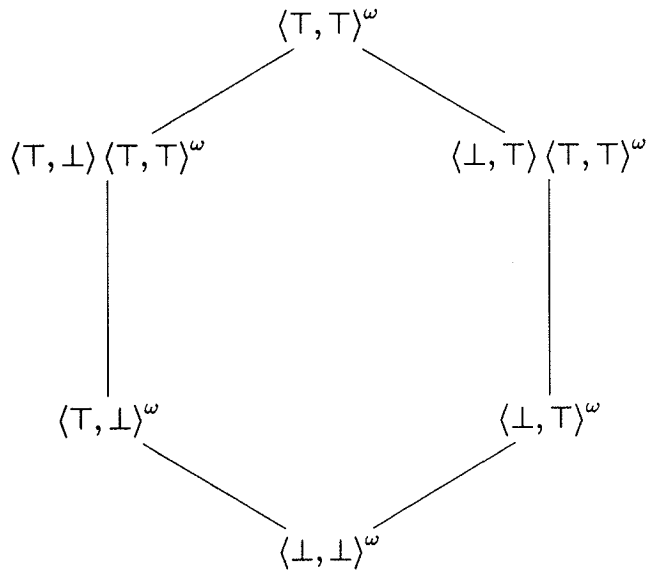
$$(\delta t)_k = d_0 \ldots d_k d_k^{\omega}.$$

- For $d \in D$ let $\gamma d = d^{\omega}$.

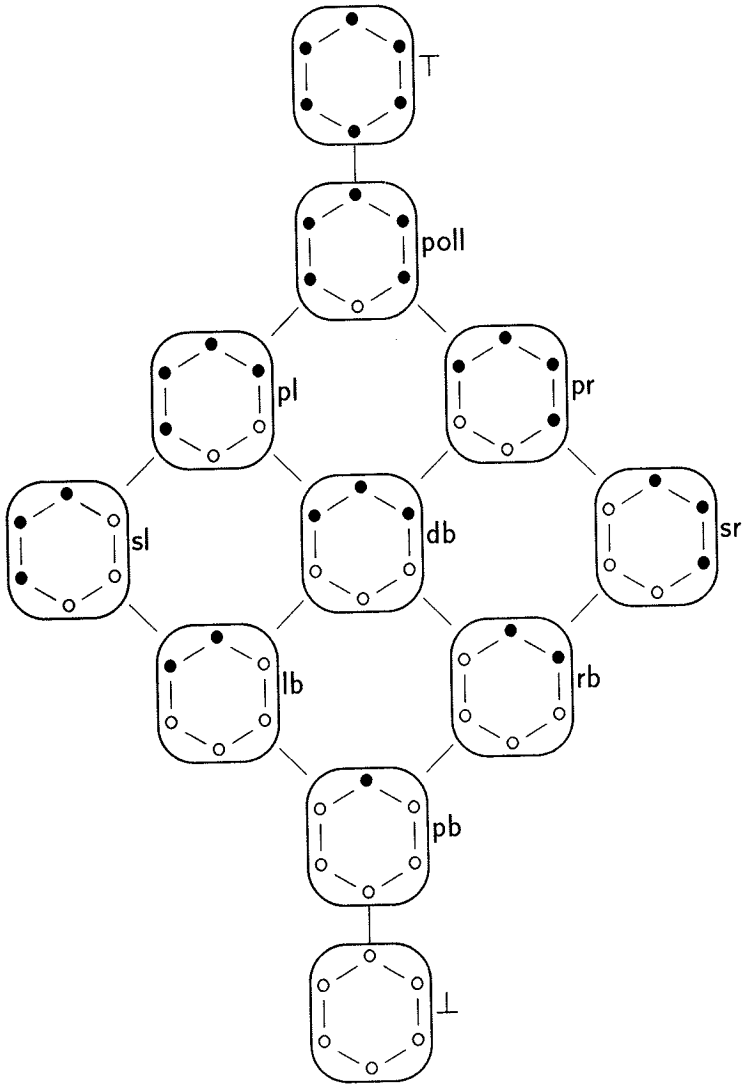
Properties

- A computation in $T_1 D$ records a sequence of incremental steps whose limit is a value in D.
- Idle steps are permitted.
- Every computation is the limit of its prefixes.
- The finite elements of $T_1 D$ are the eventually constant sequences built of finite elements of D.

PART OF $T_1(2 \times 2)$



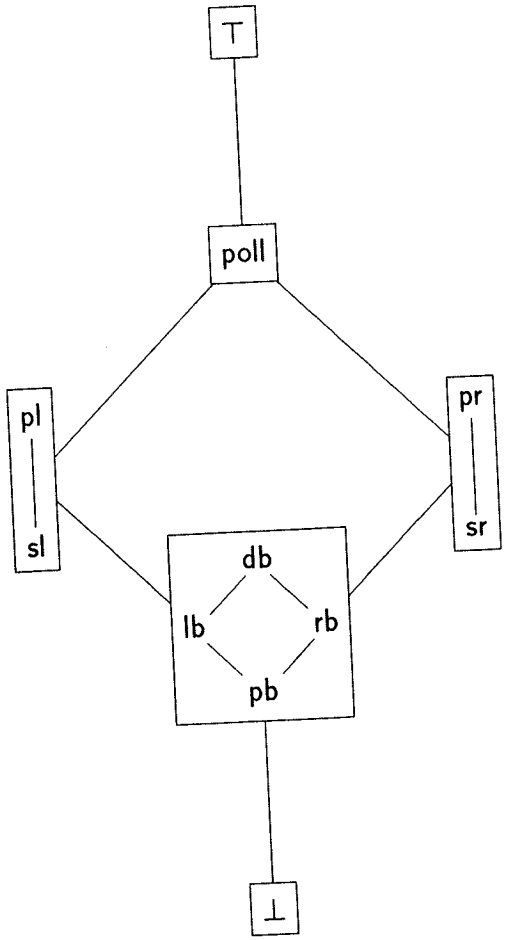
SOME T_1 ALGORITHMS from 2×2 to 2



NOMENCLATURE

- Each algorithm represented by Hasse diagram, and maps solid points to $\top$.
- Algorithm names indicate the function computed and the computation strategy used.
- For instance, the algorithms pb, lb, rb and db all compute the function b:
  - pb computes both arguments in parallel immediately
  - lb computes left first and then right
  - rb computes right first and then left
  - db computes both arguments in either order.
- pb is also characterized as the least continuous function on paths satisfying
$$\mathsf{pb}(\langle \top, \top \rangle^{\omega}) = \top.$$
- Since the diagram includes only one algorithm for poll, for $\bot$, and for $\top$, in these cases we use the same name for the algorithm as for the function.

EXTENSIONAL QUOTIENT of T_1 ALGORITHMS



MORE T_1 ALGORITHMS

$T_1(2 \times 2)$ includes paths with repeated steps, so we can make distinctions between algorithms which differ in the amount of time they allow between successive increments.

- For the function b there are algorithms pb_n, lb_n, rb_n and db_n for each $n \geq 0$, characterized as the least functions such that

$$\begin{aligned}
&\mathsf{pb}_n(\langle\bot,\bot\rangle^n\langle\top,\top\rangle^\omega) = \top\\
&\mathsf{lb}_n(\langle\bot,\bot\rangle^n\langle\top,\bot\rangle\langle\top,\bot\rangle^n\langle\top,\top\rangle^\omega) = \top\\
&\mathsf{rb}_n(\langle\bot,\bot\rangle^n\langle\bot,\top\rangle\langle\bot,\top\rangle^n\langle\top.\top\rangle^\omega) = \top\\
&\mathsf{db}_n(\langle\bot,\bot\rangle^n\langle\top,\bot\rangle\langle\top,\bot\rangle^n\langle\top,\top\rangle^\omega) = \top\\
&\mathsf{db}_n(\langle\bot,\bot\rangle^n\langle\bot,\top\rangle\langle\bot,\top\rangle^n\langle\top,\top\rangle^\omega) = \top.
\end{aligned}$$

- pb_n evaluates both arguments and returns $\top$ provided each evaluation succeeds in at most n time steps.

- lb_n evaluates both arguments and returns $\top$ provided evaluation of the left argument succeeds in at most n time steps and evaluation of the right argument succeeds in at most $2n+1$ time steps.

- The following relationships hold, for all $n \geq 0$:

$$\begin{aligned}
&\mathsf{pb}_n \sqsubseteq \mathsf{lb}_n \sqsubseteq \mathsf{pb}_{2n+1}\\
&\mathsf{pb}_n \sqsubseteq \mathsf{rb}_n \sqsubseteq \mathsf{pb}_{2n+1}\\
&\mathsf{db}_n = \mathsf{lb}_n \sqcup \mathsf{rb}_n.
\end{aligned}$$

- Each of these sequences of algorithms has the same least upper bound, the algorithm $\mathsf{alg}(\mathsf{b})$ that maps every path with lub $\langle\top,\top\rangle$ to $\top$.

STRICTLY INCREASING PATHS

- Let T_2D be the set of finite or infinite strictly increasing sequences over D:
  - $\langle d_n \rangle_{n=0}^{\infty}$, with $d_n \sqsubset_D d_{n+1}$ for all all $n \geq 0$, or
  - $d_0 \ldots d_{N-1} d_N^{\omega}$, where $d_n \sqsubset_D d_{n+1}$ for $0 \leq n < N$.

  (We represent a finite sequence as an eventually constant infinite sequence.)
- Let $\sqsubseteq_{T_2D}$ be the least partial order on T_2D such that

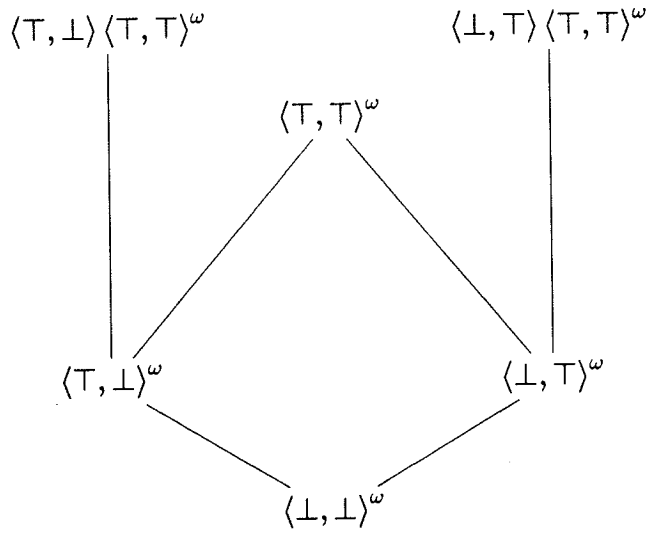
$$d_0 \ldots d_{N-1} d_N^{\omega} \sqsubseteq_{T_2D} d_0 \ldots d_{N-1} t \text{ if } t \in T_2D \;\&\; d_N \sqsubseteq_D t_0.$$

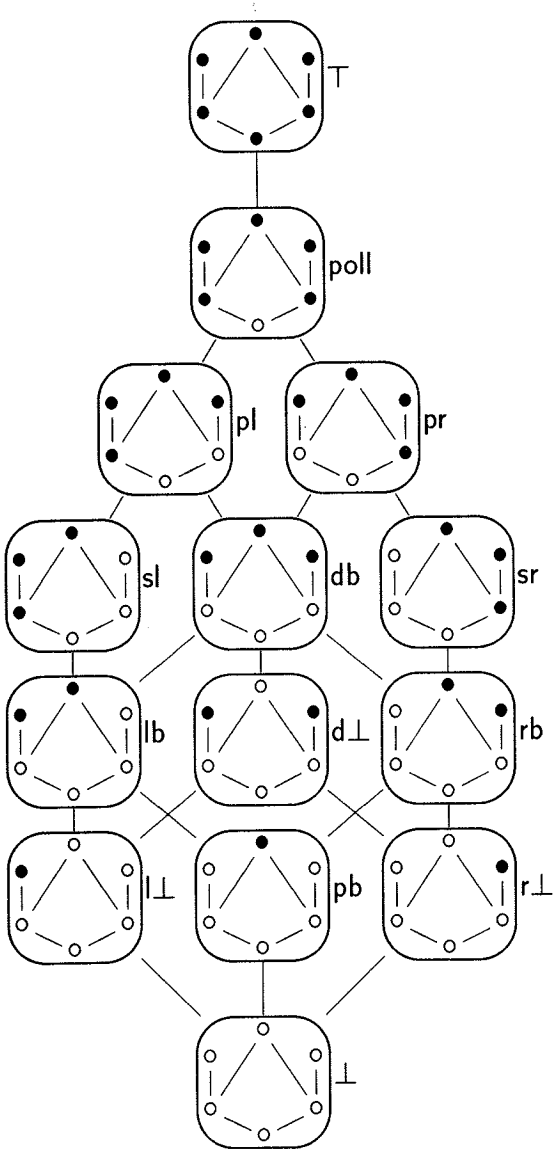
- This is the prefix ordering on sequences, adjusted to take account of the ordering on data values.
- For a continuous function $f : D \to D'$ let T_2f be the function which applies f componentwise and suppresses any repetitions (except for constant suffixes):

$$\begin{array}{rcll} T_2f(d^{\omega}) & = & (fd)^{\omega} & \\ T_2f(d_0 d_1 t) & = & (fd_0)(T_2f(d_1 t)) & \text{if } fd_0 \neq fd_1 \\ & = & T_2f(d_1 t) & \text{if } fd_0 = fd_1. \end{array}$$

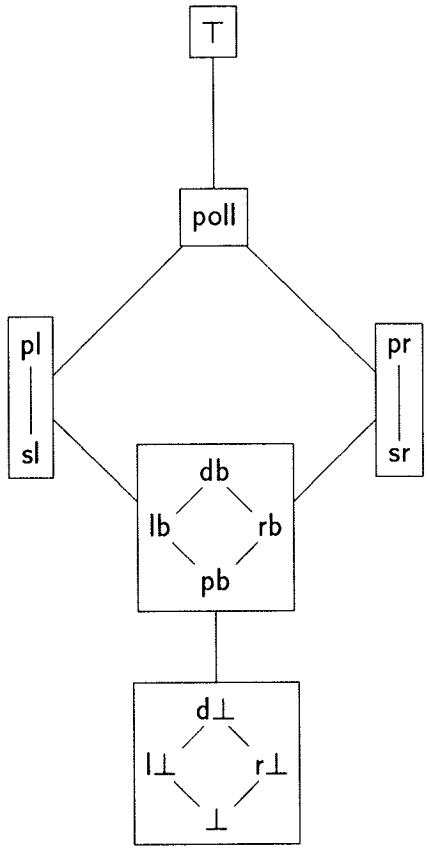
- For all $t \in T_2D$, let $\epsilon_D t$ be the least upper bound of t.
- For all $t \in T_2D$, let $\delta_D t$ be the sequence of prefixes of t.
- For $d \in D$ let $\gamma d = d^{\omega}$.

PART OF $T_2(2 \times 2)$





EXTENSIONAL QUOTIENT of T_2 ALGORITHMS



TIMED DATA

Definition

- $T_3 D = D \times \mathsf{VNat}^{\mathrm{op}}$, ordered componentwise.
- For $f : D \to D'$, $(T_3 f)\,\langle d, n\rangle = \langle f d, n\rangle$.
- $\epsilon\,\langle d, n\rangle = d$.
- $\delta\,\langle d, n\rangle = \langle \langle d, n\rangle, n\rangle$.
- $\gamma d = \langle d, 0\rangle$.

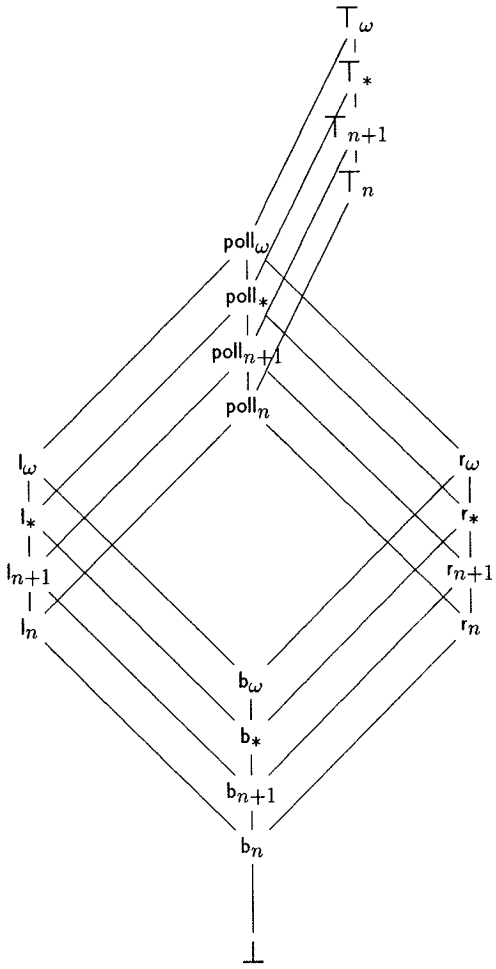
Remarks

- $\mathsf{VNat}^{\mathrm{op}}$ is the "vertical" natural numbers together with ω, in the reverse of the usual order.
- A computation in $T_3 D$ is a data value with a time or cost.
- $\langle d, n\rangle \sqsubseteq \langle d', n'\rangle$ iff $d \sqsubseteq_D d'$ and $n' \leq n$.
- For a continuous function $f : D \to D'$ and $n \in \mathsf{VNat}$ can define an algorithm f_n from D to D':

$$\begin{aligned} f_n\,\langle d, k\rangle &= f d && \text{if } k \leq n \\ &= \bot && \text{otherwise.} \end{aligned}$$

- If $f \sqsubseteq g$ then $f_n \sqsubseteq g_n$.
- For all n, $f_n \sqsubseteq f_{n+1}$.
- f_ω is simply $\mathsf{alg}(f) = \lambda\,\langle d, n\rangle\,.f d$.
- The lub of the f_n is $f_* = \lambda\,\langle d, n\rangle\,.(n = \omega \to \bot, f d)$.

SOME T_3 ALGORITHMS from 2×2 to 2



COMPUTATIONAL PAIRINGS

If $\mathcal{C}$ has distinguished finite products and projections π_i, then $\mathcal{C}_T$ has distinguished finite products, with projections

$$\begin{aligned} \widehat{\pi}_i &: A_1 \times A_2 \to^{\mathcal{C}_T} A_i \\ \widehat{\pi}_i &= \epsilon_{A_i} \circ T\,\pi_i \\ &= \pi_i \circ \epsilon_{A_1 \times A_2}. \end{aligned}$$

From now on, we assume that $\mathcal{C}$ has distinguished finite products.

Definition

Let $(T, \epsilon, \delta, \gamma)$ be a computational comonad.

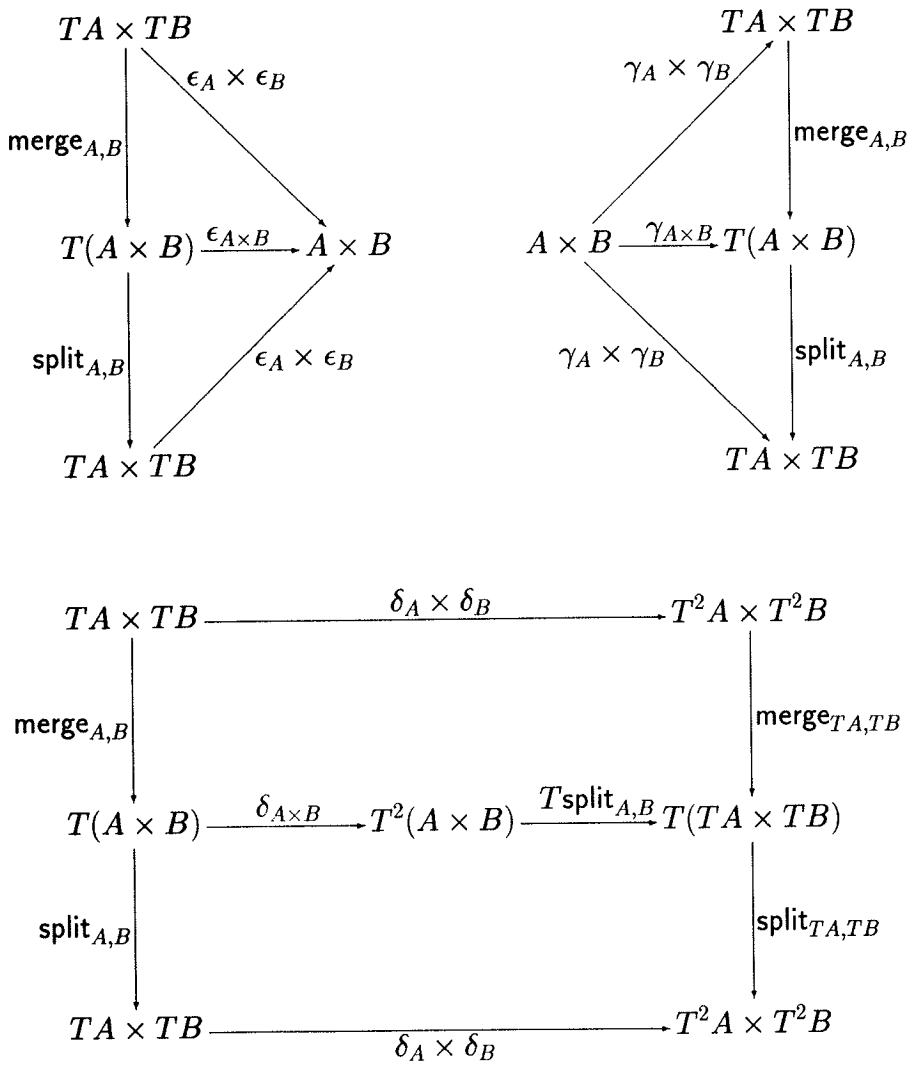
A *computational pairing* is a pair of natural transformations

$$\begin{aligned} &\mathsf{split} : T(- \times -) \stackrel{\cdot}{\to} T(-) \times T(-) \\ &\mathsf{merge} : T(-) \times T(-) \stackrel{\cdot}{\to} T(- \times -) \end{aligned}$$

such that, for all objects A and B, the following properties hold:

$$\begin{aligned} &(\epsilon_A \times \epsilon_B) \circ \mathsf{split}_{A,B} = \epsilon_{A \times B} \\ &\epsilon_{A \times B} \circ \mathsf{merge}_{A,B} = \epsilon_A \times \epsilon_B \\ &\mathsf{split}_{A,B} \circ \gamma_{A \times B} = \gamma_A \times \gamma_B \\ &\mathsf{merge}_{A,B} \circ (\gamma_A \times \gamma_B) = \gamma_{A \times B} \\ &(\delta_A \times \delta_B) \circ \mathsf{split}_{A,B} = \mathsf{split}_{TA,TB} \circ T\mathsf{split}_{A,B} \circ \delta_{A \times B} \\ &\mathsf{merge}_{TA,TB} \circ (\delta_A \times \delta_B) = T\mathsf{split}_{A,B} \circ \delta_{A \times B} \circ \mathsf{merge}_{A,B}. \end{aligned}$$

COMPUTATIONAL PAIRING



COMPUTATIONAL PAIRINGS for T_1

Parallel split and merge

$$\begin{aligned} \mathsf{split}(u) &= \langle \lambda n.\, \pi_1(u_n), \lambda n.\, \pi_2(u_n) \rangle \\ \mathsf{merge}(\langle s, t \rangle) &= \lambda n.\, \langle s_n, t_n \rangle \end{aligned}$$

- These form an isomorphism pair.
- Intuitively, each of these operations works in parallel on the two components.
- The computational pairing laws state that:
  - Merging and splitting commute with componentwise application of functions.
  - Merging and splitting respect lubs.
  - Merging and splitting respect prefixes.
  - Merging two degenerate computations produces a degenerate computation.
  - Splitting a degenerate computation produces a pair of degenerate computations.

COMPUTATIONAL PAIRINGS for T_1

Left-first interleaving

$$\begin{aligned} \mathsf{merge}_l(\langle s, t \rangle) &= \lambda n.\, \left\langle s_{\lceil n/2 \rceil}, t_{\lfloor n/2 \rfloor} \right\rangle \\ \mathsf{split}_2(u) &= \langle \lambda n.\, \pi_1(u_{2n}), \lambda n.\, \pi_2(u_{2n}) \rangle\,. \end{aligned}$$

- These satisfy the identity $\mathsf{split}_2 \circ \mathsf{merge}_l = \mathsf{id}$.
- The computational pairing properties hold, making use of the equalities

$$\begin{aligned} \lfloor \min(i, 2j)/2 \rfloor &= \min(\lfloor i/2 \rfloor, j) \\ \lceil \min(i, 2j)/2 \rceil &= \min(\lceil i/2 \rceil, j). \end{aligned}$$

Right-first interleaving

- Defined similarly.

COMPUTATIONAL PAIRINGS for T_2

- Each of the split-merge combinations above can be modified to ensure that splitting a strictly increasing sequence of pairs gives a pair of strictly increasing sequences.
- However, $T_2(A \times B)$ and $T_2A \times T_2B$ are not generally isomorphic, because a strictly increasing sequence of pairs does not necessarily increase strictly in ***both*** components at each stage.
- Nevertheless, split and merge are still natural transformations satisfying the requirements listed above for a computational pairing, and we have the identity $\mathsf{split} \circ \mathsf{merge} = \mathsf{id}$.

PAIRING of ALGORITHMS

- Using split provides a way to combine a pair of algorithms into an algorithm on pairs.
- When split is taken to be $\langle T\pi_1, T\pi_2 \rangle$ this is the standard way to form the product of two morphisms in the Kleisli category.
- We can also use split to define intensional analogues to the contravariant hom-functors $\mathcal{C}(- \times B, C)$ and $\mathcal{C}(-, B \to C)$.

Definition

Let $\mathcal{C}$ be a cartesian closed category, let $(T, \epsilon, \delta, \gamma)$ be a computational comonad, and let split and merge form a computational pairing. The contravariant functor $\mathcal{C}_T(- \hat{\times} B, C)$ from ${\mathcal{C}_T}^{\mathrm{op}}$ to Set is defined as follows.

- On objects the functor maps A to $\mathcal{C}_T(A \times B, C)$.
- On morphisms the functor maps $f : A \to^{\mathcal{C}_T} A'$ to the function $\lambda g.g \mathbin{\bar{\circ}} ((f \times \widehat{\mathsf{id}_B}) \circ \mathsf{split})$ from $\mathcal{C}_T(A' \times B, C)$ to $\mathcal{C}_T(A \times B, C)$.

Similarly, the contravariant functor $\mathcal{C}_T(-, TB \to C)$ from ${\mathcal{C}_T}^{\mathrm{op}}$ to Set, is defined by:

- On objects the functor maps A to $\mathcal{C}_T(A, TB \to C)$.
- On morphisms the functor maps $f : A \to^{\mathcal{C}_T} A'$ to the function $\lambda h.h \mathbin{\bar{\circ}} f$ from $\mathcal{C}_T(A', TB \to C)$ to $\mathcal{C}_T(A, TB \to C)$.

CURRYING and UNCURRYING of ALGORITHMS

Let $(T, \epsilon, \delta, \gamma)$ be a computational comonad and let split and merge be a computational pairing.

Given $a : T(A \times B) \to^{\mathcal{C}} C$ and $b : TA \to^{\mathcal{C}} (TB \to C)$, define

$\widehat{\mathsf{curry}}(a) : TA \to^{\mathcal{C}} (TB \to C)$ $\qquad$ $\widehat{\mathsf{uncurry}}(b) : T(A \times B) \to^{\mathcal{C}} C$

$\widehat{\mathsf{curry}}(a) = \mathsf{curry}(a \circ \mathsf{merge})$ $\qquad$ $\widehat{\mathsf{uncurry}}(b) = \mathsf{uncurry}(b) \circ \mathsf{split}.$

Then:

- $\widehat{\mathsf{curry}}$ and $\widehat{\mathsf{uncurry}}$ are natural transformations:

$$\widehat{\mathsf{curry}} : \mathcal{C}_T(- \hat{\times} B, C) \dot{\to} \mathcal{C}_T(-, TB \to C)$$
$$\widehat{\mathsf{uncurry}} : \mathcal{C}_T(-, TB \to C) \dot{\to} \mathcal{C}_T(- \hat{\times} B, C).$$

- For all $a : A \times B \to^{\mathcal{C}_T} C$, $\widehat{\mathsf{uncurry}}(\widehat{\mathsf{curry}}(a)) =^e a$.
- For all $f : A \times B \to^{\mathcal{C}} C$, $\widehat{\mathsf{uncurry}}(\widehat{\mathsf{curry}}(\mathsf{alg}\ f)) = \mathsf{alg}\ f$.

Thus computational pairing produces a weak form of exponentiation structure.

APPLICATION on ALGORITHMS

Let $\mathcal{C}$ be a cartesian closed category, $(T, \epsilon, \delta, \gamma)$ be a computational comonad and let split and merge be a computational pairing.

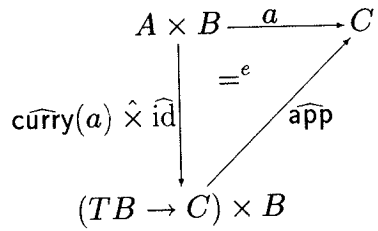
- For all B and C there is an "application morphism"

$$\widehat{\mathsf{app}}_{B,C} : [TB \to C] \times B \to^{\mathcal{C}_T} C$$

such that, for all $a : A \times B \to^{\mathcal{C}_T} C$

$$\widehat{\mathsf{app}}_{B,C} \,\bar{\circ}\, (\widehat{\mathsf{curry}}(a) \,\hat{\times}\, \widehat{\mathsf{id}}_B) =^e a.$$

- Note that although $\widehat{\mathsf{curry}}(a)$ is not the unique morphism h such that $\widehat{\mathsf{app}} \,\bar{\circ}\, (h \,\hat{\times}\, \widehat{\mathsf{id}}) =^e a$, all such morphisms satisfy the condition that $\widehat{\mathsf{uncurry}}(h) =^e a$.
- Instead of the usual diagram for exponentiation we replace $=$ by $=^e$ and we relax the uniqueness condition:



SOME CONSEQUENCES

- Our definitions are parameterized by the choice of **split** and **merge**. Once these are chosen, $\widehat{\mathsf{app}}$, $\widehat{\mathsf{curry}}$ and $\widehat{\mathsf{uncurry}}$ are determined uniquely.
- The Kleisli category itself is independent of **split** and **merge**.
- Each choice of computational pairing induces a (weak form of) exponentiation structure on this category.
- A Kleisli category may possess many different notions of merging and splitting, and therefore many different ways to curry, uncurry and apply algorithms.
- This suggests that one may use a Kleisli category to give an interpretation to a functional programming language containing several syntactically and semantically distinct forms of application.
- This might be desirable, for instance, if the language included both a strict and a non-strict form of application.

EXAMPLES, REVISITED

- The Kleisli category based on T_1 is cartesian closed, with exponentiation built from the standard split-merge combination.
- The Kleisli category of T_1 has a weak exponentiation derived from merge_l and split_2. This gives intensional forms of currying, uncurrying, and application which we call $\widehat{\mathsf{curry}}_l$, $\widehat{\mathsf{uncurry}}_l$ and $\widehat{\mathsf{app}}_l$. For all $a : A \times B \to^{\mathcal{C}_{T_1}} C$,

$$\widehat{\mathsf{app}}_l \circ (\widehat{\mathsf{curry}}_l(a) \hat{\times} \widehat{\mathsf{id}}) =^e a.$$

Since $\mathsf{split}_2 \circ \mathsf{merge}_l = \mathsf{id}$, we have

$$\begin{aligned} \widehat{\mathsf{curry}}_l(\widehat{\mathsf{uncurry}}_l\ h) &= h \\ \widehat{\mathsf{uncurry}}_l(\widehat{\mathsf{curry}}_l\ g) &=^e g. \end{aligned}$$

For example,

$$\widehat{\mathsf{uncurry}}_l(\widehat{\mathsf{curry}}_l\ \mathsf{pb}) = \widehat{\mathsf{uncurry}}_l(\widehat{\mathsf{curry}}_l\ \mathsf{lb}) = \mathsf{pb},$$

and $\widehat{\mathsf{uncurry}}_l(\widehat{\mathsf{curry}}_l\ \mathsf{rb})$ maps the path $\langle\bot,\top\rangle\langle\bot,\top\rangle\langle\top,\top\rangle^\omega$ to $\top$. This algorithm of course computes the function b.

- The Kleisli category of T_2 has a weak exponentiation based on the computational pairing **split** and **merge**. In this case,

$$\begin{aligned} \widehat{\mathsf{curry}}(\widehat{\mathsf{uncurry}}\ h) &= h \\ \widehat{\mathsf{uncurry}}(\widehat{\mathsf{curry}}\ g) &=^e g. \end{aligned}$$

For example,

$$\widehat{\mathsf{uncurry}}(\widehat{\mathsf{curry}}\ \mathsf{lb}) = \widehat{\mathsf{uncurry}}(\widehat{\mathsf{curry}}\ \mathrm{rb}) = \widehat{\mathsf{uncurry}}(\widehat{\mathsf{curry}}\ \mathrm{pb}) = \mathrm{pb}.$$

The merge_l and split_2 computational pairing also gives rise to a weak form of exponentiation for T_2.

ORDERED CATEGORIES

Now suppose that $\mathcal{C}$ is an ordered category:

- each hom-set is equipped with a complete partial order
- composition is continuous
- functors are required to act continuously on morphisms

All of our examples so far satisfy these conditions.

Suppose $(T, \epsilon, \delta, \gamma)$ is a computational comonad over an ordered category $\mathcal{C}$.

- $\mathcal{C}_T$ is again an ordered category.
- All of the results of the previous sections go through in the ordered setting. In particular:
  - The functors fun and alg respect the ordering;
  - The extensional quotient of the ordering on $\mathcal{C}_T(A, B)$ is just the ordering on $\mathcal{C}(A, B)$.

UP- and DOWN-CLOSED CATEGORIES

Definition

An ordered category $\mathcal{C}$ is *cartesian up-closed* if and only if it has finite products and for all pairs of objects B and C there is an object $B \to C$ and a pair of lax natural transformations curry, $\mathsf{uncurry}$ between $\mathcal{C}(- \times B, C)$ and $\mathcal{C}(-, B \to C)$ satisfying:

$$\begin{aligned} \mathsf{curry}(\mathsf{uncurry} h) &\leq h \\ \mathsf{uncurry}(\mathsf{curry} g) &\geq g \\ \mathsf{curry}(g \circ (f \times \mathsf{id})) &\leq \mathsf{curry}(g) \circ f \\ \mathsf{uncurry}(h) \circ (f \times \mathsf{id}) &\geq \mathsf{uncurry}(h \circ f). \end{aligned}$$

Similarly, we say that $\mathcal{C}$ is *cartesian down-closed* iff it has finite products and there is a pair of lax natural transformations curry, $\mathsf{uncurry}$ satisfying:

$$\begin{aligned} \mathsf{curry}(\mathsf{uncurry}\ h) &\geq h \\ \mathsf{uncurry}(\mathsf{curry}\ g) &\leq g \\ \mathsf{curry}(g \circ (f \times \mathsf{id})) &\geq \mathsf{curry}(g) \circ f \\ \mathsf{uncurry}(h) \circ (f \times \mathsf{id}) &\leq \mathsf{uncurry}(h \circ f). \end{aligned}$$

UP- and DOWN-EXPONENTIATION

Let $\mathcal{C}$ be an ordered category with finite products.

Definition

An *up-exponential* for objects B and C is an object $B \to C$ of $\mathcal{C}$ together with a morphism $\mathsf{app}_{B,C} : (B \to C) \times B \to^{\mathcal{C}} C$ such that for every $f : A \times B \to^{\mathcal{C}} C$ there is a least morphism $\mathsf{curry}(f) : A \to^{\mathcal{C}} (B \to C)$ such that

$$\mathsf{app} \circ (\mathsf{curry}(f) \times \mathsf{id}) \geq \mathrm{f}.$$

A *down-exponential* for objects B and C is an object $B \to C$ of $\mathcal{C}$ together with a morphism $\mathsf{app}_{B,C} : (B \to C) \times B \to^{\mathcal{C}} C$ such that for every $f : A \times B \to^{\mathcal{C}} C$ there is a greatest morphism $\mathsf{curry}(f) : A \to^{\mathcal{C}} (B \to C)$ such that

$$\mathsf{app} \circ (\mathsf{curry}(f) \times \mathsf{id}) \leq \mathrm{f}.$$

SOME PROPERTIES

- An ordered category is cartesian up-closed iff it has finite products and up-exponentials.
- An ordered category is cartesian down-closed iff it has finite products and down-exponentials.
- If the same object $B \to C$ and morphism $\mathsf{app}_{B,C}$ qualifies simultaneously as an up- and a down-exponential then it forms the usual notion of exponentiation and the category is cartesian closed.

Theorem

Let $\mathcal{C}$ be a cartesian up-closed category, let $(T, \epsilon, \delta, \gamma)$ be a computational comonad, and let split and merge be a computational pairing such that

$$\begin{aligned}
&\mathsf{split} \circ \mathsf{merge} \leq \mathsf{id} \\
&\mathsf{merge} \circ \mathsf{split} \geq \mathsf{id} \\
&(\delta \times \delta) \circ \mathsf{split} \leq \mathsf{split} \circ T\mathsf{split} \circ \delta \\
&\mathsf{merge} \circ (\delta \times \delta) \geq T\mathsf{split} \circ \delta \circ \mathsf{merge}.
\end{aligned}$$

Then $\mathcal{C}_T$ is cartesian up-closed.

A similar result holds for a cartesian down-closed category with a computational pairing satisfying reversed inequalities.

T_3 REVISITED

- In the timed data comonad T_3, the standard split operation is:

$$\mathsf{split}\,\langle\langle a, b\rangle, n\rangle = \langle\langle a, n\rangle, \langle b, n\rangle\rangle\,.$$

- An obvious merge operation is:

$$\mathsf{merge}(\langle\langle a, m\rangle, \langle b, n\rangle\rangle) = \langle\langle a, b\rangle, \max(m, n)\rangle\,.$$

- These operations define natural transformations, and:

$$\begin{aligned}
&\mathsf{split} \circ \mathsf{merge} \sqsubseteq \mathsf{id}\\
&\mathsf{merge} \circ \mathsf{split} = \mathsf{id}\\
&(\delta \times \delta) \circ \mathsf{split} = \mathsf{split} \circ T_3\mathsf{split} \circ \delta\\
&\mathsf{merge} \circ (\delta \times \delta) \sqsupseteq T_3\mathsf{split} \circ \delta \circ \mathsf{merge}.
\end{aligned}$$

- The underlying category is cartesian closed, hence also cartesian up-closed.
- Hence the Kleisli category of T_3 is cartesian up-closed.

STRICT FUNCTIONS and ALGORITHMS

- The category of domains and strict continuous functions has products but is not cartesian closed.
- The usual uncurrying operation on functions preserves strictness, but the usual currying does not. Instead, we define a variant form of currying by:

$$\begin{aligned}
&\mathsf{curry}_s : (A \times B \to_s C) \to (A \to_s (B \to_s C))\\
&\mathsf{curry}_s(f) = \lambda x.\lambda y.(x = \bot \vee y = \bot \to \bot, f(x, y)).
\end{aligned}$$

- When f is strict, $\mathsf{curry}_s(f)$ is the best strict function approximating $\mathsf{curry}(f)$ pointwise.
- For all $f : A \times B \to_s C$ and all $g : A \to_s (B \to_s C)$:

$$\begin{aligned}
&\mathsf{curry}_s(\mathsf{uncurry}\ g) = g\\
&\mathsf{uncurry}(\mathsf{curry}_s\ f) \sqsubseteq f.
\end{aligned}$$

- Hence, the category of domains and strict continuous functions is cartesian down-closed.
- Can adapt the comonads to this category.

EFFECTIVELY GIVEN DOMAINS

- A domain D is *effectively given* iff its finite elements are recursively enumerable, consistency of pairs of finite elements is decidable, and the lub of every consistent pair of finite elements is computable.
- An element of D is *computable* iff its set of finite approximations is recursively enumerable.
- The category of effectively given domains and computable functions is cartesian closed.

The T_1 comonad can be adapted to this category, by defining $T_1 D$ to be the *computable increasing paths* over D.

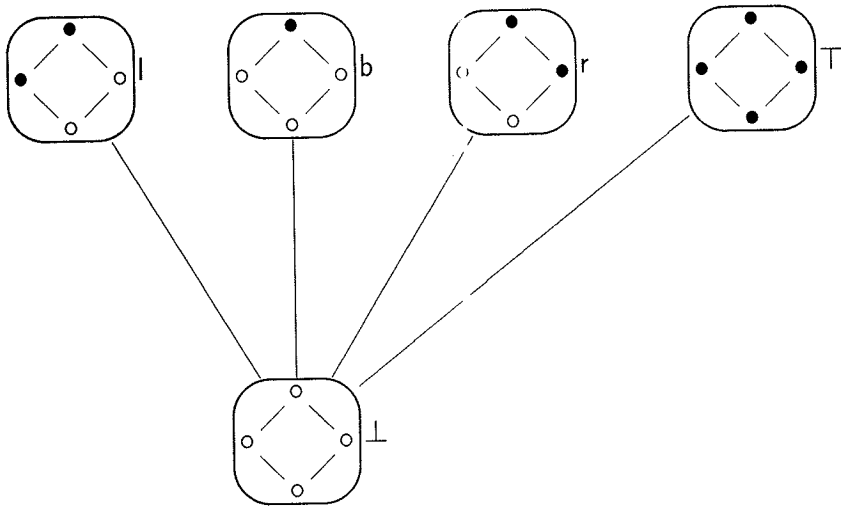
- All of the auxiliary operations (ϵ, δ, γ, and so on) are computable.
- We obtain a Kleisli category of effectively given domains and computable algorithms.
- This category quotients onto the underlying category of effectively given domains and computable functions.
- The computable algorithms category is again cartesian closed.

DI-DOMAINS and STABLE FUNCTIONS

- A dI-domain is an ω-algebraic directed-complete, bounded complete, pointed partial order satisfying:
  - Distributivity: for all x, if y and z are consistent then $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$;
  - Property I: below every finite element there are only finitely many finite elements.
- A function f on dI-domains is stable if f is continuous and $f(x \sqcap y) = f(x) \sqcap f(y)$ whenever x and y are consistent.
- The stable ordering on stable functions is defined by: f stably approximates g iff f pointwise approximates g and whenever $x \sqsubseteq y$, $f(x) = f(y) \sqcap g(x)$.
- The category of dI-domains and stable functions is cartesian closed. Currying, uncurrying and application are the usual notions, restricted to the stable setting.
- Can adapt the T_1 comonad to this stable setting, by using the stable order on $T_1 D$.

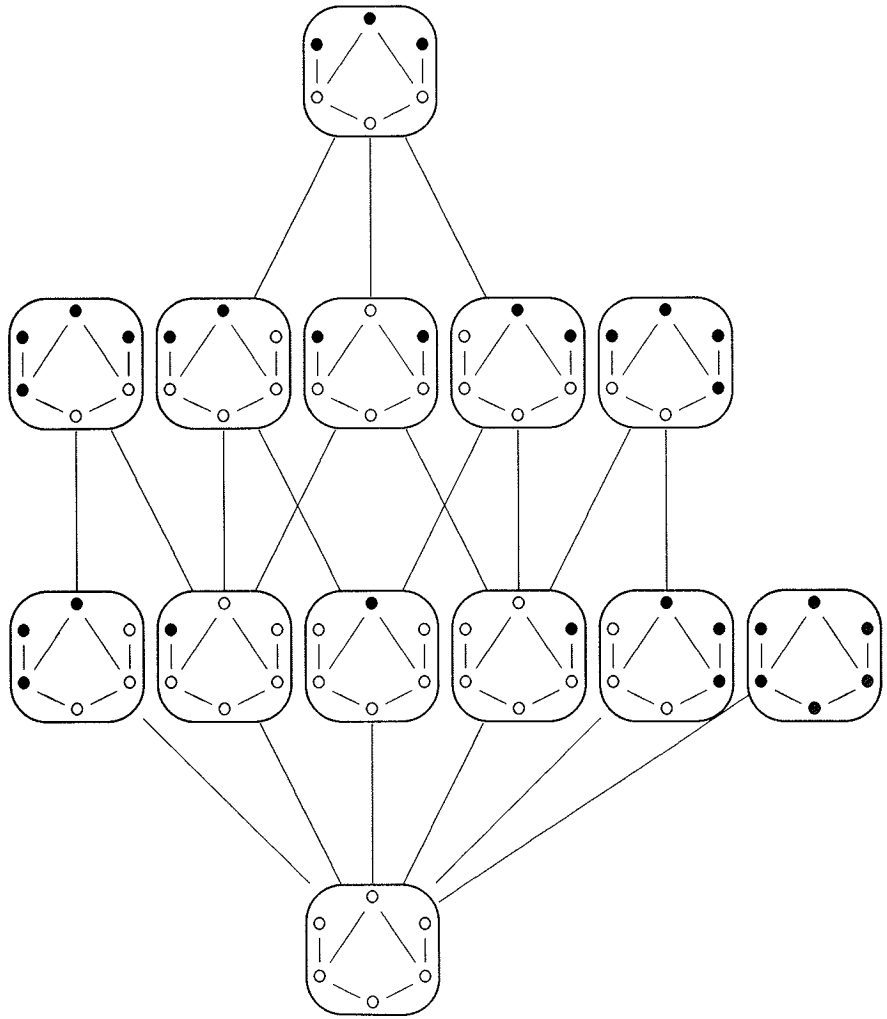
STABLE FUNCTIONS from 2×2 to 2

ordered stably



STABLE T_1 ALGORITHMS from 2×2 to 2

ordered stably



REFERENCES

1. *Computational Comonads and Intensional Semantics.* To appear in the Durham Symposium Proceedings, Cambridge University Press, 1992. Technical Report CMU-CS-91-190.

2. *Continuous Functions and Parallel Algorithms on Generalized Concrete Data Structures.* To appear in the Proceedings of MFPS'91, Springer LNCS, 1992. Technical Report CMU-CS-91-160.

3. *A Cartesian Closed Category of Parallel Algorithms on Scott Domains.* Submitted for publication. Technical Report CMU-CS-91-159.

4. *Towards a Theory of Parallel Algorithms on Concrete Data Structures.* Proceedings of Leicester Symposium on Semantics for Concurrency, Springer Verlag, 1990.

CLiCS WORKSHOP ÅRHUS, MARCH '92

CO-INDUCTION FOR RECURSIVELY DEFINED DOMAINS

Andrew Pitts
Univ. of Cambridge Computer Lab.

Aim: improve the collection of proof principles we have for reasoning about recursively defined datatypes in (strict or lazy) functional programming languages.

This talk will present one such proof principle.

(Longer-term aim: extend the analysis to include some of ML's imperative features – introduced and controlled via computational monads, evaluation logic, ...)

Even if we cut down to a fragment of ML without imperative features, the unrestricted use of the function-type constructor and of polymorphism in recursive datatype declarations means that a mathematical framework adequate for modelling the values of such datatypes is necessarily quite subtle (= "hard for software engineers to use")

To get you in the mood, here are some examples of varying nastiness...

Some varieties of recursive datatype (in pure ML)

- inductively defined <u>set</u>, eg:

  datatype α list = Nil | Cons of $\alpha * \alpha$ list

  predomain $L = 1 + A \times L$

- functorially defined <u>predomain</u>*, eg:

  datatype αstrm = Nil | Cons of $\alpha * (\text{unit} \to \alpha\text{strm})$

  predomain $S = 1 + A \times S_\perp$

  $S_\perp \xleftarrow{\cong} 1 \to S$

- mixed variance type constructor, eg:

  datatype lambda = Abs of ((unit → lambda) → lambda

  predomain $\Lambda = \Lambda_\perp \to \Lambda_\perp$

- recursively defined <u>type constructor</u>, eg:

  datatype α ty = Nil | Cons of $(\alpha * \alpha\,\text{ty})$ ty

  predomain function

  $T = \lambda X \in \mathcal{OO}.\ 1 + T(X \times T(X))$

---

* use <u>pre</u> domains (no least elts required) in order that sets be instances of "domains", so that the first eg gives the set of A-lists, not the lift of that s

The Scott / Plotkin / Smyth / Wand / Lehman / ..
theory of recursively defined (pre)domains / information systems
provides a good mathematical framework for giving
a semantics for such recursive datatype declarations
(and more...). Not only do we get solutions to
the predomain equations, but also ones that are
minimal in a suitable sense – eg initial w.r.t.
embedding / partial-projections between predomains.

However, ...

Useful logical principles for reasoning about recursively defined (pre)domains are somewhat lacking.

Criteria for usefulness:

- should apply to general recursive domains, not just to "inductive" ("algebraic") ones
- should be elementary — i.e. not depend upon the detailed construction of the recursive domain within set theory / higher order logic.
- should be structural — i.e: stated in terms of the structure of the domain constructor involved
- should be used !

An existing example : S. Abramsky, "Domain theory in logical form", Ann. Pure & Appl. Logic 51 (1991) 1–77

– provides a theory of the open subsets ("observable" properties) of recursive domains that meets the above criteria

Two important types of proof principle

INDUCTION

Used to prove that a property* of elements of a recursively defined predomain actually holds for all elements

[*would like to handle properties that determine chain-complete subsets rather than just open subsets - so outside Abramsky's setting.]

EXTENSIONALITY

Used to prove that one element (eg defined as the fixpoint of some function) of a recursively defined predomain approximates (or maybe, equals) another such element.

CLAIM: there are general INDUCTION & EXTENSIONALITY principles for recursively defined predomain constructs that meet the criteria mentioned before.
In this talk, will describe a "co-induction" principle for extensionality.

Let $\mathcal{O}$ be some collection of chain-complete posets containing $\emptyset$, 1 and closed under the following

(PRE)DOMAIN CONSTRUCTORS (of a single variable)

$$\begin{array}{lll} \Phi ::= K & & \text{fixed predomain in } \mathcal{O} \\ \mid \alpha & & \text{variable (for which a predomain can be substitut} \\ \mid \Phi \times \Phi & & \text{cartesian product} \\ \mid \Phi + \Phi & & \text{disjoint union} \\ \mid \Phi_\perp & & \text{lifting} \\ \mid \Phi \to \Phi & & \text{continuous partial function space} \\ & \leftarrow \text{(identify with } \Phi \to (\Phi_\perp)\text{)} & \end{array}$$

Notation: $\Phi(D)$ denotes the predomain resulting from substituting $D \in \mathcal{O}$ for the variable α in Φ

PROPOSITION ("bottomless" version of the standard result due to Scott et multo al)

For each such Φ, there is $\mathrm{Fix}\Phi \in \mathcal{O}$ satisfying $\Phi(\mathrm{Fix}\Phi) \cong \mathrm{Fix}\Phi$ and initial amongst such predomains with respect to embedding/partial-projection pairs.

Call $\mathrm{Fix}\Phi$ "the predomain recursively defined by Φ".

The "co-induction" principle for $\mathrm{Fix}\,\Phi$

Let $k : (\mathrm{Fix}\,\Phi)_\perp \cong (\Phi(\mathrm{Fix}\,\Phi))_\perp$

be the lift of the isomorphism exhibiting $\mathrm{Fix}\,\Phi$ as the predomain recursively defined by Φ.

Call a binary relation $\lhd \subseteq (\mathrm{Fix}\,\Phi)_\perp \times (\mathrm{Fix}\,\Phi)_\perp$ a Φ-simulation if it satisfies

$$\forall u,v \in (\mathrm{Fix}\,\Phi)_\perp \left(u \lhd v \supset k(u)\ \Phi'(\lhd)\ k(v) \right)$$

where $\lhd \mapsto \Phi'(\lhd)$ is a certain operation on binary relations defined (by induction on the structure of Φ) bel

THEOREM

For all $u,v \in (\mathrm{Fix}\,\Phi)_\perp$,
$u \sqsubseteq v$ in $(\mathrm{Fix}\,\Phi)_\perp$ if and only if there is some Φ-simulation $\lhd$ with $u \lhd v$

The theorem characterizes $\sqsubseteq$ on $(\mathrm{Fix}\,\Phi)_\perp$ (and hence $\sqsubseteq$ on $\mathrm{Fix}\,\Phi$) — it displays $\sqsubseteq$ as the greatest fixed point* of a certain monotone operator on binary relations on $(\mathrm{Fix}\,\Phi)_\perp$ (ordered by inclusion)

[* — hence the tag "co-induction".]

The principle was inspired by thinking about (the final co-algebra part of) Freyd's recent work on "Algebraically Compact" categories. However, the use of arbitrary relations appears to take us out of the category of domains.

Lifting binary relations on predomains

Notations: $Rel(D,D) = \{\lhd \mid \lhd \subseteq D \times D\}$
binary relations on D

Let $d \mapsto [d]$ denote the insertion of D into $D_\perp$.

Given $\lhd \in Rel(D,D)$,
define $\lhd_\perp \in Rel(D_\perp, D_\perp)$ to be the binary relation

$u \lhd_\perp u' \equiv \forall d \in D\,(u = [d] \supset \exists d' \in D\,(u' = [d'] \,\&\, d \lhd d'))$

(I.e. $\lhd_\perp$ relates $\perp$ to anything, and relates an element in D to something only if that something is also in D & already related to the element via $\lhd$.)

Note that if $\lhd$ is $\sqsubseteq_D$ (the partial order for D)
then $\lhd_\perp$ is $\sqsubseteq_{D_\perp}$

We can extend this property of lifting to the other predomain constructors ...

Definition of $\Phi' : Rel(D_\perp, D_\perp) \to Rel((\Phi D)_\perp, (\Phi D)_\perp)$

CASE Φ is K : $\Phi'(\lhd)$ is $\sqsubseteq_{K_\perp}$

CASE Φ is α : $\Phi'(\lhd)$ is $\lhd$

CASE Φ is $\Psi_\perp$: $\Phi'(\lhd)$ is $(\Psi'(\lhd))_\perp$ ← lifted relation

CASE Φ is $\Psi \to \Theta$: $\Phi'(\lhd)$ is the binary relation on $(\Psi D \to (\Theta D)_\perp)_\perp$ given by :

$u\ \Phi'(\lhd)\ v \equiv \forall f \in (\Psi D \to \Theta D).\ u = [f] \supset$
$\exists g \in (\Psi D \to \Theta D).\,(v = [g]\ \&$
$\forall x \in \Psi D.\ f(x)\ \Theta'(\lhd)\ g(x))$

NB : Weaker than "logical relations" — does not depend upon $\Psi'(\lhd)$. This is why $\lhd \mapsto \Phi'(\lhd)$ is monotone for inclusion of relations.

CASES Φ is $\Psi \times \Theta$, $\Psi + \Theta$ — omitted.

There is a version of the co-induction principle that uses relations on $\mathrm{Fix}\,\underline{\Phi}$ rather than on $(\mathrm{Fix}\Phi)_\perp$, plus an operation

$$\mathrm{Rel}(D,D) \longrightarrow \mathrm{Rel}(\Phi D, \Phi D)$$

However, it is a weaker principle than the one given: using it to prove $U \sqsubseteq V$ in $(\mathrm{Fix}\,\underline{\Phi})_\perp$ we'd have to exhibit a relation $\lhd$ on $(\mathrm{Fix}\,\underline{\Phi})_\perp$ that satisfies not only a "simulation-like" condition, but also the extra condition:

- $\perp \lhd V \equiv V = \underline{\perp}$

Don't intend to give the proof of the THEOREM here. Rather, will concentrate on examples of its use. The force of the Co-Induction Principle is that it gives us a (useful!) criterion for testing whether one element of the lift of a recursively defined predomain approximates another — we just have to find some simulation relating the elements, and in particular (cases the simulation could be quite small. (Cf. use of bisimulations in CCS — see later for a precise relationship.)

Here is a simple example...

A SIMPLE EXAMPLE (from the realm of lazy datatypes) to illustrate that Φ-simulations can be quite small.

Consider the predomain S interpreting the ML datatype

datatype intstrm = Nil | Cons of int * (unit → intstrm)

i.e. $S = \mathrm{Fix}\,\Phi$ with $\Phi = 1 + \mathbb{Z} \times \alpha_\perp$

To prove: there is a **unique** element $ones \in S_\perp$ satisfying

$k(ones) = [\,\mathrm{inr}(1, ones)]$ (where $k : S_\perp \cong (1 + \mathbb{Z} \times S_\perp)_\perp$)

(The existence of such an element is guarenteed by the usual fixpoint theory.)

Proof Suppose $ones' \in S_\perp$ also satisfies $k(ones') = [(1, ones$

It is easy to check that

$$\lhd \;=\; \{\,(ones,\, ones')\,\}$$

is a Φ-simulation: since $ones \lhd ones'$, it follows from the co-induction principle that $ones \sqsubseteq ones'$. A symmetric argument yields $ones' \sqsubseteq ones$, and hence $ones = ones'$. □

EXAMPLE: Abramsky's canonical domain model for the untyped "lazy lambda calculus"

is $(\mathrm{Fix}\,\Phi)_\perp$ with $\Phi = \alpha_\perp \to \alpha$.

Writing $\Lambda = (\mathrm{Fix}\,\Phi)_\perp$ and $u \Downarrow f$ to mean $u = [f] \in (\Lambda \to \Lambda)_\perp$; then in this case the co-induction principle says that $\sqsubseteq$ is the greatest binary relation $\lhd$ on Λ satisfying the "applicative bisimulation" property:

$$\forall u, v \in \Lambda.\ \forall f \in \Lambda \to \Lambda.\quad u \lhd v \;\&\; u \Downarrow f \;\supset\; \exists g \in \Lambda \to \Lambda.\,\big(v \Downarrow g \;\&\; \forall x \in \Lambda.\ f(x) \lhd g(x)\big)$$

Abramsky proves this as Theorem 4.1 ("internal full abstraction") in "The Lazy Lambda Calculus". In: D. Turner (ed.) "Research Topics in Functional Programming" (Addison-Wesley, 1990).

EXAMPLE : Abramsky's "domain equation for bisimulation"

(Ref : Inf. & Comp. 92(1991) 161–218.)

Take $\mathcal{O}$ to be $\{ D \mid D_\perp \text{ is a bifinite domain} \}$ and enlarge the collection of predomain constructors:

$$\Phi ::= \cdots \mid P\Phi \mid \cdots$$

where $(PD)_\perp$ = Plotkin powerdomain on $D_\perp$

$$= \left(\left\{ \begin{array}{l} \text{non-empty, convex, Lawson-} \\ \text{closed subsets of } D_\perp \end{array} \right\}, \begin{array}{l} \text{Egli-} \\ \text{Milner} \\ \text{order} \end{array} \right)$$

The action of P on binary relations sends $\lhd \subseteq D_\perp \times D_\perp$ to $P'(\lhd) \subseteq (PD)_\perp \times (PD)_\perp$ where for all $u, v \in (PD)_\perp$

$$u \; P'(\lhd) \; v \equiv \forall x \in u \; \exists y \in v \; (x \lhd y) \; \& \; \forall y \in v \; \exists x \in u \; (x \lhd y)$$

NB. When $\lhd$ is $\sqsubseteq_{D_\perp}$, then $P'(\lhd)$ is the Egli-Milner order, i.e. is $\sqsubseteq_{(PD)_\perp}$.

Abramsky's "domain for bisimulation" is $(\mathrm{Fix}\,\Phi)_\perp$ with $\Phi = 1 + P(\mathbb{N} \times \mathrm{Id}_\perp)$.

Writing $\mathcal{B} = (\mathrm{Fix}\,\Phi)_\perp$ and

$u \xrightarrow{n} u'$ to mean $\exists x \in P(\mathbb{N} \times \mathcal{B}).\ k(u) = [\mathrm{inr}(x)] \;\&\; [(n,u')] \in [x]$

$u\!\uparrow$ to mean $k(u) = \perp \vee \exists x.\ k(u) = [\mathrm{inr}(x)] \;\&\; \perp \in [x]$

and $u\!\downarrow$ to mean $\neg(u\!\uparrow)$

(where as usual $k : \mathcal{B} \cong (1 + P(\mathbb{N} \times \mathcal{B}))_\perp$), then in this case the co-induction principle can be proved equivalent to the assertion that $\sqsubseteq$ is the largest relation $\lhd$ on $\mathcal{B}$ satisfying the "partial bisimulation" property:

$$\forall u, v \in \mathcal{B}.\ u \lhd v \supset$$
$$\forall u', n\,(u \xrightarrow{n} u' \supset \exists v'(v \xrightarrow{n} v' \;\&\; u' \lhd v'))$$
$$\&\quad u\!\downarrow \supset v\!\downarrow \;\&\; \forall v', n\,(v \xrightarrow{n} v' \supset \exists u'(u \xrightarrow{n} u' \;\&\; u' \lhd v'$$

This is Proposition 3.11 ("internal full abstraction") of Abramsky, loc. cit.

Limitations of the co-induction principle:

Example

When $\Phi = \alpha \to K$, then (essentially) the only Φ-simulation is $\sqsubseteq$ and so the co-induction principle does not give us any useful information about the nature of $\mathrm{Fix}\,\Phi$ in this case.

(Similarly for the predomain interpreting the datatype

datatype cont = C of cont → int list | D

used in Tofte's breadth-first search function.)

The co-induction principle stated here can be extended in a number of important directions:

- Simultaneous domain equations
- parameterized recursive types (i.e. with $\mathrm{Fix}\alpha.\ \Phi(\alpha,\beta,\ldots)$ as one of the domain constructors)
- (simultaneously) recursively defined domain constructors (cf. the last of the ML examples given at the beginning)
- the co-induction principle also applies to predomains involving recursively defined predomains, i.e. ...

...

COROLLARY (of the THEOREM)

Let Φ and Ψ be predomain constructors (of a single variable). Then $u \sqsubseteq v$ in $(\Psi(\mathrm{Fix}\,\Phi))_\perp$ if and only if there is some Φ-simulation $\lhd$ with $u\ \Psi'(\lhd)\ v$.

Future directions

- monads other than lifting
- formal meta-logic incorporating the co-induction principle (& the induction principle not mentioned here)
  Already have indications of interesting axiomatic development of domain theory. Eg. fix point object and least fixed point combinator become definable.
- tie-up with Howe's operationally-based work ("Equality in Lazy Computation Systems," LICS'89)
- tie-up with Synthetic Domain Theory
- more examples needed, eg in reasoning about lazy datatypes — suggestions welcomed!

π - Calculi (Milner)

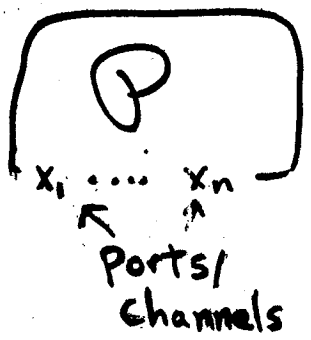
Basic Ideas:

- Fundamental Units = **Processes**
- Processes are (concurrently) communicating via distinguished **Ports** or **channels**.

  (e.g. telephone system, LAN, ...)

Features:

- untyped theory – channels, values, etc. denoted by vbls called **names**



} Process terms with free names $\subseteq \{x_1, \ldots, x$

Processes

$P ::=$ Basic Processes $\mid P \| P \mid \pi.P \mid (\nu x)P$

(possibly others – e.g. $P+Q$)

$(\nu x)P$ (hiding) : x becomes a private channel local to P
(like an $\exists$ quantifier)

π Prefixes ("plugging info")

$\bar{x}\langle y_1 \cdots y_n \rangle$: sends names y_i along x

$x(y_1 \cdots y_n)$: receives arbitrary names along x for $y_1 \cdots y_n$ – so $y_1 \cdots y_n$ are bound by this op^n.

Write $\bar{x}y_1 \cdots y_n$ for $\bar{x}\langle y_1 \cdots y_n \rangle$.
"hand-shaking protocols".

$\bar{x}y.P \| x(z).Q \| x(w)\,u(v)\,\bar{v}w.R \| \cdots$

Basic Communication

$$x(y).P \| \bar{x}z.Q \succ P\{{}^{z}/_{y}\} \| Q$$

- Unlike β-reduct^n, Q continues ....
- Not communicating processes along channels : just access protocols

Congruences : Basic one $\equiv$
(plus more subtle syntactic ones...)
compatible with $\succ$:

(Processes$/\equiv$, $\|$, $\bullet$) is commutatin monoid, where $\bullet$ = null process

$\nu x \bullet \equiv \bullet$ (Write $\pi\bullet$ as π)

$\nu x(P \| Q) \equiv (\nu x P) \| Q$ if $x \notin fn(Q)$

$\nu x y P \equiv \nu y \nu x P$ etc.

Note : $u(v).[x(y) \| \bar{x}y] \not\succ$

① $\bar{x}y \parallel x(u)\,\bar{u}v \parallel \bar{x}z$

$\swarrow$ $\searrow$

$\bar{y}v \parallel \bar{x}z$ $\qquad$ $\bar{x}y \parallel \bar{z}v$

② $x(w)\,\bar{x}u$ deadlock

③ $(\nu x)(\bar{x}y \parallel x(w)\,\bar{u}v) \parallel \bar{x}z$

$\succ \nu x(\bar{y}v) \parallel \bar{x}z \equiv \bar{y}v \parallel \bar{x}z$

④ $u(v)\,[x(y) \parallel \bar{x}z]$ deadlock

Other possible processes:

$P+Q$: choice

$!P$: replication, obeying

e.g. $!P \equiv P \parallel !P$

$\bar{x}y \parallel !x(u)\,\bar{u}v \parallel \bar{x}z \equiv$

$\bar{x}y \parallel x(u)\,\bar{u}v \parallel !x(u)\,\bar{u}v \parallel \bar{x}z$

$\succ \bar{y}v \parallel !x(u)\,\bar{u}v \parallel \bar{x}z$ ----.

Synchronous π-Calculus

(Milner 1992 : unpublished)

Processes $P ::= \bullet \mid \nu x\,P \mid P \parallel P \mid \pi P$

(later: $P+P$, ...)

No dots after prefixes

$\pi\bullet$ refers to a process that stops (no further action) after prefix π is done.

$\bar{x}z\bullet \parallel x(y)\,\bar{u}y\bullet$

$(\nu x)(\bar{x}z\bullet \parallel u(w)\bullet)$

} usually omit writing $\bullet$

Structural Congruence

(i)* $\pi_1\pi_2 P \equiv \pi_2\pi_1 P$

(ii)* $(\pi P \parallel Q) \equiv \pi(P \parallel Q)$ ← π prefix or νx

* { if nothing bad becomes free & vice versa

(iii) $\nu x\bullet \equiv \bullet$, (iv) α conversion

(v) (Processes/$\equiv$, $\parallel$, $\bullet$) commutative monoid

(vi) Usual rewriting (in contexts) : $\succ$

Ex: $\bar{x}y\,x(z)P \succ P\{y/z\}$, $x(z)\bullet \parallel P \equiv x($

Multiplicative Linear Logic

Formulas : $A \otimes B$, $A \parr B$, $A^\perp$

Proofs : **Axiom** $\vdash A, A^\perp$

Par
$$\frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}$$

Tensor
$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}$$

Cut
$$\frac{\vdash \Gamma, A \quad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}$$

$\pi \left(\begin{array}{c} \vdots \\ \vdash x_1 : A_1, \ldots, x_n : A_n \end{array} \right)$ is a synch.

π-term with free names $\subseteq \{x_1, \ldots x_n\}$

Axiom π-terms

$\vdash x : A, y : A^\perp$ $\quad x(a)\bar{y}a$

$\vdots F$

Par
$$\frac{\vdash \tilde{w} : \Gamma, x : A, y : B}{\vdash \tilde{w} : \Gamma, z : A \parr B} \qquad (\parr F)\tilde{w}z = z(x,y)\, F\tilde{w}xy$$

(receiver)

Tensor $\vdots F \quad \vdots G$

$$\frac{\vdash \tilde{w} : \Gamma, x : A \quad \vdash \tilde{v} : \Delta, y : B}{\vdash \tilde{w} : \Gamma, \tilde{v} : \Delta, z : A \otimes B} \qquad \otimes(F,G)\tilde{w}\tilde{v}z = \nu xy\,(\bar{z}xy.(F\tilde{w}x \,\|\, G\tilde{v}y))$$

(sender)

Cut ⋮F ⋮G

$\vdash \tilde{u}:\Gamma, x:C \qquad \vdash \tilde{v}:\Delta, y:C^{\perp}$

$\vdash \tilde{\omega}:\Gamma, \tilde{v}:\Delta$

$Cut(F,G) \equiv_{def} \nu z.[F[z/x] \| G[z/y]]$

Theorem 1. If ϑ is an MLL- derivation of $\vdash \Delta$ and $\vartheta \succ \vartheta'$, then $F \vdash_{\pi} F'$ where F, F' are the associated π-terms.

$$\text{cut}\ \dfrac{\dfrac{\vdots F \quad \vdots G}{\vdash \Gamma, A \qquad \vdash \Delta, B}}{\vdash \Gamma, \Delta, A \otimes B} \qquad \dfrac{\vdots H}{\vdash \Delta', A^{\perp}, B^{\perp}}}{\vdash \Gamma, \Delta, \Delta'}$$

$Cut(\otimes(F,G), \parr H) =$

$\nu z\,(\otimes(F,G)\,\tilde{u}\,\tilde{v}\,z \,\|\, (\parr H)\,\tilde{\omega}\,z)$

$\equiv \nu z\,x\,y\,(\bar{z}\,x\,y\,(F\,\tilde{u}\,x \| G\,\tilde{v}\,y) \| z(x',y')\,H\,\tilde{\omega}\,x'\,y')$

$\succ \nu x\,y\,(F\,\tilde{u}\,x \| G\,\tilde{v}\,y \| H\,\tilde{\omega}\,x\,y)$

$\equiv Cut(Cut(F,H),G)$

$$\dfrac{\dfrac{\vdots F \quad \vdots H}{\vdash \Gamma, A \qquad \vdash \Delta', A^{\perp}, B^{\perp}}\ \text{cut}}{\vdash \Gamma, \Delta', B^{\perp}} \qquad \vdots G \quad \vdash \Delta, B\ \text{cut}$$

$$\dfrac{\vdash \Gamma, \Delta', \Delta}{\vdash \Gamma, \Delta, \Delta'}$$

- Commutative red^{n}s give identical π-terms (modulo $\equiv$).

$\therefore$ translation occurs at the level of <u>proof nets</u>.

So, the theorem becomes:

<u>Thm</u>: For MLL proof nets:

$\nu > \nu' \Rightarrow \pi(\nu) > \pi(\nu')$

net red^{n} ⇒ synch. π-term red^{n}.

- In fact, the translation

Π : <u>Proof-Structures</u> → π-terms

(∴ synch. π-calculus refers to communication protocols & links, <u>not</u> derivability, although it will fully mirror proofs as well)

<u>MLL Process Combinators</u> (Abramsky)

<u>Net</u>	<u>Process Combinators</u> <u>Synch. π-terms</u>
$x:A \quad y:A^{\perp}$	$I_{x,y}$ $\quad x(a)\,\bar{y}\,a$
$x:A \quad y:B$ / $z:A \otimes B$	$\otimes^{x\,y}_{z}(P_x, Q_y)$ $\quad \nu_{xy}\,\bar{z}\langle xy\rangle [P_x \| Q_y]$
$x:A \quad y:B$ / $z:A \parr B$	$\parr^{x\,y}_{z}(R_{xy})$ $\quad z(x,y).R_{xy}$
$x:A \quad y:A^{\perp}$ / Cut	$\mathrm{Cut}_{z}(P\{z/x\}, Q\{z/y\})$ $\quad \nu z\,[P\{z/x\} \| Q\{z/y\}]$

red vbls x,y,z are free.

z is common cut-variable (hidden channel)

Abramsky

$$I_{xy} \equiv x(a).\bar{y}a.I_{xy} + y(a)\bar{x}a.I_{xy}$$

$$\bigotimes_{z}^{x,y}(P_x, Q_y) \equiv (\nu x\,y)(\bar{z}xy.\mathbf{0} \parallel P_x \parallel Q_y)$$

$$\underset{z}{\curlyvee}^{x,y} R \equiv z(xy).R$$

Combinator Eqⁿs for Sfructures (> nets)

$$Cut_x(P,Q) \equiv Cut_x(Q,P)$$

$$Cut_x(P_x, I_{xy}) \succ P\{y/x\}$$

$$Cut_z\left(\bigotimes_{z}^{x,y}(P_x,Q_y), \underset{z}{\curlyvee}^{x,y} R_{xy}\right) \succ$$

$$Cut_y(Cut_x(P_x, R_{xy}), Q_y)$$

$$\equiv Cut_x(P_x, Cut_y(R_{xy}, Q_y))$$

Commutative Rdⁿs

$$Cut_x\left(\underset{v}{\curlyvee}^{c,d} F_{xcd}, G_x\right) \succeq \underset{v}{\curlyvee}^{c,d} Cut_x(F,G)$$

(here $\curlyvee$ does not react with x)

$$Cut_x\left(\bigotimes_{v}^{c,d}(P_e, Q_{dx}), R_x\right) \equiv$$

$$\bigotimes_{v}^{c,d}(P_e, Cut_x(Q_{dx}, R_x))$$

(here $\bigotimes$ does not react with x)

π : MLL Proof Structures $\longrightarrow$ π-Calc

- $\pi\left(\begin{matrix}\mathcal{R} \\ \Gamma\end{matrix} \cup \begin{matrix}\mathcal{S} \\ \Delta\end{matrix}\right) = \pi\left(\begin{matrix}\mathcal{R} \\ \Gamma\end{matrix}\right) \| \pi\left(\begin{matrix}\mathcal{S} \\ \Delta\end{matrix}\right)$

- $\pi\left(\dfrac{\overset{\ddots}{x:A \quad y:A^\perp}}{\text{cut}}\right) = \nu z\, P_{xy}\{\frac{z}{x}, \frac{z}{y}\}$

... etc.

Prop^n : IF $\mathcal{R}$ is a proof structure, $\pi(\mathcal{R})$, modulo $\equiv$, does not depend on the particular order of the inductive construct^n of $\mathcal{R}$.

e.g. $P(u,v)\, \nu xy\, \overline{t}xy\, \overset{\ddots}{P} \equiv (\nu x y)\, \overline{t}xy\, P(uv)\, \overset{\ddots}{P}$

$$\dfrac{\dfrac{B \quad C}{B \otimes C} \quad \dfrac{B^\perp \quad C^\perp}{B^\perp \parr C^\perp}}{\vdots\ P}$$

Thm (Local Fullness) : $\forall$ proof struc. $\mathcal{R}$ for MLL,

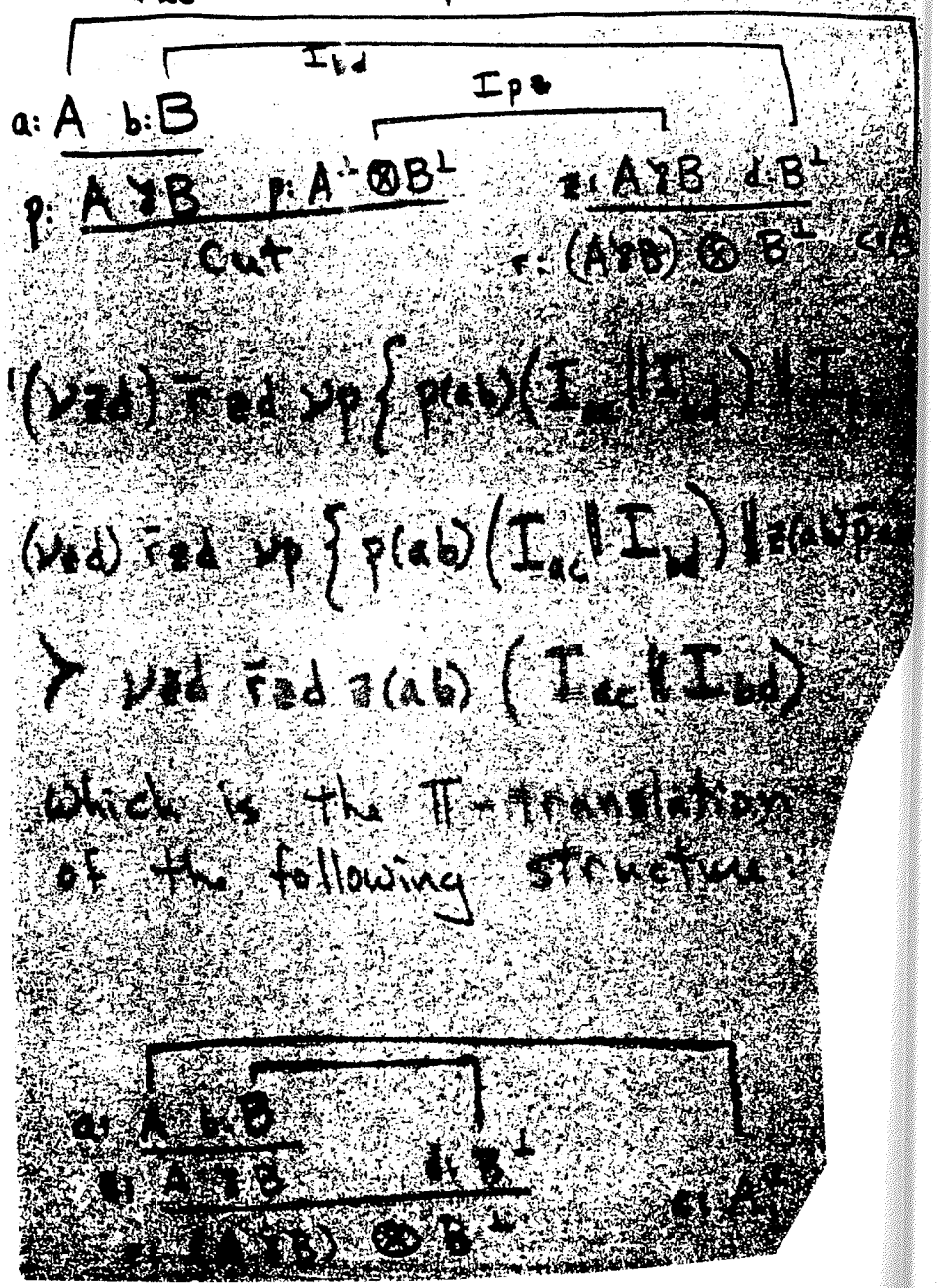
(i) IF $\pi(\mathcal{R}) \succcurlyeq_\pi Q$ then $\exists$ proof structure $\mathcal{S}$ s.t. $Q = \pi(\mathcal{S})$ & $\mathcal{R} \underset{\text{nets}}{\succcurlyeq} \mathcal{S}$

(ii)
$$\begin{array}{ccc} \mathcal{R} & \succ & \mathcal{S} \\ \pi \downarrow & & \uparrow \pi^{-1} \\ \pi(\mathcal{R}) & \succ & Q \equiv \pi(\mathcal{S}) \end{array}$$

(iii) IF $\mathcal{R}$ is a net, then $\mathcal{S}$ (above) is too.

Arises from syntactic properties of terms that arise in image of π

Example (NOT NET)



Information Flow

All the above works equally well with opposite orientation of axiom buffer. Indeed,

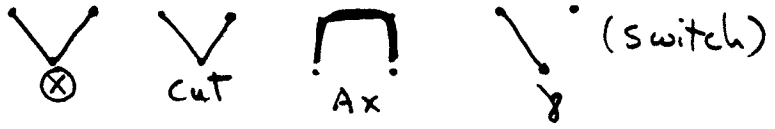
Abramsky : $\overbrace{\quad}^{x:A \quad y:A^{\perp}} : I_{xy}$

uses <u>bidirectional</u> $x(a)\,\bar{y}a + y(a)\,\bar{x}a$

We can actually obtain (always) unidirectional buffers consistent with Cut-elim. process — using more sophisticated proof net theory

- Girard (1986, ...)
- Danos-Regnier (1988-90)
- Bellin-Ketonen (84, 90-91)
- Bellin, van de Weile, Danos, Regnier, ... 92

• **Danos - Regnier** : To every MLL structure $\mathcal{R}$, associate graph $\mathcal{G}(\mathcal{R})$



Thm: $\mathcal{R}$ net $\Leftrightarrow$ $\forall$ choice of switches $\mathcal{G}(\mathcal{R})$ is acyclic, connected

• **Bellin - Ketonen** : a more abstract acyclicity & connectedness cond^{n}

• **Bellin et al** : analogous to theory of pure nets (for untyped λ-calc. & G.O.I.) — assign O's, I's to nets :

Axiom:	$\dfrac{}{I\ O}$		Cut $\dfrac{O\ I}{}$	
$\otimes$	$\dfrac{O\ I}{I}$	$\dfrac{I\ O}{I}$		$\dfrac{O\ O}{O}$
$⅋$	$\dfrac{I\ O}{O}$	$\dfrac{O\ I}{O}$		$\dfrac{I\ I}{I}$

Thm [Bellin] Let $\mathcal{R}$ be MLL proof net. Any D-R switch determines an assignment of O's & I's to formulas in $\mathcal{R}$ s.t.

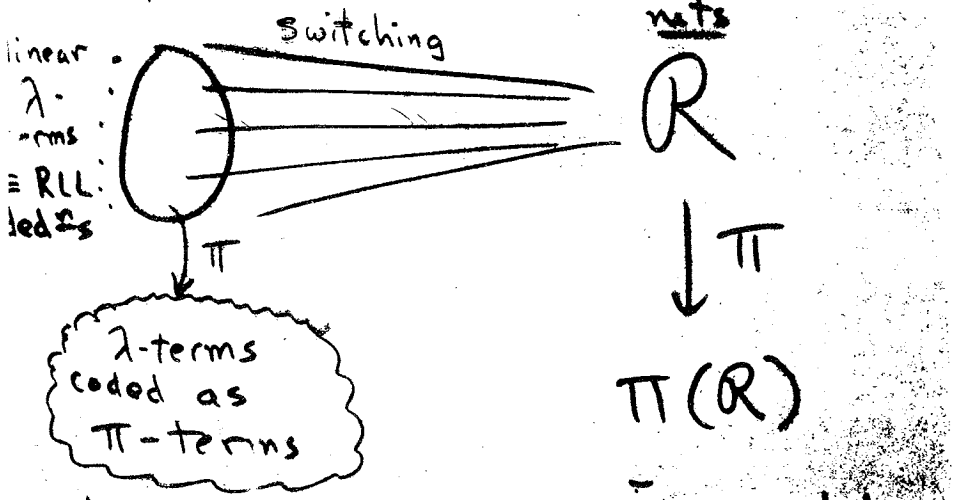
(i) The assignment follows above rules

(ii) Exactly one conclusion is O (the rest I)

(iii) There is a translation to proof $I_1^*, \ldots, I_n^* \vdash O^{**}$ in RLL{ $\multimap$ $\otimes$

$\therefore$ get linear λ-term (with let ....)

& every one arises ....



$\therefore$ Logically coherent choices of orientat^{n} of axiom buffers (appear to be 'variants' of Milner's translations ....)

The Additives

To handle additives, consider π-calculus processes:

$$P ::= \cdot \mid \omega P \mid \nu x P \mid P \| P \mid P + P$$

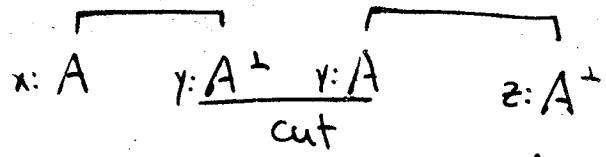
Which distributivities do we allow (both for processes & prefixes?)

Divergence of logic / concurrency world

Consider:

$$P \| (Q + R) \equiv (P \| Q) + (P \| R)$$

- Causes Troubles at axiom level:



suppose axioms = bidirectional buffe

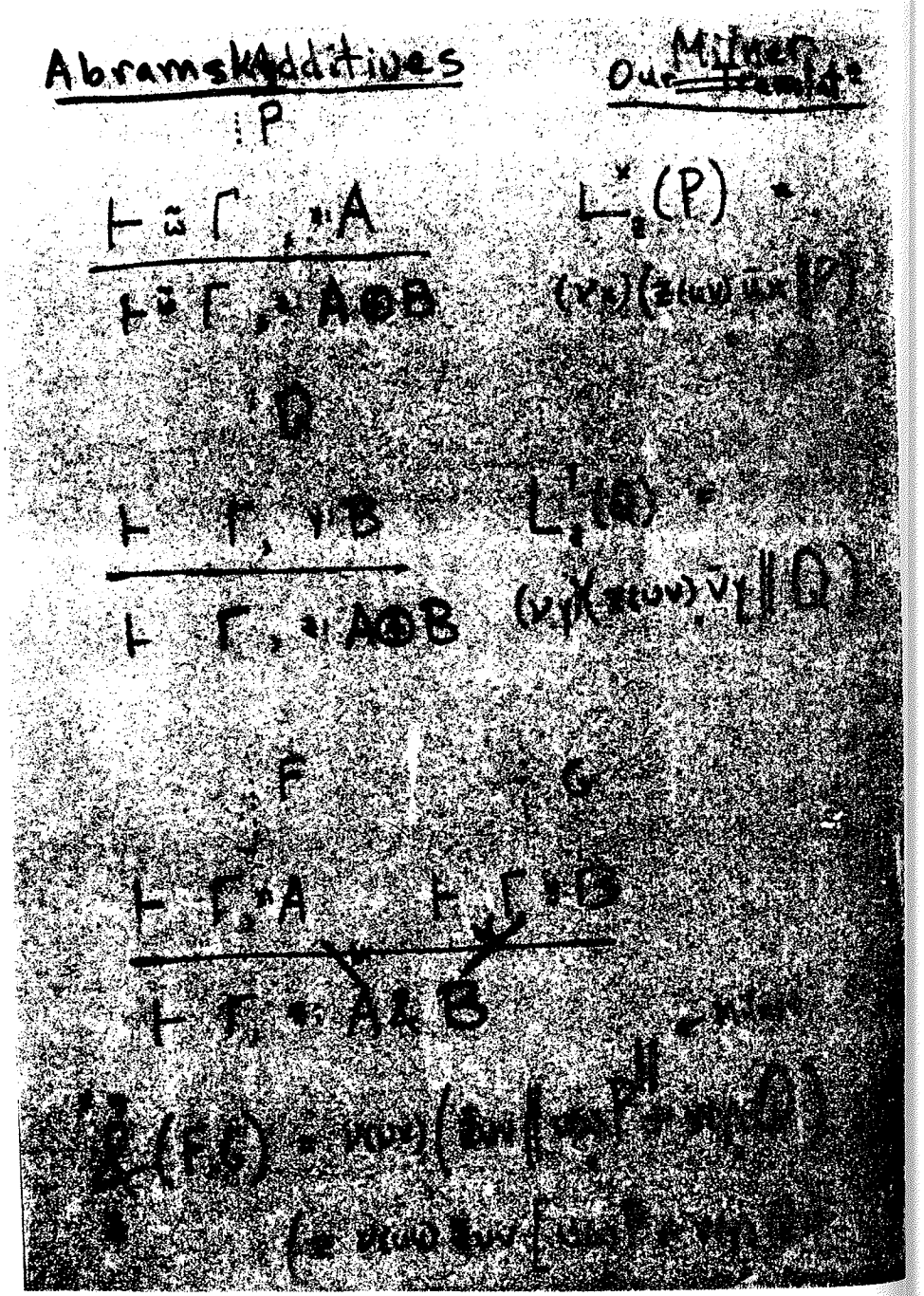
$$\nu y \left[\begin{array}{c} x(u)\bar{y}u \\ + \\ y(u)\bar{x}u \end{array} \;\middle\|\; \begin{array}{c} y(u)\bar{z}u \\ + \\ z(u)\bar{y}u \end{array} \right]$$

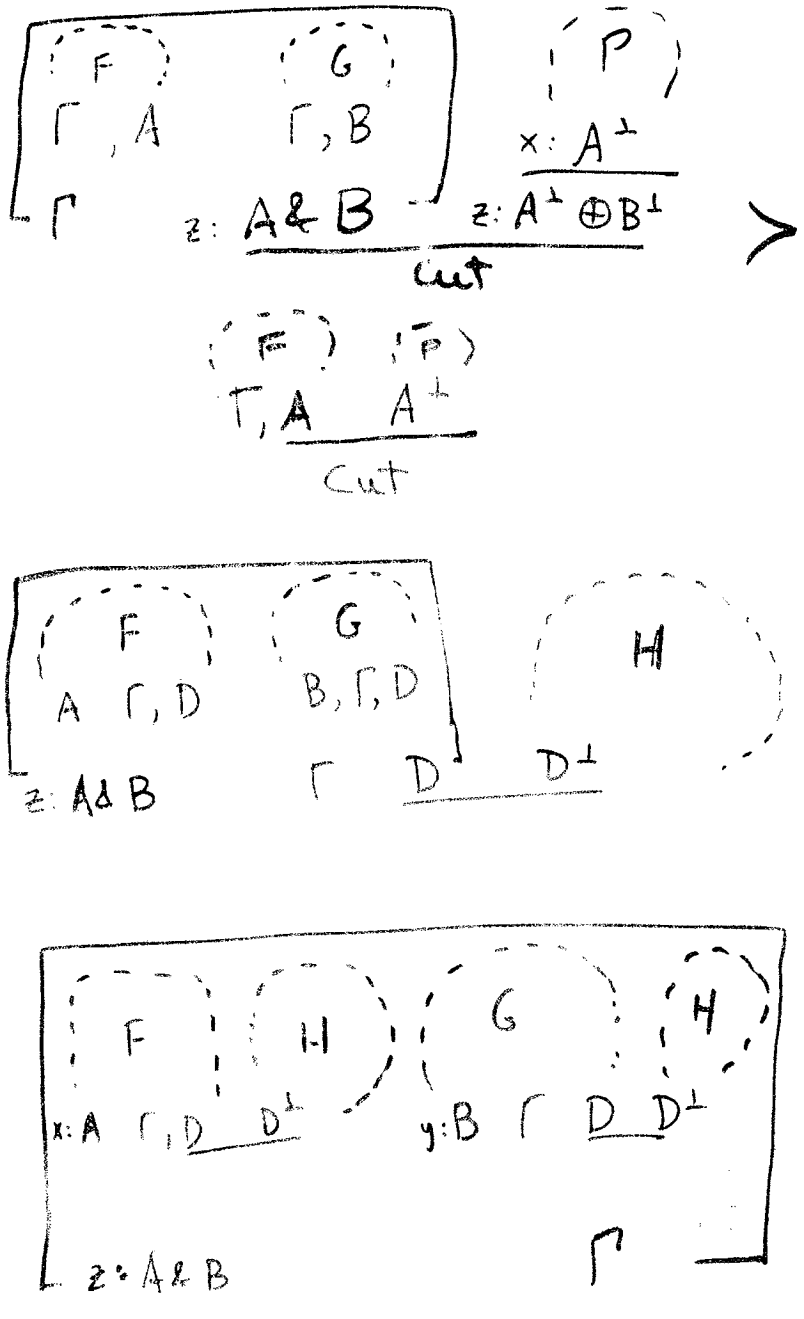
Multiplying out, 2 cross-terms are deadlocked

What to do ? at least 3 choices:

- Use 1 directional buffers (coherent w.r.t. some switching)
- No distributivities on axioms
- Garbage collection of deadlocked terms.

Now, consider Cut-elim and Girard's additive boxes.


Abramsky Additives
Our Milner


F
G
P
Γ, A
Γ, B
x: A⊥
Γ
z: A&B
z: A⊥ ⊕ B⊥
cut
F
P
Γ, A
A⊥
Cut
F
G
H
A
Γ, D
B, Γ, D
Γ
D
D⊥
z: A & B
F
H
G
H
x: A
Γ, D
D⊥
y: B
Γ
D
D⊥
z: A & B
Γ

Process Combinators

$$\mathrm{Cut}_z\left(\underset{z}{\overset{x,y}{\&}}(F,G),\ L^x_z(P)\right) \succ \mathrm{Cut}_x(F,P)$$

$$\mathrm{Cut}_z\left(\underset{z}{\overset{x,y}{\&}}(F,G),\ R^y_z(Q)\right) \succ \mathrm{Cut}_y(G,Q)$$

$$\mathrm{Cut}_d\left(\underset{z}{\overset{x,y}{\&}}(F,G),\ H\right) \succ \underset{z}{\overset{x,y}{\&}}\left(\mathrm{Cut}_d(F,H),\ \mathrm{Cut}_d(G,H)\right)$$

$$\dfrac{\dfrac{\vdots\ F}{\vdash \Gamma, x{:}A} \qquad \dfrac{\vdots\ G}{\vdash \Gamma, y{:}B}}{\vdash \Gamma,\ z{:}A \,\&\, B} \qquad \dfrac{\vdots\ P}{\vdash \Delta, A^\perp}}{\vdash \Delta, z{:}A^\perp \oplus B^\perp}$$

$$\vdash \Gamma, \Delta$$

$$(\nu z)\Big[\nu(uv)\big(\bar{z}uv \parallel u(x).F + v(y).G\big) \parallel \nu x\big[z(u'v')\,\bar{u}'x \parallel P\big]\Big]$$

$$\equiv$$

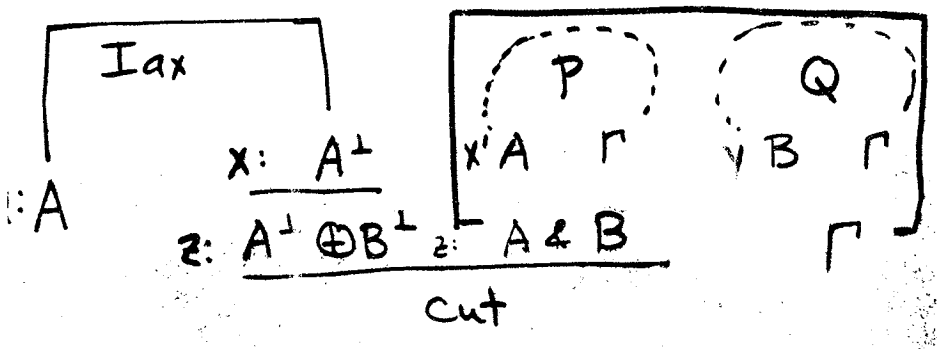
$$(\nu z u v x)\Big[\bar{z}uv\big(u(x).F + v(y).G\big) \parallel z(uv)\,\bar{u}x\,P\Big]$$

$$\succ_* \quad \nu x\big[F\{x'/x\} \parallel P\big] \qquad \circledast$$

$$\dfrac{\dfrac{\vdots\ F}{\vdash \Gamma, x{:}A} \qquad \dfrac{\vdots\ P}{\vdash \Delta, x{:}A^\perp}}{\vdash \Gamma, \Delta}$$

$\circledast$ 2 choices : (1) Communicate (2) Distri… first

EXAMPLE (FOR NETS)



Corresponds to (preparation for) projecting a pair.

$(\nu z\, u\, v\, x)$

$$\{(\bar{z}(u'v')\,\bar{u}'x \parallel Iax) \parallel \bar{z}\,u\,v\,[u(x')\,P_{x'} + v(y)\,Q_y]\}$$

Now what?

- Communicate immediately
  or
- use distributivities & communicate...

Choice (i): communicate immediately

$(\nu z\, u\, v\, x)$

$$\{z(u'v')\,\bar{u}'x \parallel Iax \parallel z\,u\,v\,[u(x')\,P_{x'} + v(y)\,Q_y]\} >$$

$(\nu z\, u\, v\, x)$

$$\bar{u}x \parallel Iax \parallel (u(x')\,P_{x'} + v(y)\,Q_y)$$

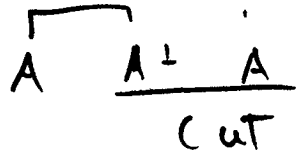
(hard to write net-theoretically — "do cut, but box still there...")

Introduce Choice: In general

$$\bar{u}x\,F \parallel (u(x')\,P_{x'} + v(y)\,Q_y) \triangleright F \parallel P\{^{x}/_{x'}\}$$

(the alternate Q disappears).

Get:



Box is erased.

∴ IF we have choice rules (where U, V distinct)

$$\bar{\nu}x\, F \parallel \left(U(x')P + V(y)Q_y\right) \triangleright F \parallel P\{x/x'\}$$

and dually

$$U(x)\, F \parallel \left(\bar{\nu}x'P + \bar{V}_y Q\right) \triangleright F\{x'/x\} \parallel P$$

can mimic cut-elimination above (at the main door).

But suppose instead we use distributivities:

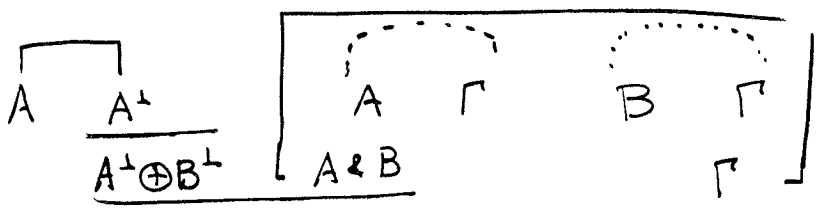
- $\alpha(Q+R) \equiv \alpha Q + \alpha R$ (α prefix or (νx))
- $P \parallel (Q+R) \equiv P \parallel Q + P \parallel R$

$(\nu z\, U V x)$

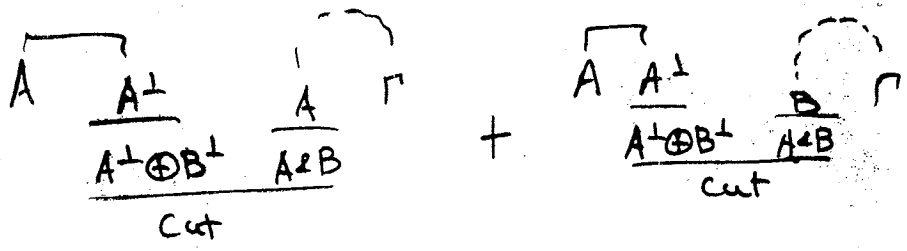
$$\left\{ \left(z(U'V')\, \bar{U}'x \parallel \text{Iax} \right) \parallel \bar{z}UV(U(x')P_{x'}) + \bar{z}UV(V(y)Q_y) \right\}$$

$\triangleright$ ..... (distribute) ....

We then get (at the net level):



becomes

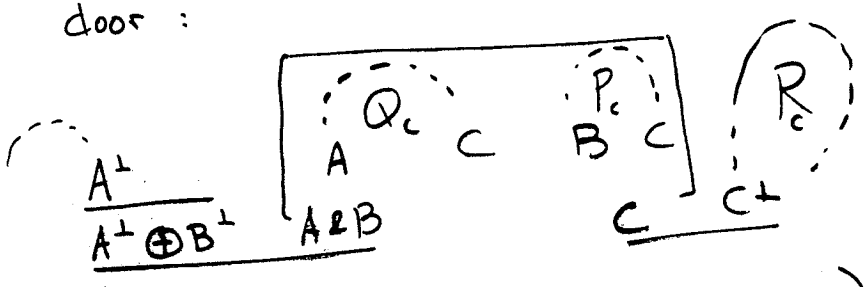


∴ These Π-calculus terms correspond to **slicings** of additive boxes.

∴ Get Σ terms which may include garbage.

∴ IF do garbage coll^{n}, and have no restriction on reductions in boxes, can represent any symmetric (additive) red^{n}.

Suppose have cut at a side door:



$$\nu u v)(\nu c)\left(R_c \parallel \bar{\epsilon} u v \left(u(x) Q_{xc} + v(y) P_{yc} \right) \right)$$

Either bring $\bar{\epsilon} u v$ to front, or distribute & permute c to front of each summand — get terms with c at front, ready to eliminate cut. ✱

In usual concurrency world, make choice of one summand & continue. Essentially imposes $\& = \oplus \cdot \chi$

Want:

$$Cut(\&(P,Q), R) > \&(Cut(P,R), Cut(Q,$$

One option: impose guarding:

Use ordinary π-calc. feature (not in synchronous):

$$\ldots\ u(x).Q + v(y).P \ldots\ .$$

(here: no internal action until prefixes are destroyed)

∴ all options for cut are available but you can't do them: no actions on cuts in box (they're frozen) — or suspended — & only cuts are through main door: action at side doors subordinate to act^n at main door.

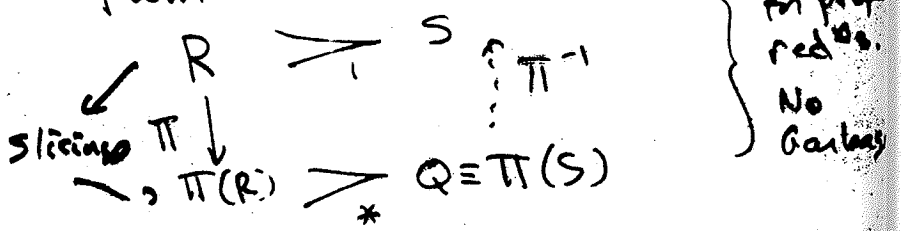
Alternatively (Milner): In a general situation

Our Translation (≈ Abramsky): $\sum \pi(S_1 P_1, \ldots, S_n P_n)$

Suggestion: $\| \pi(S_1 P_1 \ldots S_n P_n)$

Reason: In $\sum \pi(s_1 P_1 \ldots s_n P_n)$, any communicat= must choose one of the cuts & throw others away (= 1 Girard slice). In $\| \pi(s_1 P_1 \ldots s_n P_n)$ have all slices available. Is this useful?

Thm: Restricting distributivity one gets analog of previous thm characterizing proof nets:



Generalizations (by factoring π through {slicings(R)}) are valid

No time to show:

Exponentials - similar story, but problems with replicator:

$$!P \equiv !P \| !P$$

Strengthen replicator (or, following abramsky, add μ).

TO DO:

- Code Eval= strategy for BLL
- What about polymorphism (2nd order)?
- Higher-Order Process Calculi? (i.e. passing processes, not just names)

Remarks on Pi Calculus and Linear Logic

(Comments on Abramsky's Process Interpretations)

Work in Progress

P. J. Scott
Edinburgh (LFCS)

(with G. Bellin,)
LFCS

The paradigm of interaction (short version)

Yves LAFONT
Laboratoire d'Informatique
Ecole Normale Supérieure *
(URA 1327 du CNRS)

July 12, 1991

Abstract

We present a unified framework subsuming well-known models of sequential or parallel computation, such as Turing machines and cellular automata. We propose a notion of *natural encoding* motivated by issues in implementation and partial evaluation. The most remarkable feature is the Girard-Danos-Regnier criterion for deadlock-free computation. This theory is a byproduct of linear logic.

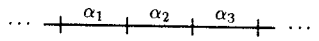
keywords: *abstract machine, deadlock, distributed computation, graph rewriting, interaction net, proof net, string rewriting.*

1 Interaction strings

We try to give a mathematical status to the notion of computational process. We are interested in the structure of computations, not only in computability or complexity, so we cannot content ourselves with traditional models like Turing machines, which are too restrictive. We insist that computations should be some kind of concrete geometric objects following some kind of physical laws, and we adopt the following basic principles:

1. A computation should be *discrete* in space as well as in time (no infinitesimal).
2. It should be *deterministic* (no arbitrary choice).
3. Interaction should be *local* (no broadcasting).
4. The whole process should be *asynchronous* (no absolute time).

In order to separate difficulties, we shall first consider the case of a computation living in a 1-dimensional (acyclic) space. By principle 1, its state at a given instant can be decomposed into atoms:



*Address: 45 rue d'Ulm, 75230 Paris Cedex 05, France. Email: lafont@dmi.ens.fr

Let Σ be the alphabet of all possible atoms. An element of Σ is called a *symbol*. so that an atom is just an occurrence of symbol. The rules of interaction must be of the form

$$\alpha_1 \dots \alpha_p \Rightarrow \beta_1 \dots \beta_q$$

where the *antecedent* $\alpha_1 \dots \alpha_p$ and the *consequent* $\beta_1 \dots \beta_q$ are *strings* in Σ^*.

Assume the antecedent of a rule has length 3:

$$\alpha_1\alpha_2\alpha_3 \Rightarrow \beta_1 \dots \beta_q.$$

Since α_1 and α_3 do not touch each other, they must interact through α_2. Now, even if the whole computation is parallel, each atom should have a sequential behaviour. For example, α_2 should interact with α_1 first, so that the above rule splits into more primitive ones, using an auxiliary symbol γ:

$$\alpha_1\alpha_2 \Rightarrow \gamma, \qquad \gamma\alpha_3 \Rightarrow \beta_1 \dots \beta_q.$$

Any rule with an antecedent of length > 2 can be decomposed in this way. Now, assume the antecedent is atomic:

$$\alpha_1 \Rightarrow \beta_1 \dots \beta_q.$$

This means that α_1 is *unstable*. We cannot allow that α_1 occurs again in the consequent $\beta_1 \dots \beta_q$ because it could never reach a stable state. So α_1 should be removed from the alphabet and replaced everywhere by $\beta_1 \dots \beta_q$. Finally, we exclude empty antecedents for similar reasons and we come down to *binary* rules:

$$\alpha_1\alpha_2 \Rightarrow \beta_1 \dots \beta_q.$$

Assume there are *overlapping* antecedents of the form $\alpha_1\alpha_2$ and $\alpha_2\alpha_3$ respectively. Then the atom α_2 is embarrassed when it has to choose between α_1 and α_3. To prevent this kind of conflict in a way that is compatible with principles 3 and 4, we have to split the alphabet in two disjoint classes:

$$\Sigma = \overleftarrow{\Sigma} \cup \overrightarrow{\Sigma}$$

The members of $\overleftarrow{\Sigma}$ (*resp.* $\overrightarrow{\Sigma}$) are called *left* (*resp. right*) symbols: they can only interact on the left (*resp.* on the right). Therefore, any antecedent must be a *cut*, *i.e.* a string of the form $\overrightarrow{\rho}\,\overleftarrow{\lambda}$ where $\overrightarrow{\rho}$ is a right symbol and $\overleftarrow{\lambda}$ is a left one. We put arrows on top of symbols to indicate their class.

Assume the consequent of a rule contains a cut. If this cut is the antecedent of another rule, it should be replaced by the corresponding consequent. Otherwise, it would stay forever without any possibility of interaction. Therefore we shall also require that consequents are *cut-free*, *i.e.* of the form $\overleftarrow{\lambda_1 \dots \lambda_p}\,\overrightarrow{\rho_1 \dots \rho_q}$.

All those considerations lead us to the following notion:

Definition: A (1-dimensional) *interaction system* consists of a finite alphabet $\Sigma = \overleftarrow{\Sigma} \cup \overrightarrow{\Sigma}$ (disjoint union) and a finite set $\mathcal{R}$ of rules of the form

$$\overrightarrow{\rho}\,\overleftarrow{\lambda} \Rightarrow \overleftarrow{\lambda_1 \dots \lambda_p}\,\overrightarrow{\rho_1 \dots \rho_q},$$

with *at most* one such rule for each pair $(\overrightarrow{\rho}, \overleftarrow{\lambda})$.

Example (double)

We first give a system computing $n \mapsto 2n$ in unary arithmetics. For that purpose we take $\Sigma = \{\overleftarrow{\mathrm{S}}, \overleftarrow{0}, \overrightarrow{2}\}$ and $\mathcal{R}$ consisting of the following rules:

$$\overrightarrow{2}\,\overleftarrow{\mathrm{S}} \Rightarrow \overleftarrow{\mathrm{SS}}\,\overrightarrow{2}, \qquad \overrightarrow{2}\,\overleftarrow{0} \Rightarrow \overleftarrow{0}.$$

Here is the computation $2 \times 2 = 4$:

$$\overrightarrow{2}\,\overleftarrow{\mathrm{SS0}} \Rightarrow \overleftarrow{\mathrm{SS}}\,\overrightarrow{2}\,\overleftarrow{\mathrm{S0}} \Rightarrow \overleftarrow{\mathrm{SSS}}\,\overrightarrow{2}\,\overleftarrow{0} \Rightarrow \overleftarrow{\mathrm{SSSS0}}.$$

Example (half)

We would like to write the following program for division by 2:

$$\overrightarrow{\mathrm{H}}\,\overleftarrow{\mathrm{SS}} \Rightarrow \overleftarrow{\mathrm{S}}\,\overrightarrow{\mathrm{H}}, \qquad \overrightarrow{\mathrm{H}}\,\overleftarrow{\mathrm{S0}} \Rightarrow \overleftarrow{0}, \qquad \overrightarrow{\mathrm{H}}\,\overleftarrow{0} \Rightarrow \overleftarrow{0}.$$

Unfortunately, the two first rules are ternary, so we must introduce an auxiliary symbol, taking $\Sigma = \{\overleftarrow{\mathrm{S}}, \overleftarrow{0}, \overrightarrow{\mathrm{H}}, \overrightarrow{\mathrm{H}'}\}$ and $\mathcal{R}$ consisting of the following rules:

$$\overrightarrow{\mathrm{H}}\,\overleftarrow{\mathrm{S}} \Rightarrow \overrightarrow{\mathrm{H}'}, \qquad \overrightarrow{\mathrm{H}}\,\overleftarrow{0} \Rightarrow \overleftarrow{0} \qquad \overrightarrow{\mathrm{H}'}\,\overleftarrow{\mathrm{S}} \Rightarrow \overleftarrow{\mathrm{S}}\,\overrightarrow{\mathrm{H}}, \qquad \overrightarrow{\mathrm{H}'}\,\overleftarrow{0} \Rightarrow \overleftarrow{0}.$$

Here is the computation $4 \div 2 = 2$:

$$\overrightarrow{\mathrm{H}}\,\overleftarrow{\mathrm{SSSS0}} \Rightarrow \overrightarrow{\mathrm{H}'}\,\overleftarrow{\mathrm{SSS0}} \Rightarrow \overleftarrow{\mathrm{S}}\,\overrightarrow{\mathrm{H}}\,\overleftarrow{\mathrm{SS0}} \Rightarrow \overleftarrow{\mathrm{S}}\,\overrightarrow{\mathrm{H}'}\,\overleftarrow{\mathrm{S0}} \Rightarrow \overleftarrow{\mathrm{SS}}\,\overrightarrow{\mathrm{H}}\,\overleftarrow{0} \Rightarrow \overleftarrow{\mathrm{SS0}}.$$

Example (explosion)

The simplest nonterminating system is given by $\Sigma = \{\overleftarrow{\mathrm{S}}, \overleftarrow{0}, \overrightarrow{\mathrm{F}}\}$ and $\mathcal{R}$ consisting of the following rules:

$$\overrightarrow{\mathrm{F}}\,\overleftarrow{\mathrm{S}} \Rightarrow \overrightarrow{\mathrm{FF}}. \qquad \overrightarrow{\mathrm{F}}\,\overleftarrow{0} \Rightarrow \overleftarrow{\mathrm{S0}}.$$

Here is an infinite computation:

$$\overrightarrow{\mathrm{F}}\,\overleftarrow{\mathrm{S0}} \Rightarrow \overrightarrow{\mathrm{FF}}\,\overleftarrow{0} \Rightarrow \overrightarrow{\mathrm{F}}\,\overleftarrow{\mathrm{S0}} \Rightarrow \overrightarrow{\mathrm{FF}}\,\overleftarrow{0} \Rightarrow \overrightarrow{\mathrm{F}}\,\overleftarrow{\mathrm{S0}} \Rightarrow \dots$$

If we replace the first rule by $\overrightarrow{\mathrm{F}}\,\overleftarrow{\mathrm{S}} \Rightarrow \overleftarrow{\mathrm{S}}\,\overrightarrow{\mathrm{FF}}$, we get an explosion:

$$\overrightarrow{\mathrm{F}}\,\overleftarrow{\mathrm{S0}} \Rightarrow \overleftarrow{\mathrm{S}}\,\overrightarrow{\mathrm{FF}}\,\overleftarrow{0} \Rightarrow \overleftarrow{\mathrm{S}}\,\overrightarrow{\mathrm{F}}\,\overleftarrow{\mathrm{S0}} \Rightarrow \overleftarrow{\mathrm{SS}}\,\overrightarrow{\mathrm{FF}}\,\overleftarrow{0} \Rightarrow \overleftarrow{\mathrm{SS}}\,\overrightarrow{\mathrm{F}}\,\overleftarrow{\mathrm{S0}} \Rightarrow \dots$$

As usual, we write $U \Rightarrow^* V$ if there is $n \in \mathbf{N}$ and $U = U_0, U_1, \dots, U_{n-1}, U_n = V$ such that $U_0 \Rightarrow U_1 \Rightarrow \cdots \Rightarrow U_{n-1} \Rightarrow U_n$. The deterministic nature of the computations is expressed by the *confluence property*:

Proposition 1 *If $T \Rightarrow^* U$ and $T \Rightarrow^* V$, there is W such that $U \Rightarrow^* W$ and $V \Rightarrow^* W$:*

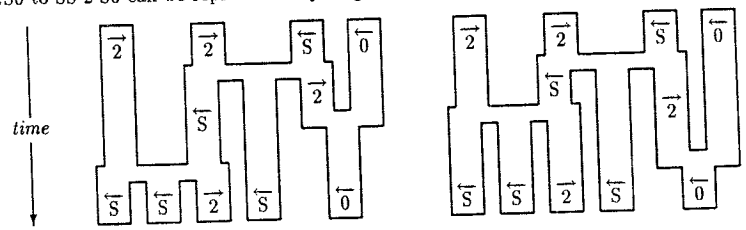
$$\begin{array}{ccc} T & \Rightarrow^* & V \\ \Downarrow_* & & \Downarrow_* \\ U & \Rightarrow^* & W \end{array}$$

Furthermore, the total number of computation steps does not depend on the strategy.

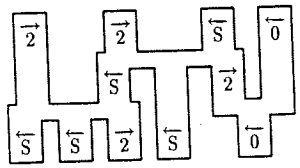
This is an obvious consequence of the absence of conflict. For example, in our first system, we have the following computations:

$$\overrightarrow{22}\overleftarrow{S0} \Rightarrow \overrightarrow{2}\,\overleftarrow{SS}\,\overrightarrow{2}\,\overleftarrow{0} \Rightarrow \overrightarrow{2}\,\overleftarrow{SS0}, \qquad \overrightarrow{22}\overleftarrow{S0} \Rightarrow \overrightarrow{2}\,\overleftarrow{SS}\,\overrightarrow{2}\,\overleftarrow{0} \Rightarrow \overleftarrow{SS}\,\overrightarrow{2}\,\overleftarrow{S}\,\overrightarrow{2}\,\overleftarrow{0}.$$

Both $\overrightarrow{2}\,\overleftarrow{SS0}$ and $\overleftarrow{SS}\,\overrightarrow{2}\,\overleftarrow{S}\,\overrightarrow{2}\,\overleftarrow{0}$ reduce in one step to $\overleftarrow{SS}\,\overrightarrow{2}\,\overleftarrow{S0}$. The two possible paths from $\overrightarrow{22}\overleftarrow{S0}$ to $\overleftarrow{SS}\,\overrightarrow{2}\,\overleftarrow{S0}$ can be represented by diagrams



which differ only by the relative order of the last two interactions. This order is irrelevant by our principle of asynchrony. In fact, they could also happen simultaneously:



But the first one must happen ... first!

Finally, we prove a simple termination result:

Proposition 2 *(exponential time)*

Let $(\Sigma, \mathcal{R})$ be a system where all consequents are of the form $\overleftarrow{\lambda_1 \dots \lambda_p}$ or $\overleftarrow{\lambda_1 \dots \lambda_p}\,\overrightarrow{\rho_1}$. Then all computations terminate in at most exponential time.

Proof: Let k be the maximal value of p (number of left atoms in the consequent) for all rules in $\mathcal{R}$. We define the *complexity* $|W|$ of a string $W \in \Sigma^*$ as follows:

$$|\overleftarrow{\lambda} W| = |W| + 1, \qquad |\overrightarrow{\rho} W| = (k+1)|W|, \qquad |\epsilon| = 0.$$

(ϵ is a notation for the empty string.)

One checks easily that $|U| > |V|$ whenever $U \Rightarrow V$. So, if W has length n, a computation starting from W terminates in N steps, where $N \leq |W| \leq \max n\,(k+1)^n$. **Q.e.d.**

2 Machines

Now we investigate models of deterministic computation in terms of interaction.

A *finite automaton* is given by a finite alphabet $\mathcal{A}$ and a finite set of states $\mathcal{Q}$ together with a (partial) transition function $\delta : \mathcal{Q} \times \mathcal{A} \rightharpoonup \mathcal{Q}$. The transition $\delta(q,a) = q'$ means

that, reading the symbol a in state q, the automaton will go to state q' and read the next symbol. This corresponds to an interaction system $(\Sigma, \mathcal{R})$ where

$$\Sigma = \{\overleftarrow{a} \mid a \in \mathcal{A}\} \cup \{\overrightarrow{q} \mid q \in \mathcal{Q}\}$$

and $\mathcal{R}$ consists of the following rules:

$$\overrightarrow{q}\,\overleftarrow{a} \Rightarrow \overrightarrow{q'} \text{ whenever } \delta(q,a) = q'.$$

Clearly, $\overrightarrow{q\,a_1 \dots a_n} \Rightarrow^* \overrightarrow{q'}$ means that, from initial state q, our automaton will reach state q' after scanning the string $a_1 \dots a_n$.

A *Turing machine* is also given by finite sets $\mathcal{A}$ and $\mathcal{Q}$, but with $\delta : \mathcal{Q} \times \mathcal{A} \rightharpoonup \mathcal{A} \times \{+,-\} \times \mathcal{Q}$. Now $\delta(q,a) = (a', \pm, q')$ means that, reading a in state q, the machine will write a', move forwards ($+$ case) or backwards ($-$ case) and go to state q'. This corresponds to an interaction system $(\Sigma, \mathcal{R})$ where

$$\Sigma = \{\overleftarrow{a} \mid a \in \mathcal{A}\} \cup \{\overleftarrow{q} \mid q \in \mathcal{Q}\} \cup \{\overrightarrow{a} \mid a \in \mathcal{A}\} \cup \{\overrightarrow{q} \mid q \in \mathcal{Q}\}$$

and $\mathcal{R}$ consists of the following rules:

$$\begin{cases} \overrightarrow{q}\,\overleftarrow{a} \Rightarrow \overleftarrow{a'}\,\overrightarrow{q'} \\ \overrightarrow{a}\,\overleftarrow{q} \Rightarrow \overleftarrow{a'}\,\overrightarrow{q'} \end{cases} \text{ if } \delta(q,a) = (a',+,q'), \qquad \begin{cases} \overrightarrow{q}\,\overleftarrow{a} \Rightarrow \overleftarrow{q'}\,\overrightarrow{a'} \\ \overrightarrow{a}\,\overleftarrow{q} \Rightarrow \overleftarrow{q'}\,\overrightarrow{a'} \end{cases} \text{ if } \delta(q,a) = (a',-,q').$$

The configuration of figure 1 is represented by

$$\overrightarrow{\dots a_{-2}a_{-1}q}\,\overleftarrow{a_0a_1a_2\dots} \quad \text{or} \quad \overrightarrow{\dots a_{-2}a_{-1}a_0}\,\overleftarrow{qa_1a_2\dots}$$

depending whether the head is moving forwards or backwards. Here we have no special rule for *blank*, so that, in order to emulate a Turing machine with an infinite tape, we should start with an infinite string. But never mind, we shall not have this problem in the next case.

A *SCD machine* (*SCD* for *Stack*, *Code* and *Dump*) can be described by finite sets $\mathcal{A}$ and $\mathcal{Q}$ with $\delta : \mathcal{Q} \times \mathcal{A} \rightharpoonup (\mathcal{Q}^* \times \mathcal{Q}) \cup \mathcal{A}^*$ (disjoint union). This machine has a *stack* for data and a *dump*, *i.e.* another stack for return addresses. Here $\delta(q,a) = (q_1 \dots q_n, q')$ means that, after popping a from the stack in state q, the machine will push $q_1, \dots, q_n$ on the dump and go to state q', but if $\delta(q,a) = a_1 \dots a_n$, it will push $a_1, \dots, a_n$ on the stack and pop the state from the dump. This corresponds to an interaction system $(\Sigma, \mathcal{R})$ where

$$\Sigma = \{\overleftarrow{a} \mid a \in \mathcal{A}\} \cup \{\overrightarrow{q} \mid q \in \mathcal{Q}\}$$

and $\mathcal{R}$ consists of the following rules:

$$\begin{aligned} &\overrightarrow{q}\,\overleftarrow{a} \Rightarrow \overrightarrow{q_n \dots q_1 q'} \text{ if } \delta(q,a) = (q_1 \dots q_n, q'), \\ &\overrightarrow{q}\,\overleftarrow{a} \Rightarrow \overleftarrow{a_1 \dots a_n} \text{ if } \delta(q,a) = a_1 \dots a_n. \end{aligned}$$

The configuration of figure 2 is represented by $\overrightarrow{\dots q_2q_1q}\,\overleftarrow{a_0a_1a_2\dots}$

A (1-dimensional) *synchronous cellular automaton*[1] is given by $\mathcal{Q}$ and $\delta : \mathcal{Q} \times \mathcal{Q} \times \mathcal{Q} \rightharpoonup \mathcal{Q}$. Each *cell* has a left and a right neighbour: $\delta(p,q,r) = q'$ means that if a cell is in state q,

[1] Usually, this is simply called a *cellular automaton*, but we shall introduce a notion of *asynchronous cellular automaton*.

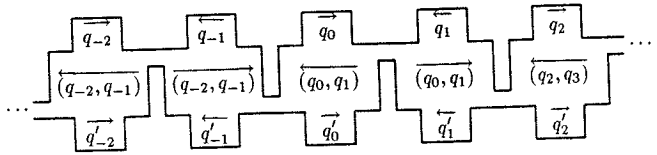
with its left neighbour in state p and its right neighbour in state r, then it will go to state q'. Of course, it is understood that all cells change *at the same time*, which seems inconsistent with our principle of asynchrony. But never mind, following [Margolus], we decompose transitions so that each cell looks alternatively at each of his neighbours. More precisely, we take

$$\Sigma = \{\overleftarrow{q} \mid q \in \mathcal{Q}\} \cup \{\overleftarrow{(p,q)} \mid p,q \in \mathcal{Q}\} \cup \{\overrightarrow{q} \mid q \in \mathcal{Q}\} \cup \{\overrightarrow{(p,q)} \mid p,q \in \mathcal{Q}\}$$

and $\mathcal{R}$ consisting of the following rules:

$$\overrightarrow{p}\,\overleftarrow{q} \Rightarrow \overleftarrow{(p,q)}\overrightarrow{(p,q)}, \qquad \overrightarrow{(p,q)}\overleftarrow{(r,s)} \Rightarrow \overleftarrow{q'}\,\overrightarrow{r'} \text{ if } \delta(p,q,r) = q' \text{ and } \delta(q,r,s) = r'.$$

The configuration of figure 3 is represented by $\ldots \overrightarrow{q_{-2}}\overleftarrow{q_{-1}}\overrightarrow{q_0}\overleftarrow{q_1}\overrightarrow{q_2} \ldots$ and a global transition of the automaton corresponds to a computation of the following form:



In fact, we have just shown that a synchronous cellular automaton can be emulated by an asynchronous one.

An *asynchronous cellular automaton* is indeed given by two disjoint sets $\mathcal{Q}^-$ and $\mathcal{Q}^+$ together with $\delta : \mathcal{Q}^+ \times \mathcal{Q}^- \to (\mathcal{Q}^- \times \mathcal{Q}^-) \cup (\mathcal{Q}^+ \times \mathcal{Q}^+) \cup (\mathcal{Q}^- \times \mathcal{Q}^+)$. The difference with the synchronous case is that each cell is looking at one of its neighbours, and a transition only happens when two cells are facing each other. This corresponds to an interaction system $(\Sigma, \mathcal{R})$ where

$$\Sigma = \{\overleftarrow{q} \mid q \in \mathcal{Q}^-\} \cup \{\overrightarrow{q} \mid q \in \mathcal{Q}^+\}$$

and $\mathcal{R}$ consists of the following rules:

$$\begin{aligned}
\overrightarrow{p}\,\overleftarrow{q} &\Rightarrow \overleftarrow{p'q'} \quad \text{if } \delta(p,q) = (p',q') \in \mathcal{Q}^- \times \mathcal{Q}^-,\\
\overrightarrow{p}\,\overleftarrow{q} &\Rightarrow \overrightarrow{p'q'} \quad \text{if } \delta(p,q) = (p',q') \in \mathcal{Q}^+ \times \mathcal{Q}^+,\\
\overrightarrow{p}\,\overleftarrow{q} &\Rightarrow \overleftarrow{p'}\,\overrightarrow{q'} \quad \text{if } \delta(p,q) = (p',q') \in \mathcal{Q}^- \times \mathcal{Q}^+.
\end{aligned}$$

The configuration of figure 3 is represented by $\ldots \overrightarrow{q_{-2}}\overleftarrow{q_{-1}}\overrightarrow{q_0}\overleftarrow{q_1}\overrightarrow{q_2} \ldots$

Each model of computation considered in this section corresponds to some restriction on consequents in the interaction rules (figure 5).

In the case of SCD machines, consequents are *homogeneous*, *i.e.* they consist only of left atoms or only of right atoms. In general, an interaction may generate two cuts, as in

$$\overrightarrow{2}\overrightarrow{2}\overleftarrow{S}\overleftarrow{0} \Rightarrow \overrightarrow{2}\,\overleftarrow{S}\overleftarrow{S}\,\overrightarrow{2}\,\overleftarrow{0},$$

but not if the consequent of the rule is homogeneous. Furthermore, if $U \Rightarrow V$ by such a rule and U is of the form $\overrightarrow{\rho_1 \ldots \rho_p}\overleftarrow{\lambda_1 \ldots \lambda_q}$, so is V. Homogeneous consequents modelise *sequential interaction*.

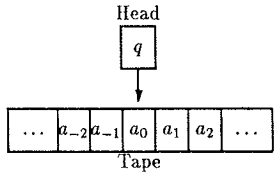


Figure 1: Turing machine

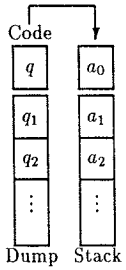


Figure 2: SCD machine

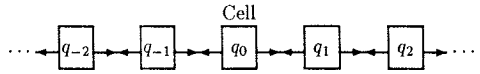


Figure 3: synchronous cellular automaton

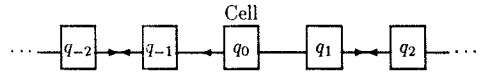


Figure 4: asynchronous cellular automaton

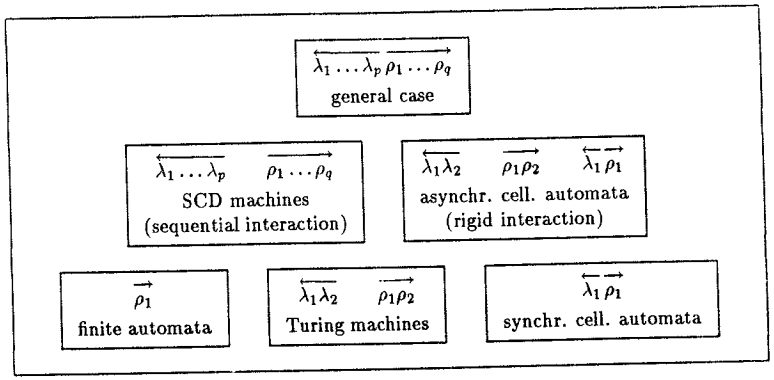


Figure 5: little taxonomy of interaction

In the case of asynchronous cellular automata, consequents have length 2. So, if for example $\overrightarrow{\rho}\,\overleftarrow{\lambda} \Rightarrow \overleftarrow{\lambda_1\lambda_2}$, we can say that $\overrightarrow{\rho}$ *becomes* $\overleftarrow{\lambda_1}$ and $\overleftarrow{\lambda}$ becomes $\overleftarrow{\lambda_2}$. Consequents of length 2 modelise *rigid interaction*. Turing machines correspond to the case where interaction is both sequential and rigid.

3 Natural encoding

A *universal Turing machine* $\mathcal{M}$ has the property that any other Turing machine can be simulated by $\mathcal{M}$. However, this simulation does not preserve the structure of computations. We shall need a stronger notion of encoding:

Let $(\Sigma, \mathcal{R})$ and $(\Sigma', \mathcal{R}')$ be two interaction systems, and let Φ be a map from Σ to Σ'^*. Then Φ extends to Σ^* in an obvious way.

Definition: Φ is a *natural encoding* of $(\Sigma, \mathcal{R})$ into $(\Sigma', \mathcal{R}')$ if it is compatible with orientation and with interaction in the following sense:

- $\Phi(\overleftarrow{\Sigma}) \subset \overleftarrow{\Sigma'}^*$ and $\Phi(\overrightarrow{\Sigma}) \subset \overrightarrow{\Sigma'}^*$,
- $\Phi(\overrightarrow{\rho}\,\overleftarrow{\lambda}) \Rightarrow^* \Phi(\overleftarrow{\lambda_1 \dots \lambda_p}\,\overrightarrow{\rho_1 \dots \rho_q})$ whenever $\overrightarrow{\rho}\,\overleftarrow{\lambda} \Rightarrow \overleftarrow{\lambda_1 \dots \lambda_p}\,\overrightarrow{\rho_1 \dots \rho_q}$ in $(\Sigma, \mathcal{R})$.

Note that a *symbol* is translated into a *string* and a *rule* into a *computation*. Since Φ preserves orientation, cut-free strings are translated into cut-free strings.

Example (quadruple)

Consider $\Sigma = \{\overleftarrow{S}, \overleftarrow{0}, \overrightarrow{4}\}$ and $\mathcal{R}$ consisting of the following rules:

$$\overrightarrow{4}\,\overleftarrow{S} \Rightarrow \overleftarrow{SSSS}\,\overleftarrow{4} \qquad \overrightarrow{4}\,\overleftarrow{0} \Rightarrow \overleftarrow{0}.$$

Now let $(\Sigma', \mathcal{R}')$ be the system of section 1 for doubling, and consider Φ given by:

$$\Phi(\overleftarrow{S}) = \overleftarrow{S}, \qquad \Phi(\overleftarrow{0}) = \overleftarrow{0}, \qquad \Phi(\overrightarrow{4}) = \overrightarrow{22}.$$

Since $\overrightarrow{22}\,\overleftarrow{S} \Rightarrow^* \overleftarrow{SSSS}\,\overrightarrow{22}$ and $\overrightarrow{22}\,\overleftarrow{0} \Rightarrow^* \overleftarrow{0}$, Φ is indeed a natural encoding.

Example (degenerate)

For any two systems $(\Sigma, \mathcal{R})$ and $(\Sigma', \mathcal{R}')$, there is always a natural encoding mapping all symbols in Σ to the empty string. To avoid this kind of degeneracy, we shall consider *injective encodings*, *i.e.* such that Φ defines a one-to-one map from Σ^* to Σ'^*.

Proposition 3 *(number of steps)*

Let Φ be a natural encoding. If $U \Rightarrow^* V$ in $(\Sigma, \mathcal{R})$, then $\Phi(U) \Rightarrow^* \Phi(V)$ in $(\Sigma', \mathcal{R}')$. Moreover, there is a constant k such that, if the first computation takes n steps, the second one takes at most kn steps, and if Φ is injective, it takes at least n steps.

In particular, if Φ is injective and $(\Sigma', \mathcal{R}')$ terminates, so does $(\Sigma, \mathcal{R})$.

Proof: Each rule in $(\Sigma, \mathcal{R})$ corresponds indeed to a computation in $(\Sigma', \mathcal{R}')$, and there is only a finite number of them. The constant k is the maximal length of those computations. If Φ is injective and $\overrightarrow{\rho}\,\overleftarrow{\lambda} \Rightarrow W$ is a rule in $\mathcal{R}$, then $\Phi(\overrightarrow{\rho}\,\overleftarrow{\lambda})$ is of the form $\overrightarrow{\rho_1 \dots \rho_p}\,\overleftarrow{\lambda_1 \dots \lambda_q}$ with $p, q > 0$, whereas $\Phi(W)$ is cut-free. So $\Phi(\overrightarrow{\rho}\,\overleftarrow{\lambda}) \Rightarrow^* \Phi(W)$ in at least one step, and more generally, computations take longer in $(\Sigma', \mathcal{R}')$. **Q.e.d.**

Now we introduce the second crucial notion of this section:

Definition: A class $\mathcal{C}$ of interaction systems is *universal* if, for any other system $(\Sigma, \mathcal{R})$, there is an *injective* natural encoding of $(\Sigma, \mathcal{R})$ into an element of $\mathcal{C}$. In particular, when $\mathcal{C}$ is a singleton, we call it a *universal system*.

Proposition 4 *(two left symbols)*

The class of systems with two left symbols is universal.

By symmetry, the class of systems with two right symbols is also universal, but this does not mean that the class of systems with two left symbols *and* two right symbols is universal.

Proof: The idea consists in encoding left symbols by numerals. More precisely, a system $(\Sigma, \mathcal{R})$ where $\overleftarrow{\Sigma} = \{\overleftarrow{\lambda_0}, \dots \overleftarrow{\lambda_n}\}$ is encoded into a system $(\Sigma', \mathcal{R}')$ where

$$\Sigma' = \{\overleftarrow{S}, \overleftarrow{0}\} \cup \{(\overrightarrow{\rho}, i) \mid \overrightarrow{\rho} \in \overrightarrow{\Sigma} \text{ and } 0 \le i \le n\}.$$

So, in general $\overrightarrow{\Sigma'}$ is larger than $\overrightarrow{\Sigma}$. The encoding is defined as follows:

$$\Phi(\overleftarrow{\lambda_i}) = \overleftarrow{\underbrace{S \dots S}_{i \text{ times}}\,0}, \qquad \Phi(\overrightarrow{\rho}) = \overrightarrow{(\rho, 0)}.$$

The rules of $\mathcal{R}'$ are $\begin{cases} \overrightarrow{(\rho, i)}\,\overleftarrow{S} \Rightarrow \overrightarrow{(\rho, i+1)} & \text{for each } \overrightarrow{\rho} \in \overrightarrow{\Sigma} \text{ and } 0 \le i < n, \\ \overrightarrow{(\rho, i)}\,\overleftarrow{0} \Rightarrow \Phi(W) & \text{for each rule } \overrightarrow{\rho}\,\overleftarrow{\lambda_i} \Rightarrow W \text{ in } \mathcal{R}. \end{cases}$

By construction, Φ is indeed an injective encoding. **Q.e.d.**

Proposition 5 *(short consequents)*

The class of systems with consequents of length at most 3 is universal. More precisely, consequents may be restricted to the following forms:

$$\overleftarrow{\lambda_1}\overrightarrow{\lambda_2\rho_1}, \qquad \overleftarrow{\lambda_1}\overrightarrow{\rho_1\rho_2}, \qquad \overleftarrow{\lambda_1}, \qquad \overrightarrow{\rho_1}, \qquad \epsilon.$$

Proof: The idea consists in encoding all symbols by strings of length 2. In order to separate the crucial combinatorial argument from the tedious details of book-keeping, we shall tackle an abstract version of the problem where all left symbols (*resp.* all right symbols) are identified.

Consider the *nondeterministic* system consisting of the following rules:

$$\overrightarrow{\rho}\overleftarrow{\lambda} \Rightarrow \overleftarrow{\lambda}\overrightarrow{\lambda\rho}, \qquad \overrightarrow{\rho}\overleftarrow{\lambda} \Rightarrow \overleftarrow{\lambda}\overrightarrow{\rho\rho}, \qquad \overrightarrow{\rho}\overleftarrow{\lambda} \Rightarrow \overleftarrow{\lambda}, \qquad \overrightarrow{\rho}\overleftarrow{\lambda} \Rightarrow \overrightarrow{\rho}, \qquad \overrightarrow{\rho}\overleftarrow{\lambda} \Rightarrow \epsilon.$$

The following computations are possible:

$$\overrightarrow{\rho\rho}\overleftarrow{\lambda\lambda} \Rightarrow^* \epsilon, \qquad \overrightarrow{\rho\rho}\overleftarrow{\lambda\lambda} \Rightarrow^* \overleftarrow{\lambda}\overrightarrow{\rho\rho}\overleftarrow{\lambda\lambda}, \qquad \overrightarrow{\rho\rho}\overleftarrow{\lambda\lambda} \Rightarrow^* \overrightarrow{\rho\rho}\overleftarrow{\lambda\lambda}\overrightarrow{\rho}.$$

For example the second one is $\overrightarrow{\rho\rho}\overleftarrow{\lambda\lambda} \Rightarrow \overrightarrow{\rho}\overleftarrow{\lambda\lambda}\overrightarrow{\rho}\overleftarrow{\lambda} \Rightarrow \overleftarrow{\lambda}\overrightarrow{\rho\rho}\overleftarrow{\lambda}\overrightarrow{\rho}\overleftarrow{\lambda} \Rightarrow \overleftarrow{\lambda}\overrightarrow{\rho\rho}\overleftarrow{\lambda\lambda}$. By iterating those computations, we get $\overrightarrow{\rho\rho}\overleftarrow{\lambda\lambda} \Rightarrow^* \overleftarrow{\lambda\ldots\lambda\rho\ldots\rho}$ with arbitrary numbers of λ and ρ.

Now, it is not hard to adapt this argument to the encoding of any (deterministic) system into another one with consequents of the required form. The alphabet must be enriched, as in the proof of proposition 4, but it is possible to keep the same left alphabet. **Q.e.d.**

Propositions 4 and 5 are compatible in the sense that the intersection of the two classes is also universal. Indeed, any system $(\Sigma, \mathcal{R})$ can be encoded into a system $(\Sigma', \mathcal{R}')$ with only two left symbols, which itself can be encoded into a system $(\Sigma'', \mathcal{R}'')$ with the same left alphabet and with consequents of the required form.

Proposition 4 is clearly optimal: a system with two left symbols cannot be injectively encoded into a system with one (or zero) left symbol! What about proposition 5? The "explosive system" of section 1 cannot be injectively encoded into a system where all consequents have length at most 2, so consequents of length 3 are certainly needed. Similarly, by proposition 2, a nonterminating system cannot be encoded into a system with consequents of the form

$$\overleftarrow{\lambda_1}\overrightarrow{\lambda_2\rho_1}, \qquad \overleftarrow{\lambda_1}, \qquad \overrightarrow{\rho_1}, \qquad \epsilon.$$

So $\overleftarrow{\lambda_1}\overrightarrow{\rho_1\rho_2}$ is needed (and $\overleftarrow{\lambda_1\lambda_2}\overrightarrow{\rho_1}$ by symmetry). Finally, a nonempty string cannot reduce to ϵ if all consequents are nonempty, so the empty consequent is also needed. We do not know whether consequents of the form $\overleftarrow{\lambda_1}$ and $\overrightarrow{\rho_1}$ are really needed here.

Proposition 6 *(genetic machine)*

There is a universal system.

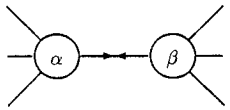
The idea consists in encoding each system by a string. Since the computation is distributed, this *code* must be available about everywhere, hence the analogy with genetic code. The description of such a system is left as an exercise.

4 Interaction nets

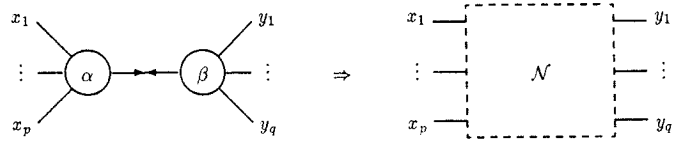
We consider now the general case of computations taking place in an arbitrary space. The difference is that an atom may have more than two neighbours, but again, it can only interact with one of them, through its *principal port*.

Let Σ be a finite set of symbols with *arities* in $\mathbf{N}$. A *net* on Σ is an undirected graph whose vertices are labelled by symbols in Σ, such that if α has arity n, the corresponding vertices have one principal port and n *auxiliary ports*. In fact, we shall often consider *open nets*, whose vertices can also be labelled by *variables*. Each variable must be connected to exactly one other vertex.

A *cut* consists of two atoms connected through their principal ports:



Definition: A (general) *interaction system* consists of a finite alphabet Σ with arities and a finite set $\mathcal{R}$ of rules of the form



where $\mathcal{N}$ is an open cut-free net, and with *at most* one such rule for each pair (α, β).

A similar notion of *connection graph* has been proposed in [Bawden].

Example (unary arithmetics)

Let Σ consist of $0, \varepsilon$ (of arity 0), S (of arity 1) and $+, \times, \delta$ (of arity 2). The rules given in figure 6 correspond to the following definitions:

$$\mathrm{s}x + y = \mathrm{s}(x + y), \qquad 0 + y = y, \qquad \mathrm{s}x \times y = (x \times y) + y, \qquad 0 \times y = 0.$$

The principal port of the symbol + corresponds to the first argument x, because addition is defined by induction on x. For multiplication, there is a little problem: the second argument y is used twice (successor case) or not at all (zero case). That is why we introduce auxiliary agents δ for explicit duplication and ε for explicit erasing. Note that consequents may be cyclic (third rule), disconnected (fourth rule) or empty (last rule). Figure 7 represents the computation $2 \times 2 = 4$.

In the framework of interaction nets, a computation may get stuck for two different reasons: if some cut is created for which no rule applies, or if a *vicious circle* appears (figure 8). The first problem can be eliminated by using an appropriate type discipline. The second one corresponds to *deadlock*, for which a *global correctness criterion*[2] is needed. Of course, we could simply forbid cycles in nets, but then our program for multiplication would be

[2] This is essentially Jean-Yves Girard's long trip condition for proof nets [Girard87] modified by Vincent Danos and Laurent Régnier [DanosRegnier]. Alan Bawden suggested to use this criterion in the case of interaction nets (private communication). It is equivalent to the semi-simplicity condition of [Lafont90].

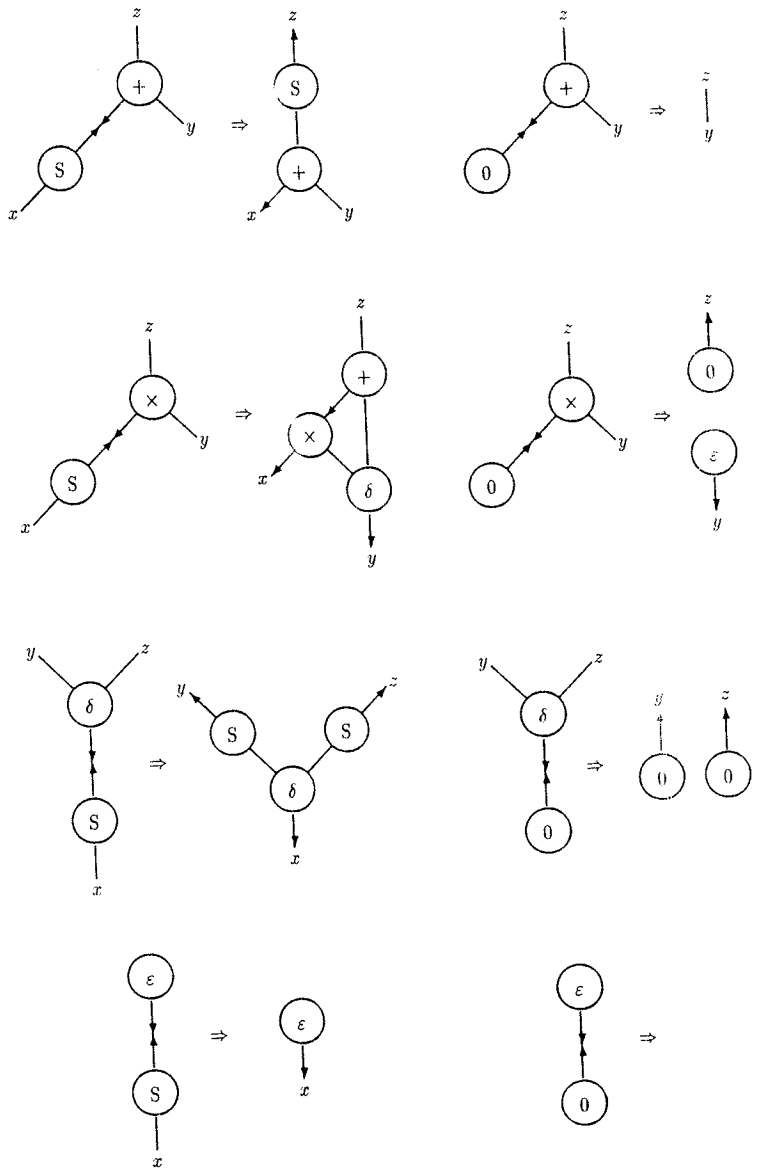


Figure 6: rules for unary arithmetics

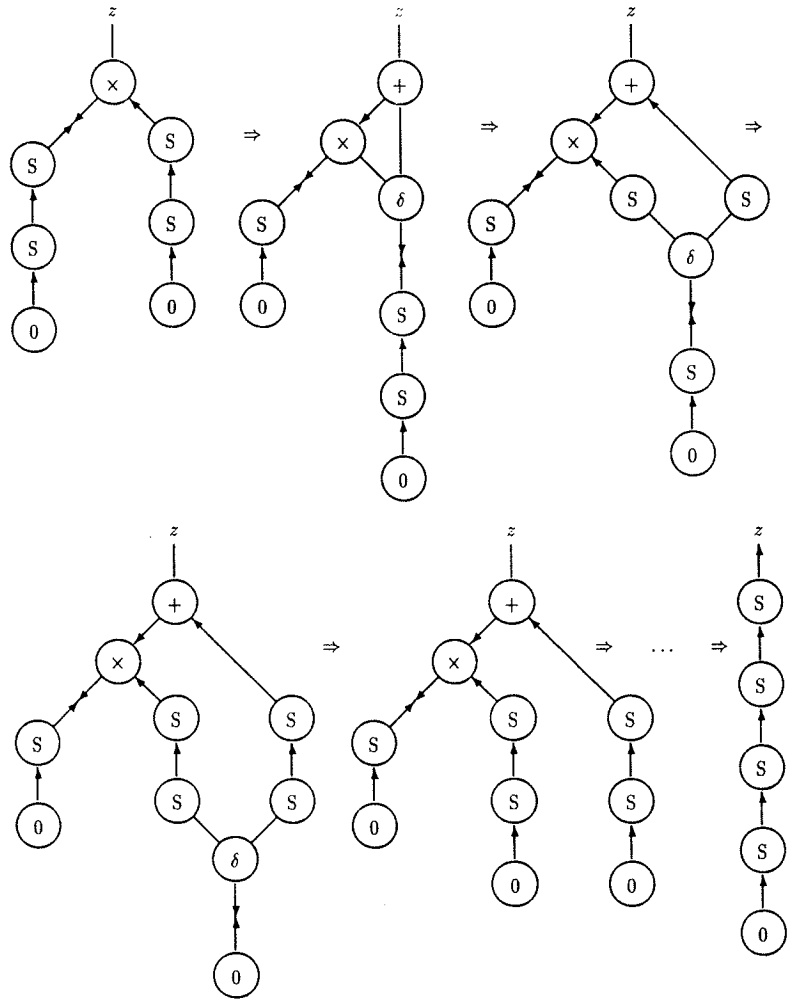


Figure 7: a computation

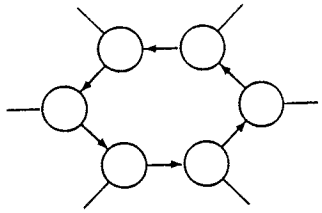


Figure 8: vicious circle (deadlock)

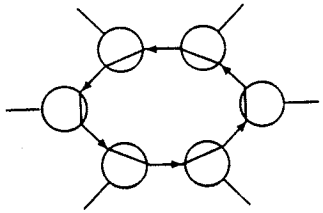
illegal, which is a shame. So we must be more subtle, discriminating between bad and good cycles. To simplify, assume that all symbols have arity 0, 1 or 2. Each symbol of arity 2 must be equipped with one of the following *switches*:

- the 3-switch (with three positions):

- the 2-switch (with only two positions):

For symbols of arity 1, there is only one degenerate switch with one position:

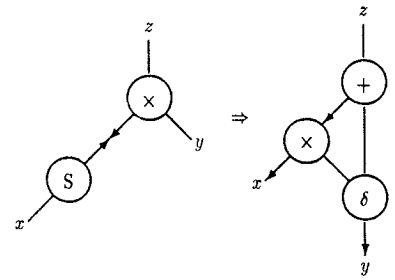
Notice that the principal port can always be connected to any auxiliary port. In particular, for a vicious circle, the following switching is always possible:



Definition: A net is *correct* if no switching creates cycles.

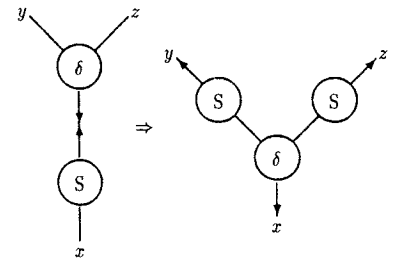
By our previous remark, a correct net contains no deadlock, the converse being obviously false. Furthermore, correctness can be checked in at most exponential time (in fact quadratic, see [Danos,Gallier]).

Consider our system for unary arithmetics, where + and × are equipped with a 3-switch, and δ is equipped with a 2-switch. It is easy to prove that all rules in figure 6 preserve correctness. Take for example:



Assume the antecedent belongs to some correct net $\mathcal{N}$, and choose a switching σ for the rest of $\mathcal{N}$. Clearly, x cannot be connected to y by σ, because it would create a cycle, and similarly, x cannot be connected to z or y to z (figure 9). If we replace the antecedent by the consequent, a cycle can only be created inside the consequent itself, but this is impossible, since δ is equipped with a 2-switch. So the net is still correct after substitution.

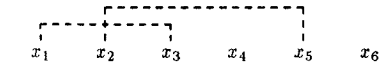
Now take the following rule:



The difference with the previous case is that now, since δ is equipped with a 2-switch, y can be connected to z outside the antecedent. So we must take care of this connection when we check the correctness of the consequent (figure 10).

The other rules are handled similarly. Notice that no other choice would be adequate for switches: for example, if the symbol + were equipped with a 2-switch, one of the rules would be illegal (figure 11).

Now we shall consider the general case of an interaction system equipped with switches. If $X = \{x_1, \ldots, x_n\}$ is a set of variables, a *connection* on X is just a set of disjoint pairs in X. For example, $\{\{x_1, x_3\}, \{x_2, x_5\}\}$ is a connection on $\{x_1, \ldots, x_6\}$:



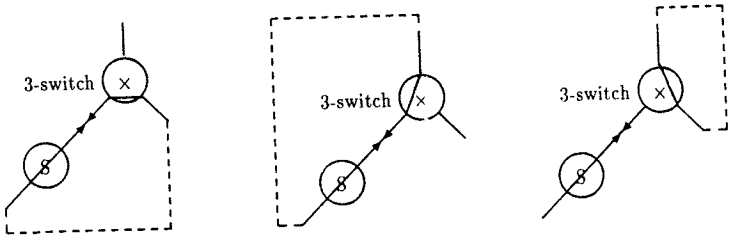


Figure 9: impossible configurations in a correct net

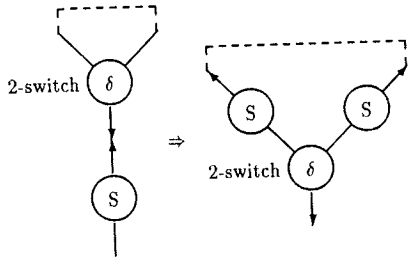


Figure 10: contextual correctness

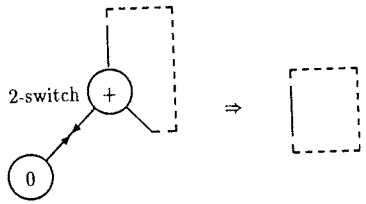


Figure 11: illegal rule due to bad switch

Definition: Let $\mathcal{N}$ be an open net with variables $x_1, \ldots, x_n$. Its *dual* $\mathcal{N}^\perp$ is the set of all connections on $\{x_1, \ldots, x_n\}$ such that no switching creates cycles.

In particular, $\mathcal{N}$ is correct if the empty connection belongs to $\mathcal{N}^\perp$.

Definition: A rule $\mathcal{N} \Rightarrow \mathcal{N}'$ is *legal* if $\mathcal{N}^\perp \subset \mathcal{N}'^\perp$.

It is clear that legal rules preserve correctness, and since correct nets are deadlock-free, we conclude:

Proposition 7 *(deadlock-freeness)*

If all rules are legal, then a correct net will never deadlock.

Of course, it is only a sufficient condition, but the point is that it is a decidable one.

Conclusion

The paradigm of interaction shed new light on many issues in computer science. We can already make some observations:

- The distinction between inputs and outputs is not so essential, but the notion of principal port is crucial.
- In an asynchronous parallel world, sharing and garbage collection must be part of the computation.
- Two kinds of correctness are needed: local (the typing) and global (for deadlock-freeness).

The following points are under development:

- a theory of natural encoding for interaction nets with applications to partial evaluation of programs,
- interaction nets as a general purpose programming language,
- rigid interaction as a model of parallel asynchronous hardware.

References

[Bawden] A. Bawden, Connection Graphs, in: *Proceedings of ACM Conference on Lisp and Functional Programming* (1986) 258-265.

[Danos] V. Danos, La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du λ-calcul), thèse de doctorat, Université de Paris 7 (1990).

[DanosRegnier] V. Danos & L. Régnier, The structure of multiplicatives, Archive for Mathematical Logic **28** (1989) 181-203.

[Gallier] J. Gallier, Constructive Logics. Part II: Linear Logic and Proof Nets, Research Report Digital PRL **8** (1991).

[Girard87] J.Y. Girard, Linear logic, *TCS* **50** (1987) 1-102.

[Girard89] J.Y. Girard, Towards a geometry of interaction, in: *Conference on categories, computer science and logic*, Contemporary Mathematics, *AMS* **92** (1989).

[Lafont90] Y. Lafont, Interaction Nets, in: *Proceedings of Seventeenth ACM Symposium on Principles of Programming Languages* (1990) 95-108.

[Margolus] N. Margolus, Parallel Quantum Computation (1989).

Categorical Abstract Machines for Higher-Order Typed λ-Calculi

Eike Ritter

University of Cambridge

1 Motivation

2 Construction of the Machines

3 Typechecking

4 Conclusions

Why categorical abstract machines?

Categorical combinators provide a mathematical ba

for reasoning abou environment and substitution

- Simple and conceptually clean combinators should lead to a simple and conceptually clean instruction set
- Modular structure:
  Language extension ≙ Additions of machine instructio
- Correctness proof comparatively easy

Construction in three steps

1) Design of categorical combinators
2) Reduction rules + Reduction strategies
3) Implementation

Earlier examples

CAM: Machine for the simply-typed λ-calculus (Curien

PLC-CAM: (Ritter, Stroup)

Problems with the relation CAM-CCC:

1) Composition preserves
   - Environments
   - Terms
2) Composition models both
   - Substitution
   - Function application
3) Categorical products model both
   - Product types
   - Contents

Disadvantages:

1) Transformations for environments not, not apply to terms
   (x: 13, y: 0) ; x+3 ; (x: x+1, y: x)
   (0, arg), eval = 5
2) They should calculate by proofs:
   Modules correctness proof

What can we do?

Solution: Use indexed category structure

Generalizes immediately to dependent types + CC

$A \simeq B$ denotes the conversion relation based on the reduction $\rightsquigarrow$.

Contexts

$$\frac{}{\vdash [] \text{ ctxt}} \qquad \frac{\Gamma \vdash A \text{ type}}{\vdash (\Gamma, x{:}A) \text{ ctxt}}$$

Types

$$\frac{}{\Gamma \vdash 1 \text{ type}} \qquad \frac{(\Gamma, x{:}A) \vdash B \text{ type}}{\Gamma \vdash \Pi x{:}A.B \text{ type}} \qquad \frac{(\Gamma, x{:}A) \vdash B \text{ type}}{\Gamma \vdash \Sigma x{:}A.B \text{ type}}$$

$$\frac{\vdash \Gamma \text{ ctxt}}{\Gamma \vdash \mathsf{Prop} \text{ type}} \qquad \frac{\Gamma \vdash p{:}\mathsf{Prop}}{\Gamma \vdash \mathsf{Proof}(p) \text{ type}}$$

Terms

$$\frac{}{\Gamma \vdash (){:}1} \qquad \frac{(\Gamma, x{:}A) \vdash t{:}B}{\Gamma \vdash (\lambda x{:}A.t){:}\Pi x{:}A.B} \qquad \frac{\Gamma \vdash t{:}\Pi x{:}A.B \quad \Gamma \vdash s{:}A}{\Gamma \vdash ts{:}B[x \leftarrow s]}$$

$$\frac{\vdash (\Gamma, x{:}A, \Gamma') \text{ ctxt}}{(\Gamma, x{:}A, \Gamma') \vdash x{:}A} \qquad \frac{(\Gamma, x{:}A) \vdash p{:}\mathsf{Prop}}{\Gamma \vdash (\forall x{:}A.p){:}\mathsf{Prop}}$$

$$\frac{\Gamma \vdash t_1{:}A \quad (\Gamma, x{:}A) \vdash B \text{ type} \quad \Gamma \vdash t_2{:}B[x \leftarrow t_1]}{\Gamma \vdash \mathsf{Pair}(t_1, t_2, x.B){:}\Sigma x{:}A.B} \qquad \frac{\Gamma \vdash t{:}\Sigma x{:}A.B}{\Gamma \vdash \pi_1(t){:}A}$$

$$\frac{\Gamma \vdash t{:}\Sigma x{:}A.B}{\Gamma \vdash \pi_2(t){:}B[x \leftarrow \pi_1(t)]} \qquad \frac{\Gamma \vdash t{:}A \quad A \simeq B \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash t{:}B}$$

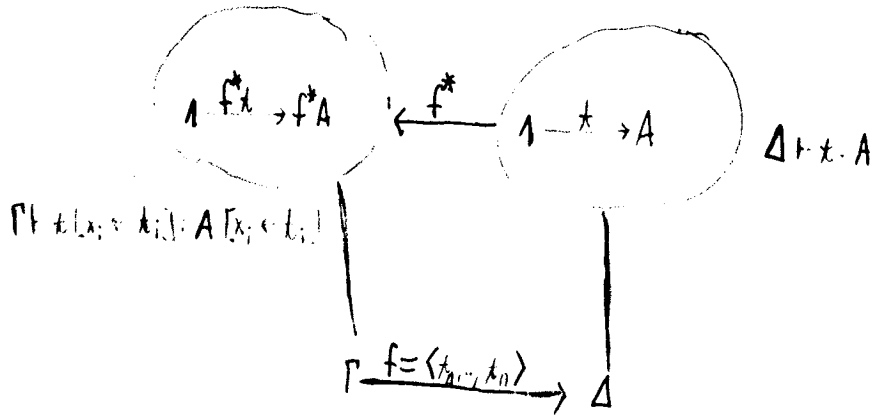
Table 1: Valid Judgements in the Calculus of Constructions

A derivation of categorical combinators

[illegible]

$$
\begin{aligned}
\Gamma &::= [\,] \mid \Gamma \cdot A \\
f &::= \langle\rangle \mid Id \mid f;f \mid Fst \mid \langle f, t \rangle \\
A &::= 1 \mid f * A \mid \Pi(A,A) \mid \Sigma(A,A) \mid \Omega \mid T \\
t &::= 0 \mid f * t \mid Snd \mid Cur(A,t) \mid App \mid pair \mid \pi_0 \mid \pi_2
\end{aligned}
$$

Connection to the categorical structure



Connection to the syntax

Syntax	Combinator
$\Gamma, x:A$	$\rho \cdot A$
$\lambda x:A.t$	$Cur(A,t)$
$\Pi x:A.B$	$\Pi(A,B)$
$(\lambda x:A.t)s$	$Cur(A,t); \langle Id, s\rangle * App$
$x[y \leftarrow t, x_{\cdots}$	$\langle t, s\rangle * Snd = s$
$(\lambda x:A.t)s = t[x \leftarrow s]$	$Cur(A,t); \langle Id, s\rangle * App =$ $\langle Id, s\rangle * t$

2. Reduction rules:

for now Reduction to WHNF. (not inside abstraction)

first step: Orient the equations:

examples:

$\langle f, t\rangle * Snd \rightsquigarrow t$ handling environments

$f * Cur(A,t); \langle Id, s\rangle * App \rightsquigarrow \langle f, s\rangle * t$ (β- reduction)

Important Step: which reduction strategy?

Influencing factors:

1) Evaluating f and t inside $\langle f, t \rangle$?
2) Reducing f always in $f*t$?
3) Do you evaluate first t_1 or t_2 in $t_1; t_2$?

3* Yes ⇒ eager strategy
⇒ Generalization of the CAM

3* No ⇒ lazy strategy
⇒ Generalization of Krivine machine

Theorem 1:

If f canonical, then for all t, t

$$f;t \to^* h$$
$$f*A \to^* B$$
$$f*t \to^* s$$

where h, B, s canonical.

Proof: Transfer reducibility proof of simple combinators (yields only weak normalization)

The Machines

- combinators as ...
- Machine walks through ... spine ... graph

Four registers:

1) **Environment:** stores the current environment
2) **Code** stores the comb. to be reduced, controls execution
3) **Can. Comb.** stores the result of the computation
4) **Stack** (only in the eager case)

Theorem 2:

f canonical and $f;t$ in $nf \to^* B$ and $f*t \to^* s$, then the machines compute these normal forms.

Env.	Code	Can. Comb.	Stack	⇒	Env.	Code	Can. Comb.	Stack
f	g	C	S		h	-	C	S
f	A				i		K	S
f	I				i		s	S

Proof: Induction over the derivation of the reduction relation ... obtained by merging registers Env and Can. Comb ...

Reduction to NF:
Apply this machine successively

May require extra weakening: $f*Cur(t,t') \to Cur(f*A, \ldots)$ if $f;t \to^* t'$

Typechecking

Problem with [illegible] for well-typedness: Rule

$$\frac{\Gamma \vdash t : A \qquad A = B}{\Gamma \vdash t : B}$$

Solution (Harper/Pollack)

Can locate the places where this check occurs:

- $\langle f, t\,[B] \rangle$: Has t a type equal to B?

  Cur (A, t); $\langle Id, s \rangle$ * App:

  Type of s must be convertible to A.

yields a syntax-driven type-inference system

General architecture

Context	Code	Type	Stack		Context	Code	Type	Stack
Γ	t	C	S		A		C	S
	A				$q\Gamma$		B	
	t				$\Delta\Gamma$		A	S

with

$\Gamma \vdash f : \Delta \qquad \Gamma \vdash A = B \qquad \Gamma \vdash t : 1 \to A$

What have we achieved?

- [illegible]
- environments and terms
- combinators yield simple [illegible] abstract machines

Further research

1.) Strong Normalization of the Combinators?
(Reducibility approach yields only weak normalization)

2.) What about linear logic?
Symmetric monoidal closed categories pose a similar problem:

Δ and contexts are identified

Bart Jacobs

Affine and Material Semantics

Abstract

By a semantic analysis it can be shown that the standard embedding of intuitionistic logic into (intuitionistic) linear logic — which makes essential use of Girard's ! — can be factored in two different ways: either via *affine logic* (in which one standardly has weakening but contraction only using an explicit operation $!^+$) or via¿*material logic* in which one has contraction and an operation $\bot$ for weakening). Operationally, one can think of

$>!A$ as A, as often as you like;
$!^+A$ as A, at *least* once;
$\bot A$ as A, at *most* once.

This yields two insights:

- Girard's ! is not a primitive operation, but can be described as composite $\bot!^+$ or as $!^+\bot$.
- Linear functions $!^+A \to B$ use their input at least once: this suggests eager evaluation. Similarly $\bot A \to B$ suggests lazy evaluation, because an element $a \in A$ may be discarded.

All this has a clean categorical semantics using notions from monad theory, introduced some 25 years ago by Anders Kock and Jon Beck.

Full Intuitionistic Linear Logic

Martin Hyland
Dept of Pure Mathematics
University of Cambridge
CB2 1SB
Valeria de Paiva
Computer Laboratory
University of Cambridge
CB2 3QG

Overview

a theory of proofs for a system of full intuitionistic linear logic

- no categories ?
- novel and (hidden) use of categorical logic
- build up to full multiplicative system in steps
- logics and their extensions to term assignment systems
- other FILL talks to come..

Tensor Term System
(based on Abramsky's CILL)

- If X is a finite set of variables, define $\mathcal{P}_X$ the set of patterns with variables in X by:

$$* \in \mathcal{P}_\emptyset \qquad x \otimes y \in \mathcal{P}_{\{x,y\}}$$

- Define $\mathcal{T}_X$ the linear terms with set of free variables X, inductively as follows:
  - $x \in \mathcal{T}_{\{x\}}$;
  - $* \in \mathcal{T}_\emptyset$;
  - $t \in \mathcal{T}_X, u \in \mathcal{T}_Y, X \cap Y = \emptyset$ implies $t \otimes u \in \mathcal{T}_{X \cup Y}$;
  - $w \in \mathcal{T}_X, p \in \mathcal{P}_Y, t \in \mathcal{T}_{Y \cup Z}$, $X \cap Z = \emptyset, Y \cap Z = \emptyset$ implies $(\mathtt{let}\ w = p\ \mathtt{in}\ t) \in \mathcal{T}_{X \cup Z}$

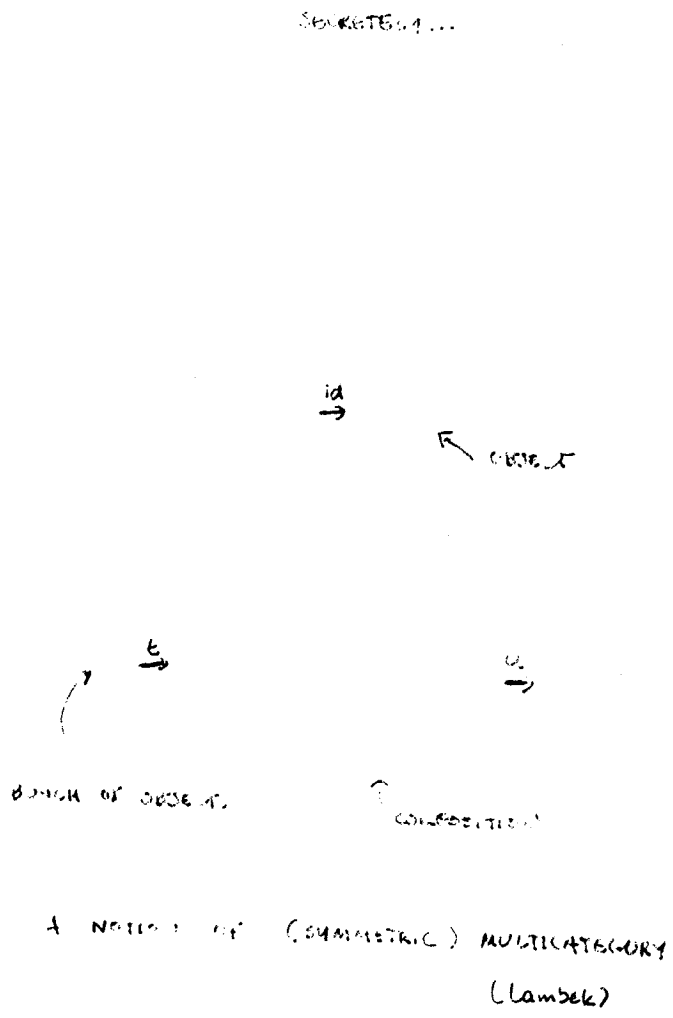
Term assignment for Tensor Logic

Structural Rules:

$$\frac{}{x : A \vdash x : A} \quad (identity)$$

$$\frac{\Gamma, x : A, y : B, \Gamma' \vdash C}{\Gamma, y : B, x : A, \Gamma' \vdash C} \quad (permutation)$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma', x : A, \Delta \vdash u : B}{\Gamma', \Gamma, \Delta \vdash u[t/x] : B} \quad (cut)$$



Logical Rules for Tensor:

$$\frac{}{\vdash * : I} \quad (I_r)$$

$$\frac{\Gamma, \Gamma' \vdash t : A}{\Gamma, z : I, \Gamma' \vdash \mathbf{let}\ z = * \ \mathbf{in}\ t : A} \quad (I_l)$$

$$\frac{\Gamma, x : A, y : B, \Gamma' \vdash t : C}{\Gamma, z : A \otimes B, \Gamma' \vdash \mathbf{let}\ z = x \otimes y \ \mathbf{in}\ t : C} \quad (\otimes_l)$$

$$\frac{\Gamma \vdash u : A \qquad \Gamma' \vdash v : B}{\Gamma, \Gamma' \vdash u \otimes v : A \otimes B} \quad (\otimes_r)$$

OR A 'TENSOR' MULTICATEGORY (LAMBEK'S NOTATION)
(SYMMETRIC)

"ESSENTIALLY" A SYMMETRIC MONOIDAL CAT

$(\mathbf{C}, \mathbf{I}, \bullet)$ + nat. transf

a, b, c

WANT : A TERM CALCULUS
HAVE : A TERM ASSIGNMENT
NEED : REDUCTION RULES

EXAMPLE

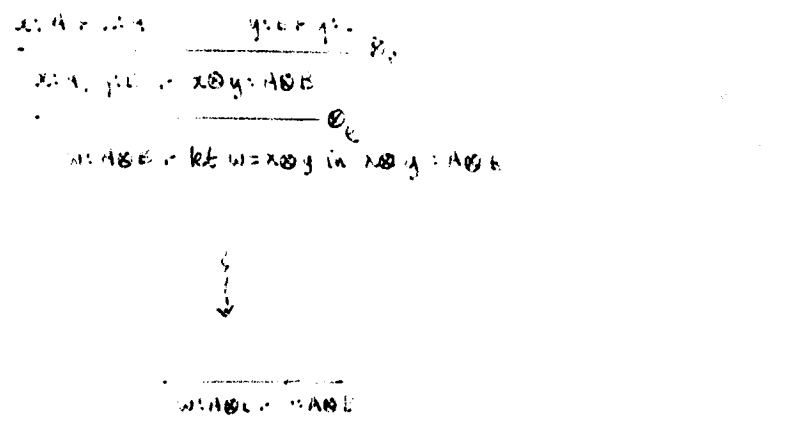
$$\dfrac{\dfrac{\Gamma \vdash u:A \quad \Gamma' \vdash v:B}{\Gamma,\Gamma' \vdash u \otimes v : A \otimes B}\otimes_i \qquad \dfrac{x:A,\, y:B,\, \Delta \vdash t:C}{w:A\otimes B,\, \Delta \vdash \mathrm{let}\ w = x\otimes y\ \mathrm{in}\ t : C}\otimes_e}{\Gamma,\Gamma',\Delta \vdash \mathrm{let}\ u\otimes v = x\otimes y\ \mathrm{in}\ t : C}\ \mathrm{cut}$$

$$\dfrac{\Gamma \vdash u:A \quad \Gamma' \vdash v:B \quad x:A,\, y:B,\, \Delta \vdash t:C}{\Gamma,\Gamma',\Delta \vdash t[u/x,\, v/y] : C}\ \text{“cut}_s\text{”}$$

HENCE β-EQUALITIES :

$$\mathrm{let}\ u\otimes v = x\otimes y\ \mathrm{in}\ t = t[u/x,\, v/y]$$
$$\mathrm{let}\ * = *\ \mathrm{in}\ t = t$$

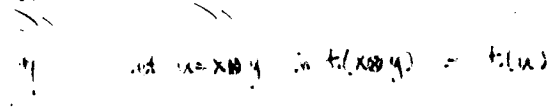
ANOTHER [illegible] :



HENCE

$$\mathrm{let}\ w = x\otimes y\ \mathrm{in}\ x\otimes y = w$$

OR BETTER



$$\mathrm{let}\ u\otimes v\ \mathrm{in}\ t(x\otimes y) = t(u)$$

$$\mathrm{let}\ \lambda\ast\ \mathrm{in}\ t(\ast) = t(\lambda)$$

QUESTION : CAN WE ORIENT EQNS TO OBTAIN A TERM REWRITING SYSTEM AND PROVE USUAL PROPERTIES (CR, TERMINATION, etc)?

WORK WITH MONICA NESI...

Tensor-implication logic

- $t \in \mathcal{T}_X, u \in \mathcal{T}_Y, X \cap Y = \emptyset$ implies $tu \in \mathcal{T}_{X \cup Y}$;

- $t \in \mathcal{T}_{X \cup \{x\}}, x \notin X$ implies $\lambda x.t \in \mathcal{T}_X$.

Rules for linear implication

$$\frac{\Gamma \vdash t : A \quad \Gamma', x : B, \Delta \vdash u : C}{\Gamma, f : A \multimap B, \Gamma', \Delta \vdash u[ft/x] : C} \quad (\multimap_l)$$

$$\frac{\Gamma, x : A, \Gamma' \vdash t : B}{\Gamma, \Gamma' \vdash \lambda x.t : A \multimap B} \quad (\otimes_r)$$

Par Logic

Structural Rules:

$$\frac{}{A \vdash A} \quad (identity)$$

$$\frac{A \vdash \Delta \mid B \mid C \mid \Delta'}{A \vdash \Delta \mid C \mid B \mid \Delta'} \quad (perm)$$

$$\frac{A \vdash \Delta \mid B \mid \Delta' \qquad B \vdash G_i}{A \vdash \Delta \mid G_i \mid \Delta'} \quad (cut)$$

Logical Rules:

$$\frac{}{\bot \vdash} \quad (\bot_l)$$

$$\frac{A \vdash \Delta \mid \Delta'}{A \vdash \Delta \mid \bot \mid \Delta'} \quad (\bot_r)$$

$$\frac{A \vdash D_i \qquad B \vdash E_i}{A \square B \vdash D_i \mid E_i} \quad (\square_l)$$

$$\frac{A \vdash \Delta \mid B \mid C \mid \Delta'}{A \vdash \Delta \mid B \square C \mid \Delta'} \quad (\square_r)$$

Term assignment for Par Logic

If X is a finite set of variables, define $\mathcal{P}_X$ the set of patterns with set of variables X by:

$$\circ \in \mathcal{P}_\emptyset \qquad x\Box - \in \mathcal{P}_{\{x\}} \qquad - \Box y \in \mathcal{P}_{\{y\}}$$

Define $\mathcal{T}_X$ the linear terms with set of free variables X, inductively as follows:

- $x \in \mathcal{T}_{\{x\}}$; $\circ \in \mathcal{T}_\emptyset$;
- $t \in \mathcal{T}_X, u \in \mathcal{T}_Y, X \cap Y = \emptyset$ implies $t \Box u \in \mathcal{T}_{X \cup Y}$;
- $t \in \mathcal{T}_X, p \in \mathcal{P}_Y, u \in \mathcal{T}_{Y \cup Z}, X \cap Z = \emptyset$, $Y \cap Z = \emptyset$ implies $(\mathbf{match}\ t = p\ \mathbf{in}\ u) \in \mathcal{T}_{X \cup Z}$

Term system for Par logic

Structural Rules:

$$\frac{}{x : A \vdash x : A}\ (identity)$$

$$\frac{x : A \vdash \Delta \mid y : B \mid z : C \mid \Delta'}{x : A \vdash \Delta \mid z : C \mid y : B \mid \Delta'}\ (perm)$$

$$\frac{x : A \vdash \Delta \mid t : B \mid \Delta' \qquad y : B \vdash g_i : G_i}{x : A \vdash \Delta \mid g_i[t/y] \mid \Delta'}\ (cut)$$

Logical Rules for Par system:

$$\frac{}{x : \bot \vdash} \ (\bot_l)$$

$$\frac{x : A \vdash \Delta \mid \Delta'}{x : A \vdash \Delta \mid \circ : \bot \mid \Delta'} \ (\bot_r)$$

$$\frac{x : A \vdash \Delta \mid u : B \mid v : C \mid \Delta'}{x : A \vdash \Delta \mid u \square v : B \square C \mid \Delta'} \ (\square_r)$$

$$\frac{x : A \vdash d_i : D_i \qquad y : B \vdash e_i : E_i}{z\, A \square B \vdash \mathbf{m}z = x \square - \mathbf{in}\ d_i : D_i \mid \mathbf{m}z = - \square y \mathbf{in}\ e_i : E_i}$$

Tensor-par logic

Structural Rules:

$$\frac{}{x : A \vdash x : A} \ (identity)$$

$$\frac{\Gamma, x : A, y : B, \Gamma' \vdash \Delta \mid z : C \mid w : D \mid \Delta'}{\Gamma, y : B, x : A, \Gamma' \vdash \Delta \mid w : D \mid z : C \mid \Delta'} \ (perm)$$

$$\frac{\Gamma_1 \vdash \Delta \mid t : B \mid \Delta' \qquad \Gamma_2, y : B, \Gamma_3 \vdash g_i : G_i}{\Gamma_2, \Gamma_1, \Gamma_3 \vdash \Delta \mid g_i[t/y] \mid \Delta'} \ (cu$$

ASSIGNMENT

Tensor-Par Logic:

Tensor rules

$$\frac{\Gamma, \Gamma' \vdash t_i : A}{\Gamma, z : I, \Gamma' \vdash \mathbf{let}\ z = * \ \mathbf{in}\ t_i : A_i}\ (I_l)$$

$$\frac{}{\vdash * : I}\ (I_r)$$

$$\frac{\Gamma, x : A, y : B, \Gamma' \vdash t_i : C_i}{\Gamma, z : A \otimes B, \Gamma' \vdash \mathbf{let}\ z = x \otimes y\ \mathbf{in}\ t_i : C_i}\ (\otimes_l)$$

$$\frac{\Gamma \vdash u : A \mid \Delta \qquad \Gamma' \vdash v : B \mid \Delta'}{\Gamma, \Gamma' \vdash u \otimes v : A \otimes B \mid \Delta \mid \Delta'}\ (\otimes_r)$$

Par rules

$$\frac{}{x : \bot \vdash}\ (\bot_l)$$

$$\frac{\Gamma \vdash \Delta \mid \Delta}{\Gamma \vdash \Delta \mid \circ : \bot \mid \Delta'}\ (\bot_r)$$

$$\frac{\Gamma \vdash \Delta \mid u : B \mid v : C \mid \Delta'}{\Gamma \vdash \Delta \mid u \square v : B \square C \mid \Delta'}\ (\square_r)$$

$$\frac{\Gamma, x : A \vdash d_i : D_i \qquad \Gamma', y : B \vdash e_i : E_i}{\Gamma\Gamma' z . A \square B \vdash \mathtt{m} z = x \square - \ \mathbf{in}\ d_i : D_i \mid \mathtt{m} z = - \square y \mathbf{in} e_i : E_i}$$

(full) Multiplicative fragment

Logical Rules for linear implication:

$$(\multimap_l)\frac{\Gamma \vdash \Gamma' \mid t : A \quad x : B, \Delta \vdash u_i : C_i}{\Gamma, f : A \multimap B, \Delta \vdash \Gamma' \mid u_i[ft/x] : C}$$

$$(\multimap_r)\frac{\Gamma, x : A, \Gamma' \vdash t : B \mid e : \Delta}{\Gamma, \Gamma' \vdash \lambda x.t : A \multimap B \mid e : \Delta} \text{ if } x \notin \Delta$$

THEOREM:

[illegible] FOR THE

[illegible] FRAGMENT OF ALL.

FURTHER WORK

(LOTS TO DO ...)

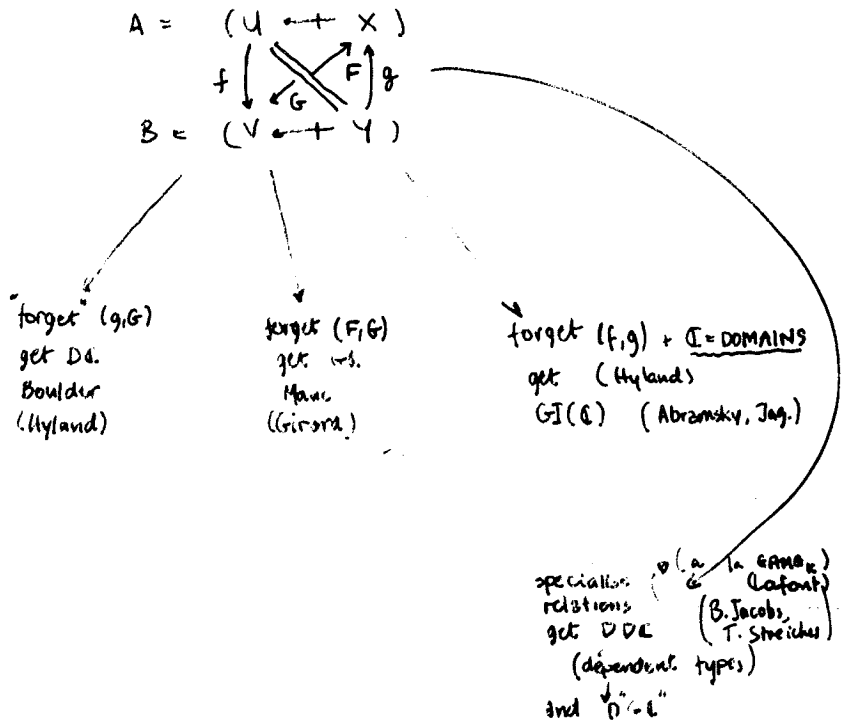
- MODALITIES
- ADDITIVES
- TERM CALCULI PROPERTIES (with M. Nesi)
- PROCESS NETS INT. (with Nick Benton & Gavin Bierman)
- GENERALISATIONS

(A. Joyal)

Thanks

A JOKE

$\mathbb{L}$ WITH LOTS OF PROPERTIES



On Compactness and $\underline{\underline{Cat}}$-enriched categories

Marcello Fiore and Gordon Plotkin

What is the point (?) of compactness?

Look at What one is trying to do.

(1) Can we solve more problems?
(2) Can we solve present problems more efficiently, perspicuously?
Should one change the problem set?

Aarhus
March 1992.

To 1) main hope = to solve problem of axiomatizing domain theory. Comes from reference that all categories where one solves domain equations are more or less ideal (wherein or category, eg in Cat)

2) at gross level: coping with contravariance: reduction to covariance, avoiding embeddings/projections, limits (except to get started) all main idea: think of $T : \mathbb{K}^{op} \times \mathbb{K} \to \mathbb{K}$ rather than $T : \mathbb{K}^{E} \to \mathbb{K}^{E}$. Seems to me more important to think of such T's than to avoid embeddings etc as such; they are a perfectly good tool.

Conjectures: Every functor has an initial algebra and a final coalgebra and they are canonically "isomorphic" (as to speak)

The dramatic "every" interesting per se, but can be reformulated by restricting the quantifier (every "nice" functor ...).

Equal emphasis on the initiality and finality and that they are the same, as for computation (as in Set). We need in programming languages to choose which to use for computation (cf. Hagino 'adjoints'); computational category is theory - programming language design inspired by category theory, where one aims to provide direct implementation of categorical concepts)

Problem Set: Solve recursive domain equations [or other things, eg type operators]

$y \cong F(x, y)$ parameter (multiple instances of problem)

$\cong$ $y \cong G(x, x, y, y)$

contra, co; $-$, $+$

Special Case: F co-variant $F : \mathbb{K} \times \mathbb{L} \to \mathbb{L}$

General Case: $G : \check{\mathbb{K}} \times \mathbb{L} \to \mathbb{L}$ $\quad (\check{\mathbb{K}} = \mathbb{K}^{op} \times \mathbb{K})$

Why: (a) to get domains for programming language semantics
(b) to get datatypes within programming languages $\mathbb{N} \cong [\![\theta]\!]\mathbb{N}$

initiality:	primitive recursion	induction	fairly well worked out for covariant, less well for contravariant
finality:	observation	coinduction	
resolution:	all ability to both	good programming tool	

Search + Kleene's μ ~ recursion

Kleene: $h(x) \simeq \mu n.\ P(x, n)$ $\qquad P : \mathbb{N} \to \mathbb{T}$

Search:

"Primitive corecursion":

$$\begin{array}{ccc} \mathbb{1} + X & \xleftarrow{\beta} & X \\ \downarrow \mathbb{1}+h & & \downarrow h \\ \mathbb{1} + \mathbb{N} & \longleftarrow & \mathbb{N} \end{array} \qquad t : X \to \mathbb{1}$$

define $P : X \to \mathbb{T}$ $\quad P = (\mathbb{1} + t) \circ \beta$

$f : X \to X$ $\quad f = (? + id) \circ \beta$

$h(x) \simeq \mu n.\ P(f^n(x))$

Kleene $\Rightarrow$ Search + P.r. : clear for X a product of $\mathbb{N}$'s.

Search $\Rightarrow$ Kleene : Take $X = \mathbb{N} \times \mathbb{N}$

$$\beta(x, n) \simeq \begin{cases} inl(*) & (\text{if } P(n)) \\ inr(x, n+1) & (\text{if } \neg P(n)) \end{cases}$$

Then $h'(x, m) \simeq \mu n \geq m.\ P(x, n)$

$\therefore$ $h(x) \simeq h'(x, 0)$

So if have distributive algebra can has strong natural numbers object, "also" a final co-algebra: can represent all partial functions: [Gödel's theorem might] $\Rightarrow$ by hand: $h \triangleq \theta$

in $\theta(m) \simeq n$ / $h \circ \underline{n} = n$

$\theta(m) \downarrow$ if $h \circ \underline{m}$ total

? Converse false with ord. ? but from ... : Logic ?

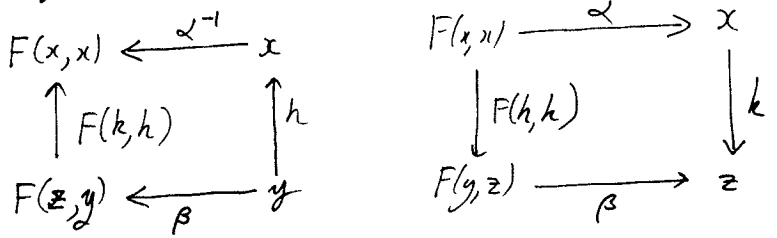
The covariant case

Need more than $\underline{\underline{\mathcal{K}}}^{E}$ to get initiality (or finality for that matter)
want to deal with $T : \underline{\underline{\mathcal{K}}} \to \underline{\underline{\mathcal{K}}}$ and above is argument for compactness
But need something to get completeness / compactness; ~~Kinitiality~~ known in
a CPO-enriched setting; ~~$\mathcal{K}$~~-finality not known in general.

The contravariant case

Need $x \cong F(x,x) : \alpha$, regarding $F : \underline{\underline{\mathcal{K}}}^{\vee} \to \underline{\underline{\mathcal{K}}}$ (could also
think of $\underline{\underline{\mathcal{K}}}^{\Delta} \to \underline{\underline{\mathcal{K}}}$ ~ well known to me, but less general and natural;
not sure if $\underline{\underline{\mathcal{K}}}^{\vee}$ known to me before last year, maybe at $\mathcal{K}$.; $\underline{\underline{\mathcal{K}}}^{\Delta}$ not pursued
more here.)

Then natural request is that for any y, z, $\gamma : F(y,z) \to z$, $\beta : F(y,z) \leftarrow y$ there are unique $h : x \leftarrow y$, $k : x \to z$ st.
the following two diagrams commute:



This implies compactness (for suitable F) ~~taken over~~ $\underline{\underline{\mathcal{K}}}$, taking them
is bifunctor, dummy in the first argument: $G : \underline{\underline{\mathcal{K}}}^{\vee} \to \underline{\underline{\mathcal{K}}}^{\vee}$
Also implies compactness for associated $G : \underline{\underline{\mathcal{K}}}^{\vee} \to \underline{\underline{\mathcal{K}}}^{\vee}$ ~~(induced in evident~~
~~way from $F : \underline{\underline{\mathcal{K}}}^{\vee} \to \underline{\underline{\mathcal{K}}}$)~~ Conversely if have compactness for G can find such x, α : so can
always get on to the diagonal.

Proof Follows from more general stuff below ☒

So compactness comes up as a necessary ~~(for $\underline{\underline{\mathcal{K}}}$)~~ and
sufficient ~~(and also necessary)~~.

Organizationally can help. eg. (Freyd) $\underline{\underline{\mathcal{K}}}$ compact $\Rightarrow$ $\underline{\underline{\mathcal{K}}}^{\mathcal{K}^{op} \times \mathcal{K}}$ compact (although
or (rather than go into diagonal in brute force way) show $\underline{\underline{\mathcal{K}}}^{\mathcal{K}^{op} \times \mathcal{K}}$
compact & then get on to diagonal. Also useful as a language.

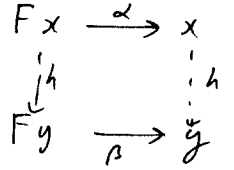
V-Completeness Every $\underline{\underline{V}}$-functor $F : \underline{\underline{\mathcal{K}}} \to \underline{\underline{\mathcal{K}}}$ on a V-category
has an initial algebra.

Examples $\underline{\underline{V}} = \underline{\underline{Pos}}$ algebras as usual with induced order on
hom-sets.

$\underline{\underline{V}} = \underline{\underline{CPO}}$ as for $\underline{\underline{Pos}}$

Question Do it 2-categorically.

Strong V-Completeness Consider (e.g.) in $\underline{\underline{Pos}}$



? Can the passage $\beta \mapsto h$ be $\underline{\underline{V}}$-given, (is it monotone)
i.e. is there is a
suitable: $\underline{\underline{V}}(Fy, y) \to \underline{\underline{V}}(x, y)$ ~~$\underline{\underline{V}}(x,y)$~~

[Note if monotone then (in CPO-enriched case) also continuous]

V-Compactness [$\underline{\underline{V}}$-Lambek holds]

Fairly Sure: $\underline{\underline{\mathcal{K}}}, \underline{\underline{\mathcal{L}}}$ strongly $\underline{\underline{V}}$-Complete $\Rightarrow$ so is $\underline{\underline{\mathcal{K}}}^{\mathcal{L}^{op} \times \mathcal{L}}$
(similarly Compactness)

Problem is re parameterization
— will return to this point later.

(6)

Basic Construction : Enough colimits give initial algebras

initial $F : \mathbb{K} \to \mathbb{K}$

$\Delta = \bot \xrightarrow{i_{F\bot}} F\bot \xrightarrow{F(i_{F\bot})} F^2\bot \longrightarrow$

Suppose $\rho : \Delta \to Fix_F$ exists colimiting

and $F\rho : F\Delta \to F(Fix_F)$ colimiting.

$\downarrow z_F$

then $\rho^- : \Delta^- \to Fix_F$ colimiting

$z_F : F(Fix_F) \to Fix_F$ is initial algebra

(Continuity is CT. recursion ; other things fix too)

Note co- basic lemma.

Want : $\bot$ initial
colimits of such chains.

Pos - enriched categories

Adjoint $f \dashv g$ iff $g \circ f \geq id$ & $f \circ g \leq id$

Embedding-Projection Pair $f \vartriangleleft g$ iff $g \circ f = id$ & $f \circ g \leq id$

$f \dashv g$, $f' \dashv g'$ $f \leq f'$ iff $g \leq g'$

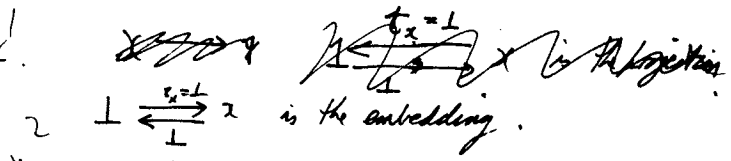
& $f \xhookrightarrow{1-1} g$

Theorem T.f.a.e. 1. $\bot$ exists, $\bot \circ \bot = \bot$, there is an initial object i

2. there is an e-initial object.

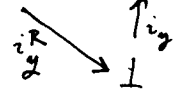
[AND ITS DUAL] And they imply 3. composition is left-strict.

Proof $1 \Rightarrow 2$.

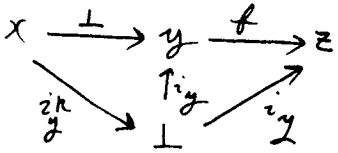


$\bot \underset{\bot}{\overset{r_x = \bot}{\rightleftarrows}} x$ is the embedding.

$2 \Rightarrow 1$



$2 \Rightarrow 3$.



Remark : get

Δ an embedding-projection chain.

Theorem T.f.a.e 1. there is an e-1 zero object.

2. there is a 0 object $\bot$, and & $\bot \circ \bot = \bot$

3. there is an initial object and composition is bistrict

3D. ——— final " " " " " .

(8)

CPO-enriched categories

Completeness ⇔ Compactness

Theorem T.f.a.e for a chain Δ of embeddings and a $\rho: \Delta \to x$.
1. ρ is colimiting in $\underline{\mathbb{K}}$
2. ρ is an embedding chain & $\bigvee \rho_m \circ \rho_m^R = id$.

and finally
3. ρ is colimiting in $\underline{\mathbb{K}}^E$.

Proof 2 ⇒ 1. For $\psi: \Delta \to y$ the mediating map $x \to y$ is $\bigvee \psi_m \circ \rho_m^R$ ☒

Note ① duals: $\underline{\mathbb{K}}^{op}$ have order alone.

left adjoint	⟷ right adjoint
right adjoint	⟷ left adjoint
embedding	⟷ projection
projection	⟷ embedding.

② As condition 2 is essentially self-dual compact together in one long theorem.

③ Recall Adjoint pair situation.

Corollary Any CPO-enriched category with e-initial objects and colimits of embedding chains is algebraically complete in the CPO-enriched sense

Remarks Any initial algebra of a CPO-functor is strongly initial (assuming the hom-sets have least elements) by a least fixed-point argument. So Similarly for final co-algebras. So strong and weak senses of completeness and compactness coincide.

(ac.) ~~CPO, the final co-algebras~~
ACPO

Proposition Let $\underline{\mathbb{K}} \supseteq \underline{\mathbb{T}}$. Suppose $\underline{\mathbb{K}}$ has an e-initial object ~~0~~ and $Tx \xrightarrow{\alpha} x$ is initial. Then $x \cong \varinjlim \Delta$.

Proof The canonical cone $\rho: \Delta \to x$ is defined by $\rho_0 = \bot$, $\rho_{n+1} = \alpha \circ T(\rho_n)$. Show $\bigvee \rho_n \circ \rho_n^R = id$ as LHS is a homomorphism from (x,α) to (x,α) ☒. For this take $\pi_n: x \to x$ where $\pi_0 = \bot$, $\pi_{n+1} = \alpha \circ T(\pi_n) \circ \alpha^{-1}$ and get $\pi_n = \rho_n \circ \rho_n^R$ ☒

Problem Find a $\underline{\mathbb{K}}$ with an e-initial object which is algebraically complete, but does not have colimits of all embedding chains.

Theorem [Freyd] Let $\underline{\mathbb{K}}$ have an e-p zero object. Then if $\underline{\mathbb{K}}$ is algebraically complete it is algebraically compact.

Proof Assume a particular F. ~~By the proposition we can assume that there is an initial Algebra $\alpha: T(x \times y) \to x \times y$ constructed as for the Basic Lemma~~ Let $\alpha: T(x) \to x$ be initial. & To show (x, α^{-1}) final let (y, β) be another co-algebra. Define ~~f_n~~ $f_n: y \to x$ by $f_0 = \bot$, $f_{n+1} = \alpha^{-1} \circ Tf_n \circ \beta$, $f = \bigvee f_n$.

Existence $f = \alpha^{-1} \circ Tf \circ \beta$.

Uniqueness Let f' be another homomorphism. Then $\pi_n \circ f' = f_n$ ☒

Corollary Let $\underline{\mathbb{K}}$ have an e-p zero object and colimits of embedding chains. Then it is algebraically compact.

Exs $\underline{\mathbb{P}}$CPPO and many related categories.

CPPO - for algebraically co-complete.

(10)

Parameterisation (Nm - enriched)

Data $F : \underline{\underline{K}} \times \underline{\underline{L}} \longrightarrow \underline{\underline{L}}$

Suppose for every x in $\mathrm{Ob}(\underline{\underline{K}})$ have $z_x : F_x(G_x) \to G_x$ initial.

For $x \xrightarrow{f} x'$ define $G_x \xrightarrow{Gf} G_{x'}$ to be unique st.

$$\begin{array}{ccc} F_x(G_x) & \xrightarrow{z_x} & G_x \\ {\scriptstyle F_x(Gf)}\downarrow & & \downarrow{\scriptstyle Gf} \\ F_x(G_{x'}) & \xrightarrow{F(f,G_{x'})} & F_{x'}(G_{x'}) \xrightarrow{z_{x'}} G_{x'} \end{array}$$

Then $G : \underline{\underline{L}} \longrightarrow \underline{\underline{L}}$ is a functor, which we will call $F^\dagger$

$$y \cong F(x,y) \quad : \quad y = G_x F^\dagger(x).$$

Remark (Enriched) Suppose we are now in the CPO-enriched context of $\perp$'s exist. In a previous rek remark strong initiality was noted, so here the passage $f \longmapsto Gf$ is continuous as it is $f \longrightarrow F(f, G_{x'}) \circ z_{x'} \longrightarrow$ least upper bound. Hence if $\perp$'s exist, get ability to do parameterisation.

Conjecture $\underline{\underline{V}}$-Parameterisation works if have strong $\underline{\underline{V}}$-initial algebras. Pressure can then go on as for Freyd to get pressurably conjectured $\underline{\underline{V}}$-theorems.

get

$$\begin{array}{ccc} \underline{\underline{K}} & \xrightarrow{F^\dagger} & \underline{\underline{L}} \\ {\scriptstyle \langle \mathrm{Id}, F^\dagger \rangle}\downarrow & \nearrow_{z} & \nearrow_{F} \\ \underline{\underline{K}} \times \underline{\underline{L}} & & \end{array}$$

and given any some universal condition I have not worked out for initiality / completeness.

? Good way to treat at the 2-level

Contravariant Functors (non-enriched)

$\check{\underline{\underline{K}}} =_{\mathrm{df}} \underline{\underline{K}}^{\mathrm{op}} \times \underline{\underline{K}}$, $(\check{\underline{\underline{K}}})^{\mathrm{op}} = \underline{\underline{K}} \times \underline{\underline{K}}^{\mathrm{op}} =_{\mathrm{df}} \hat{\underline{\underline{K}}}$

Note unitary involution $D : \check{\underline{\underline{K}}} \cong \hat{\underline{\underline{K}}}$, so $\mathrm{Id}_{\check{\underline{\underline{K}}}} = D^{\mathrm{op}} \circ D$.

Remark Can consider iterated version when $\underline{\underline{K}}^\Delta$ arrows $x \rightrightarrows y$ where now have $\Delta : \underline{\underline{K}}^\Delta \cong (\underline{\underline{K}}^\Delta)^{\mathrm{op}} = \underline{\underline{K}}^\Delta$ $\quad \Delta(f,g) = (g,f)$.
Here $\Delta = \Delta^{\mathrm{op}} \circ \Delta$; Δ fixes the objects.

Symmetric Functors

$$\begin{array}{ccc} \check{\underline{\underline{K}}} & \xrightarrow{F} & \check{\underline{\underline{L}}} \\ {\scriptstyle D_{\underline{\underline{K}}}}\downarrow & & \downarrow{\scriptstyle D_{\underline{\underline{L}}}} \\ \hat{\underline{\underline{K}}} & \xrightarrow{F^{\mathrm{op}}} & \hat{\underline{\underline{L}}} \end{array}$$

1-1 correspondence

$$\begin{array}{ccc} \check{\underline{\underline{K}}} & \xrightarrow{G} & \underline{\underline{L}} \\ \| & & \| \\ \text{Symmetric } \check{\underline{\underline{K}}} & \xrightarrow{F} & \check{\underline{\underline{K}}} \end{array}$$

Proof $\Rightarrow$ take $F = \langle G^{\mathrm{op}} \circ D, G \rangle$ or, e.g. $F(x,y) = \langle G(y,x), G(x,y) \rangle$.
Calculate $D \circ F = \langle G, G^{\mathrm{op}} \circ D \rangle$
$F^{\mathrm{op}} \circ D = \langle G \circ D^{\mathrm{op}}, G^{\mathrm{op}} \rangle \circ D = D \circ F$.

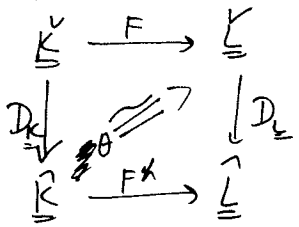
$\Leftarrow$ set $F = \langle H, G \rangle$. Take the second component:
Notice/bet $D \circ F = \langle G, H \rangle$ and $F^{\mathrm{op}} \circ D = \langle H^{\mathrm{op}} \circ D, G^{\mathrm{op}} \circ D \rangle$
so $H = G^{\mathrm{op}} \circ D$ getting 1-1 correspondence, other fact being trivial.

Enrichment Goes through with out difficulty. As having e-h zeros & colimits of embedding chains is preserved by $(\ )^{\mathrm{op}}$ & $-\times-$ - get $\underline{\underline{K}}^{\mathrm{op}} \times \underline{\underline{K}}$ compact if $\underline{\underline{K}}$ satisfies these conditions (Alternative: $\underline{\underline{V}}$-Freyd). Still need to get on the diagonal.

(12)

Parameterisation and weakly symmetric functors.

Weak Symmetry



and suppose $\underline{\underline{\check{L}}}$ is complete.

Parameterisation $F : \underline{\underline{\check{K}}} \times \underline{\underline{\check{L}}} \longrightarrow \underline{\underline{\check{L}}}$ / Consider.

$F^\dagger : \underline{\underline{\check{K}}} \longrightarrow \underline{\underline{\check{L}}}$ obtained from $\gamma_x : F_x(F^\dagger(x)) \cong (F^\dagger)_x$

Claim $F^\dagger$ is weakly symmetric.

Proof (1) As $\gamma_x : F_x(F^\dagger x) \longrightarrow F^\dagger x$ is initial,

$D\gamma_x^* : DF^\dagger x \longrightarrow (DF)_x(F^\dagger x) = F_{Dx}(DF^\dagger x)$ (symmetry of F)

is final.

(2) By completeness, as $\gamma_{Dx} : F_{Dx}(F^\dagger(Dx)) \longrightarrow F^\dagger(Dx)$ is initial

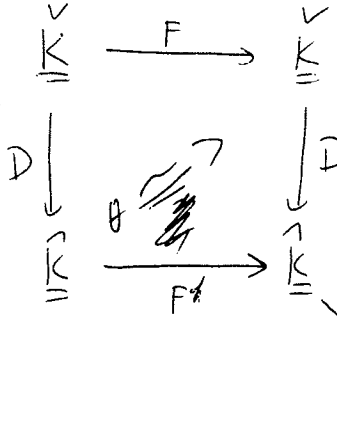
$\gamma_{Dx}^{-1} : (F^\dagger)^{\theta_x}(Dx) \longrightarrow F_{(Dx)}((F^\dagger)^{\theta_x} Dx)$ is final.

and have fill-in unique homomorphism:

$\theta_x : (F^\dagger)^{\theta_x} D_x \cong DF^\dagger(x)$

which is (hopefully) natural ☒

Any weakly symmetric functor is naturally equivalent to a symmetric one



Take F of the form:

$F = \langle G^{\theta_x}, H \rangle$

$H : \underline{\underline{\check{K}}} \to \underline{\underline{K}}$

$G : \underline{\underline{K}} \to \underline{\underline{K}}$

$F^\theta = \langle G, H^\theta \rangle$

$\pi_2 \circ D \circ F = \pi_2 \circ \langle H, G^{\theta_x} \rangle = G^{\theta_x}$

$\cong \pi_2 \circ F^\theta \circ D = H^\theta \circ D$

So $F \cong \langle H^\theta \circ D, H \rangle$ which is symmetric.

So given $F^\sharp : \underline{\underline{\check{K}}} \times \underline{\underline{\check{L}}} \longrightarrow \underline{\underline{L}}$ symmetric, can get $F^\dagger : \underline{\underline{\check{K}}} \longrightarrow \underline{\underline{L}}$ symmetric (& still satisfying the adjunction diagrams)

N.B. In trivial case $\underline{\underline{K}} = \mathbb{1}$ find general solution of the diagram gives required solution on the diagonal.

What if want to get $\gamma_x : F_x(F^\dagger x) \cong F^\dagger x$ iso.

$D\gamma_x = \gamma_x^{-1}$ (i.e. $D\gamma = \gamma^{-1}$)

Remark ($\mathcal{V}$-enriched version goes thro' just need existence of $\mathcal{V}$'s to allow the parameterisation.

(14)

Higher-Order Domain Equations

Covariant Case

$\underline{\underline{CPO}}$-Cat is a ccc by $\underline{\underline{V}}$-nonsense. [Can also work with $\underline{\underline{CPO}}_{\perp}$]

Functor spaces are $\underline{\underline{CPO}}$-functors and natural transformations (ordered pointwise).

It may be that the $\underline{\underline{CPO}}$-algebraically compact objects (with e-p zero) are not closed under functor spaces. However the objects with e-p zeros and colimits of embedding chains are (recall limits of functors are calculated pointwise)

Y-Combinator: $eval_{\underline{\underline{K}},\underline{\underline{K}}} : (\underline{\underline{K}} \Rightarrow \underline{\underline{K}}) \Rightarrow \underline{\underline{K}}$.

General Case $\check{\underline{\underline{K}}}$ is actually a ~~functor~~ construction on $\underline{\underline{CPO}}$-Cat.

$$\Sigma : \check{\underline{\underline{K}}} \longrightarrow \underline{\underline{K}} \quad \text{is} \quad \Pi_2$$

lifting is
$$\frac{\check{\underline{\underline{K}}} \xrightarrow{G} \underline{\underline{L}}}{\underline{\underline{K}} \times \underline{\underline{L}}^{\vee} \xrightarrow{\langle G^{\sharp} \circ D,\, G \rangle} \underline{\underline{L}}^{\vee}}$$

as previously considered.

Trivially, $(\,)^{\vee}$ preserves products so (by general nonsense) $\underline{\underline{CPO}}\text{-}Cat_{V}^{*}$ is also ccc, inheriting products and with

$$\underline{\underline{K}} \Rightarrow_{V} \underline{\underline{L}} = (\underline{\underline{K}}^{\vee} \Rightarrow \underline{\underline{L}})$$

and will have

$$\mathbb{Y} : (\underline{\underline{K}} \Rightarrow_{V} \underline{\underline{K}}) \Rightarrow_{V} \underline{\underline{K}}$$

available.

Note Applications to Fw + recursion.

FUNCTORIAL PARAMETRICITY

PETER FREYD

EDMUND ROBINSON

GIUSEPPE ROSOLINI

A report of the same title can be obtained from the Computer Science Dept., School of Cognitive & Computing Sciences, University of Sussex, Brighton BN1 9QH. (Report No. 12/91).

PARAMETRICITY

I STRACHEY SOMETHING IS "PARAMETRIC"
IF IT IS GIVEN BY A "UNIFORM ALGORITHM"

- SUGGESTS THINGS ARE BASICALLY UNTYPED
- (FOR POLYMORPHISM) LEADS TO SOMETHING SLIGHTLY MORE GENERAL THAN A PER MODEL.

II FOURMAN - PHOA :
PARAMETRISED THINGS LIVE IN FREE CATEGORY.
EXAMPLE : PARAMETRISED TYPES LIVE IN SLICE CATEGORY.

III REYNOLDS :
PARAMETRISED TYPES ARE FUNCTIONS FROM SPACE OF PARAMETERS TO TYPES - SATISFYING SOME PARAMETRICITY CONDITION.
PARAMETRICITY NEEDED TO PRESERVE ABSTRACTNESS OF IMPLEMENTATION.

$\Gamma \vdash F\ \mathrm{Type}$

IS INTERPRETED BY A FUNCTION

$$[\![\Gamma]\!] \longrightarrow [\![\mathrm{Type}]\!]$$

EG $X : \mathrm{Type} \vdash F(X) : \mathrm{Type}$

IS INTERPRETED BY A FUNCTION

$$[\![F]\!] : [\![\mathrm{Type}]\!] \longrightarrow [\![\mathrm{Type}]\!]$$

(SATISFYING PARAMETRICITY CONDITIONS)

IF YOU ARE A CATEGORY THEORIST, YOU IMMEDIATELY ASSUME THIS MEANS $[\![F]\!]$ IS A FUNCTOR (AND VALUES OF TYPE F ARE GIVEN BY NATURAL TRANSFORMATIONS)

USE $\mathcal{M}$ FOR CATEGORY OF TYPES

PROBLEM 1 : GIVEN $\mathbb{M}$ CARTESIAN CLOSED FUNCTOR CATEGORY $\mathbb{M}^{\mathbb{D}}$ MAY NOT BE.

BUT IF $\mathbb{M}$ IS SUFFICIENTLY COMPLETE, IT WILL BE.

MOTIVATING EXAMPLE : $\mathbb{M} = \mathrm{SET}$

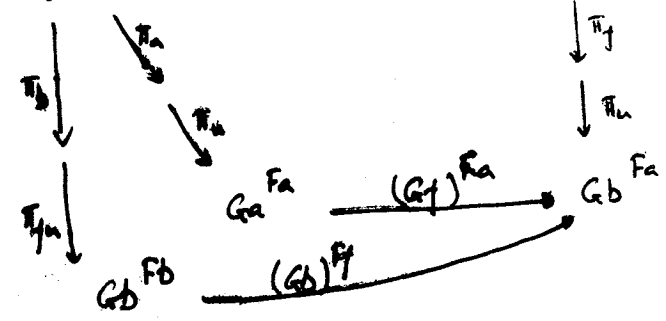
$F, G \in \mathrm{SET}^{\mathbb{D}}$

YONEDA $\Rightarrow$ $G^F(d) \cong \mathrm{SET}^{\mathbb{D}}(\mathbb{D}(d,-), G^F)$

$\cong \mathrm{SET}^{\mathbb{D}}(\mathbb{D}(d,-) \times F, G)$

$$\prod_{x \in \mathbb{D}_0} \mathrm{SET}\left(\mathbb{D}(d,x) \times Fx, Gx\right) \rightrightarrows \prod_{a \xrightarrow{f} b \in \mathbb{D}_1} \mathrm{SET}\left(\mathbb{D}(d,a) \times Fa, Gb\right)$$

$$\prod_{x \in \mathbb{D}_0} \prod_{y \in \mathbb{D}(d,x)} Gx^{Fx} \rightrightarrows \prod_{a \xrightarrow{f} b \in \mathbb{D}_1} \prod_{u \in \mathbb{D}(d,a)} Gb^{Fa}$$



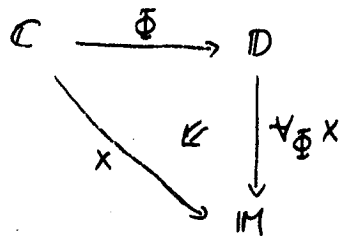
HOW COMPLETE SHOULD $\mathbb{M}$ BE ?

GIVEN $\Phi : \mathbb{C} \longrightarrow \mathbb{D}$

HAVE $\Phi^* : \mathbb{M}^{\mathbb{D}} \longrightarrow \mathbb{M}^{\mathbb{C}}$

QUANTIFICATION ($\forall$) IS A RIGHT ADJOINT FOR THIS.

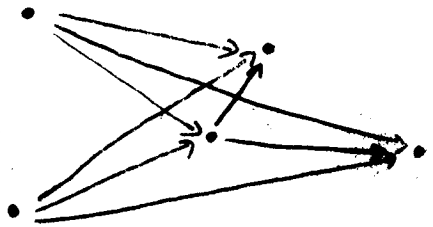
I.E. WE NEED RIGHT KAN EXTENSIONS.



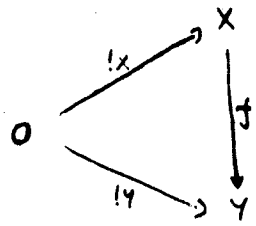
KAN EXTENSIONS ARE LIMITS.

WE WANT $\mathbb{M}$ TO HAVE ALL RIGHT KAN EXTENSIONS + BECK CONDITION.

$\mathbb{M}$ COMPLETE $\Leftrightarrow$ $\mathbb{M}$ COCOMPLETE.



EXAMPLE:
INITIAL OBJECT = $\varprojlim \mathrm{Id}_{\mathbb{M}}$



IN PTIC $!X \cdot !0 = !X$
SO $!0$ IS A MAP OF CONES
$!0 = \mathrm{Id}_0$
BUT IF $f: 0 \to X$ $f = f \circ \mathrm{Id}_0 = f \circ !0 = !X$.

PROBLEM 2.

IF F AND G ARE PARAMETRIZED TYPES

THEN $[\![F \to G]\!]$ IN $\mathbb{M}^{\mathbb{D}}$ IS $[\![G]\!]^{[\![F]\!]}$

DERIVED RULE OF SUBSTITUTION IMPLIES

$$[\![(F \to G)(a)]\!] = [\![G(a)]\!]^{[\![F(a)]\!]}$$

IF FUNCTION SPACES ARE CALCULATED POINTWISE!

6.

THEOREM: IF $\mathcal{M}$ IS A COMPLETE INTERNAL CARTESIAN CLOSED CATEGORY IN A LOCALLY CARTESIAN CLOSED CATEGORY WITH COPRODUCTS $\mathcal{E}$, $\mathbb{D}$ IS AN $\mathcal{M}$-ENRICHED INTERNAL CATEGORY IN $\mathcal{E}$, THEN FUNCTOR SPACES IN $\mathcal{M}^{\mathbb{D}}$ ARE CALCULATED POINTWISE IFF $\mathbb{D}$ IS A GROUPOID.

PROOF: WANT $\forall F, G : \mathbb{D} \to \mathcal{M}$ $\forall A : \mathbb{C} \to \mathbb{D}$

$$(G^F) \circ A \xrightarrow{\sim} (G \circ A)^{F \circ A}$$

NOW $\mathcal{M}^{\mathbb{C}}(X, G A^{FA}) \simeq \mathcal{M}^{\mathbb{C}}(X \times FA, GA)$

$$\simeq \mathcal{M}^{\mathbb{D}}(L_A(X \times FA), G)$$

AND $\mathcal{M}^{\mathbb{C}}(X, G^{FA}) \simeq \mathcal{M}^{\mathbb{D}}(L_A X, G^F)$

$$\simeq \mathcal{M}^{\mathbb{D}}(L_A X \times F, G)$$

SINCE THIS IS TO HOLD FOR ALL G

$$L_A(X \times FA) \xrightarrow{\sim} L_A X \times F$$

TAKE SPECIAL CASE

$\mathbb{C} = \mathbb{1}$

$I = X : \mathbb{1} \to \mathcal{M}$ IS INCLUSION OF TERMINAL OBJECT.

WE HAVE $A : \mathbb{1} \to \mathbb{D}$

SO NOW

$$(L_A 1 \times FA|_{\mathbb{D}}) \xrightarrow{\sim} L_A(1 \times FA)$$

(USING BECK CONDITIONS FOR L_A)

$$\xrightarrow{\sim} L_A 1 \times F$$

$\mathbb{D}$ $\mathcal{M}$-enriched $\Rightarrow$ $L_A(1) \simeq \mathbb{D}(A, -)$

TAKE $F = \mathbb{D}(d, -)$

GET $\mathbb{D}(A, -) \times \mathbb{D}(d, A)$

$$\simeq \mathbb{D}(A, -) \times \mathbb{D}(d, -)$$

SO $\mathbb{D}(A, d) \times \mathbb{D}(d, A)$

$$\simeq \mathbb{D}(A, d) \times \mathbb{D}(d, d)$$

SO EVERY MAP HAS A RT INVERSE

□

8.

EXAMPLE : WE NEED SOME CONDITION ON $\mathbb{D}$.

- SUPPOSE ONLY FUNCTORS $\mathbb{D} \longrightarrow \mathbb{M}$ ARE CONSTANT ....

SO CAN'T HAVE TYPES WITH ONE FREE TYPE VARIABLE

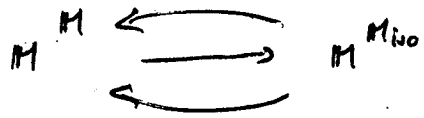
$\mathbb{M}^{\mathbb{M}}$

CAN HAVE

$\mathbb{M}^{\mathbb{M}_{iso}}$

$\mathbb{M}_{iso}$ - SAME OBJECTS AS $\mathbb{M}$
MORPHISMS ARE ISOS OF $\mathbb{M}$.

BUT



10.

THIS GIVES

$$[\![\forall X . F]\!] = \{ (a_M)_{M \in \mathbb{M}} \mid \forall f : M \xrightarrow{\sim} M' \quad F(f)\, a_M = a_{M'} \}$$

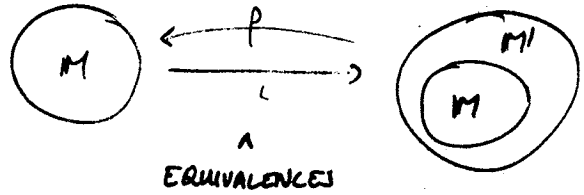
11.

ALTERNATIVE CHARACTERIZATION:

WE EXPECT

$$[\![\forall X . F]\!] \rightarrowtail \prod_{X \in \mathbb{M}_0} F(X)$$

CONSIDER ADJOINING EXTRA COPIES OF TYPES



LIFT $(a_M)_{M \in \mathbb{M}_0}$ TO $\mathbb{M}'$ ALONG ρ

$(b_{M'})_{M' \in \mathbb{M}'_0}$

WHERE $b_{M'} = a_{\rho(M')}$

DEMAND IMPLEMENTATION INDEPENDENCE

IE GIVEN ρ', LIFT ALONG ρ' = LIFT ALONG ρ.

SUBSET OF PRODUCT FOR WHICH THIS IS ALWAYS TRUE IS

$$\{ (a_M)_{M \in \mathbb{M}_0} \mid \forall M \xrightarrow[f]{\sim} M' . \; F(f)\, a_M = a_{M'} \}.$$

SEQUENTIALITY, GAMES AND LINEAR LOGIC

François Lamarche

LIENS, 45 rue d'Ulm, Paris 75230 Cedex 05

The category of CDSes and sequential functions was defined by Kahn-Plotkin. Unfortunately it was not cartesian-closed. Berry-Curien corrected this problem by adding intensional information to the functions. In their model the set of morphisms $X \to Y$ is obtained by constructing the CDS $X \Rightarrow Y$ and looking at its "elements". It is Bucciarelli-Ehrhard [TCS, to appear] who gave an intrinsic definition of a sequential algorithm $X \to Y$. In this note we describe what these linear sequential algorithms are and construct a model of full linear logic.

We will use modified Conway games. We start with the following three ingredients:

1) a set $(X, \leq)$ which is a <u>forest</u>. That is $p\downarrow$ is a finite totally ordered set for every $p \in X$. We define the predecessor

$$p^- = \text{largest } q < p \text{ if it exists.}$$

and get a partition $X = \overset{\dagger}{\bigcup}_{n \geq 0} X_n$ where

$$p \in X_0 \Leftrightarrow p^- \text{ undefined} \qquad p \in X_{n+1} \Leftrightarrow p^- \in X_n .$$

2) A function $Col : X \to \{\bullet, \circ\}$ which is <u>alternating</u> :

$$Col(p^-) \neq Col(p)$$

this should be read

$Col(p) = \bullet$ { p is a <u>value</u>; p is one of my moves; p has Danos-Régnier polarity O (output) }

$Col(p) = \circ$ { p is a <u>cell</u>; p is one of my opponent's moves; p has Danos-Régnier polarity I (input) }

3) A coherent domain structure $\frown$ on $X_\circ$. That is, $\frown$ is an <u>anti</u>reflexive, symmetric binary relation on $X_\circ$. A subset $U \in X_\circ$ is said to be <u>coherent</u> if $p, q \in U.\ p \neq q \Rightarrow p \frown q$.

Notice that given $p, q \in X$ then the inf $p \wedge q$ exists iff p, q lie in the same connected component of $(X, \leq)$. We write X^* for the set of values. X°_n for the set of cells of height n, etc...

An ANSWER is a subset $x \subset X^*$ such that

- it is down-closed : $p \in x \Rightarrow p^{--} \in x$ if it exists.
- if $p, q \in x$ then
  - $p \wedge q$ defined $\Rightarrow$ $p \wedge q$ is a <u>value</u>
  - $p \wedge q$ undefined $\Rightarrow$ $p_\circ \frown q_\circ$ where $p_\circ, q_\circ \in X_\circ$, $p_\circ \leq p$, $q_\circ \leq q$

We denote the set of answers by X_*

The domain-theoretic interpretation of this is that if $p \in x$ then the value p occupies the cell p^-. The condition on $p \wedge q$ being a value means that in a given answer a cell can be filled by at most one value.

Given a game $X = (X, \leq, Col, \frown)$ we define its orthogonal to be the game $X^\perp = (X, \leq, \overset{\text{exchange}}{\bullet\circ}, \smile)$ where $\smile$ is the coherent domain structure on $X_\circ$ orthogonal to $\frown$:

$$p \smile q \quad \text{iff} \quad p \neq q,\ p \not\leq q$$

A QUESTION is an answer in $X^\perp$. That is, it is a $\smile$-coherent, down-closed collection of cells that meet over cells. The space of questions is denoted by X^*. Both X_* and X^* have a natural order structure given by inclusion. Here is a bit of domain-theoretical taxonomy for experts in these things : $(X_*, \leq)$ and $(X^*, \leq)$ are always Scott domains, more precisely dI-domains.

There are standard embeddings $X^\bullet \hookrightarrow X_*$, $X^\circ \hookrightarrow X^*$ whose images are the sets of primes (also called complete coprimes) in the respective domains.

PROPOSITION

Given $x \in X_*$, $\alpha \in X^*$ then $x\!\downarrow \cap\, \alpha\!\downarrow$ is a path in X, that is, a totally ordered subset of X. □

(Y)

This allows us to interpret an answer x as a (partial) playing strategy for me, a question as a partial playing strategy for my opponent, and $x \lfloor \frown a \rfloor$ as the actual play-by-play outcome of the game (seen as an increasing sequence of moves) that arises when the strategies are tried on each other.

Since we made no well-foundedness assumption a game (COMPUTATION) may last forever. In this model the Conway concept of winning is irrelevant... although this may not be the case for other related models.

Notice that in general the starting player is defined only at match time.

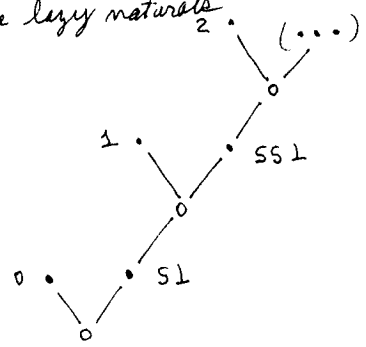
EXAMPLES

A filiform, prime CDS is a game such that

(a) $X_0 = X_0^0$ (b) All cells have a scalar above them

(c) $p \frown q \Rightarrow p \neq q$ in X_0

For example the lazy naturals



Given two games X, Y we define $X \multimap Y$ as follows:

$X \multimap Y_n$ is the set of all (non empty!) sequences $\langle (p_0,q_0)(p_1,q_1)\dots(p_m,q_m)\rangle$ of pairs $p_i \in X$, $q_i \in Y$ (where the operation $(-)^-$ is "remove the last pair of the list") satisfying in addition the conditions

1) (p_i, q_i) can only be $(\cdot,\cdot)$ $(0,0)$ or $(\cdot,0)$

The color of the prefix sequence ending with (p_i,q_i) will be • or ○

2) $(p_0,q_0) \in X_0 \times Y_0$ and the coherent domain structure is the standard one for $\multimap$: $(p,q) \frown (p',q')$ iff $p \frown p' \Rightarrow q \frown q'$.

3) You can move up by only one step on one side. That is,

$(p_{i+1} = p_i, q_{i+1} \neq q_i)$ or $(p_{i+1} \neq p_i, q_{i+1} = q_i)$

Thus the answers of $(X \multimap Y)$ are the linear sequential algorithms $X \to Y$. We have to know how to compose them. Let $f \in (X \multimap Y)_*$. Define

$$f^{\#} = \{(p,q) \in X \times Y \mid (p,q) \text{ appears in a sequence in } f\}$$

(6)

FACT

f is entirely determined by $f^\#$.

Let us say a little bit more why this is so. It turns out that $f^\# \cap X^\bullet \times Y^\bullet$ is the trace of a linear function $X_* \xrightarrow{f_*} Y_*$.

Recall that a function between Scott domains is linear iff it preserves all sups that exist and all the infs bounded from above (pullbacks). Equivalently, since we have enough primes in our domains, f_* is linear iff for all primes $q \in Y_*$, $f_*^{-1}(q\uparrow)$ is a disjoint union $\overset{+}{\bigcup_{i\in I}} p_i\uparrow$ where the p_i are primes in X_*. Given $f^\#$ it is easy to see what f_* should be: $f_*(x) = \{q \mid (p,q) \in f^\# \text{ and } p \in x\}$.

Conversely we can recover $f^\# \cap X^\bullet \times Y^\bullet$ from f_* by taking the trace

$$Tr(f_*) = \{(p,q) \in X^\bullet \times Y^\bullet \mid p \text{ is one of the } p_i \text{ in the disjoint union } f_*^{-1}(q\uparrow) = \overset{+}{\bigcup_i} p_i\uparrow\}$$

(Here we identify the values of X, Y with the primes they determine in X_*, Y_*).

The same goes for $f^\# \cap X^\circ \times Y^\circ$ and we get $X^* \xleftarrow{f^*} Y^*$ linear.

(7)

Thus, since $f^\# = \left(f^\# \cap X^\bullet \times Y^\bullet\right) \cup \left(f^\# \cap X^\circ \times Y^\circ\right)$ we can describe a linear sequential algorithm as a pair (f_*, f^*) of linear functions whose traces merge well. In particular, given $f \in (X \multimap Y)_*$, if $f^\# \subset X \times Y$ is given the ordinary product ordering then it is a forest, such that given $(p,q) \in f^\#$ we have that

$$(p,q)^- = \begin{cases} (p, q^{--}) & \text{if } (p,q), (p,q)^- \text{ are } (\bullet\circ) \\ (p^{--}, q) & \text{if } (p,q), (p,q)^- \text{ are } (\circ\bullet) \\ (p^-, q^-) & \text{if } (p,q), (p,q)^- \text{ are different colors} \end{cases}.$$

An additional condition is that the roots of that forest have to be coherent in a way that the reader can easily make explicit.

If $g \in (Y \multimap Z)_*$ we can get gf by looking at the composites $f^* \circ g^*$, $f_* \circ g_*$. Since traces compose as graphs, $(gf)^\#$ is the composite $g^\# \circ f^\#$ of the graphs $g^\#, f^\#$.

Since we have $(-)^\perp$ and $\multimap$ the other multiplicatives will follow automatically. The underlying sets are constructed the same way except that rules 1) and 2) are now

(8)

for $\otimes$	for $\parr$
use only $(\bullet,\bullet)$ $(\circ,\bullet)(\circ,\bullet)$	$(\circ,\circ)$ $(\bullet,\circ)(\bullet,\circ)$
and get $\bullet$ $\circ$	$\circ$ $\bullet$
standard $\otimes$-rule for $\frown$	standard $\parr$-rule for $\frown$

The unit for $\otimes$ is $I = \{\bullet\}$ and the dualizing object $\bot = \{\circ\}$.

The additives are constructed just as expected. Given X, Y then $X \oplus Y$, $X \& Y$ are constructed on the disjoint sum $(X+Y, \leq)$ by taking their respective coherent domain structures on $X_0 + Y_0$.

Now for the exponentials. First, given $x \in X_*$ and $q \in X^\circ$ we say that q is enabled by x, and write $x \vdash q$, when

$$q \notin x\downarrow \quad \text{and} \quad \begin{cases} q^- \in x & \text{or} \\ q \in X_0 \text{ and } q \cup (x \cap X_0) \text{ is a consistent set} \end{cases}$$

In other words q is immediately accessible from x and filling q will yield an answer.

then $!X$ is given by

$$(!X)_\circ = \{ u \in X_\circ^\bullet \mid u \text{ is finite \& coherent} \}$$

with usual coherence relation. $u \frown v$ iff $u \neq v$ and $u \cup v$ is coherent. Notice that if X is a CDS then $!X$ will have the unique element $\emptyset$. Everything in $(!X)_\circ$ has color $\bullet$.

$$(!X)_{2m} = \{ (u, x, \langle p_1 \cdots p_m \rangle) \mid u \in X_\circ \,,\ x = u \cup \{p_1 \cdots p_m\} \text{ and } x \in X_* \,,\ \langle p_1 \cdots p_m \rangle \text{ is a bijective path in } x - u \text{ which is always consistent: } \forall i \quad u \cup \{p_1 \cdots p_i\} \in X_* \}$$

$$(!X)_{2m+1} = \{ (u, x, \langle p_1 \cdots p_m \rangle, q) \mid u, x, \langle p_1 \cdots p_m \rangle \text{ as above and } x \vdash q \}$$

we define $(u, x, \langle p_1 \cdots p_m \rangle, \alpha)^- = (u, x, \langle p_1 \cdots p_m \rangle)$

$$(u, x, \langle p_1 \cdots p_m \rangle)^- = (u, x, \langle p_1 \cdots p_{m-1} \rangle, p_m^-)$$

Given $f \in (!X \multimap Y)_*$, $g \in (!Y \multimap Z)$ we can get the Kleisli composite $gf \in (!X \multimap Z)_*$ by suitably combining the traces $f^\#, g^\#$. From this we can deduce the full comonad structure. $f^\#$ and $g^\#$ can also be thought of as the traces of (slightly generalized) Buchholz-Ehrhard algorithms, whose composition is obvious.

J. Raes.

A logical framework
based on categories
with structure

A notion of map between logic programs (joint with Leon Sterling)

Aim: To develop systematic programming methodology for logic programming, i.e. structured programming.

Basic idea to start:

Some programs essentially have same computations, e.g.

(1) conn(x,y) ← edge(x,z), conn(z,y)
conn(x,x) ←

(2) num-conn(x,y,n+1) ← edge(x,z), num-conn(z,y,n)
num-conn(x,x,0) ←

or paths

but not (2') num-conn(x,y,n) ← edge(x,z), num-conn(x,y,n-1)
num-conn(x,z,0) ←

Leading to meta-interpreters

Need formal def'ns

Assume given inf sets V, Γ, A of vars, fn symbols & pred symbols.
A base is a finite set of atomic fmlae
A definite logic program P:
P = (V_P, Γ_P, B_P, C_P):
V_P ⊆ V, Γ_P ⊆ Γ, B_P is a base with vars & fn symbols only from V_P & Γ_P (all finite)
C_P = set of definite clauses, all of whose at fmlae are in B_P.

A P-substitution is a subst. that fixes V \ V_P & moves V_P into T_P, set of terms determined by V_P & Γ_P.

Denote by M_P the monoid of P-substs.
& Σ_P the ~~"A semilattice determined by finite conjuncts of P-s~~ total set given by free split fib with strictly basic fn prods as sketch given by C_P.

We defined a little differently. [Finite sets of P instances of basic fmlae ~~only considered~~ maps reducing clauses to consequence

Def A program map R: P → Q consists of
1) a monoid map R: M_P → Q_P
2) a map of ∧-semilattice R: Σ_P → Q_P s.t.

$$\begin{array}{ccc} (\sigma_P, X) & M_P \times \Sigma_P & \longrightarrow M_Q \cdot \Sigma_Q \\ \downarrow & \downarrow & \downarrow \\ X\sigma & \Sigma_P & \longrightarrow \Sigma_Q \end{array}$$

subst

commutes,
i.e. R(Xσ) = R(X)R(σ).

[Idea of map was central.]

Def Given X in Σ_P, Y in Σ_Q, say X is minimal over Y if R(X) = Y & for any subset X' of X, R(X') = Y ⇒ X' = X.
(As R induces a bijection).

Def A program map is an extension if
1) R(σ) invertible in M_Q ⇒ σ invertible in M_P
& R takes basic fmlae in P to either basic fmlae in Q or T
2) X minimal over Y, H ← Y clause in Q ⇒ ∃! clause A ← Z in P over H ← Y with X ⊆ Z & R(Z \ X) = T.
3) B basic, σ ∈ M_P, X min over BR(σ) ⇒ ∃ A σ_A. Aσ = X & R(A) = B
4) Ψ = mgu(BR(σ), B') ⇒ ∃ η. M_P R(η) = Ψ.

Def A (SLD-derivation) computation in P with goal G (where G is an instance of a basic fmla) consists of a seq
(A_k, σ_k, C_k, R_k) | 1 ≤ k ≤ n with
R_0 = G
∀ k > 0 A_k ∈ R_{k-1}, σ_k ∈ M_P, C_k is a clause H_k ← B_k in P,
R_k ∈ Σ_P s.t.
1) σ_k = mgu(A_k, H_k)
2) R_k = (R_{k-1} \ A_k ∪ B_k) σ_k.

Call it a refutation if R_n = T.

(3)

Thm $R : P \to Q$ an ext'n, G in $\&Q$ of form $R(\vec{t})$ with P basic & $\vec{t}$ comp.
Then $\forall$ comp $(A_R, \sigma_R, C_R, R_R)$ in Q with good G
& given minimal ctt S over R_R,
$\exists G'$ & comp $W' = (A_P', \sigma_P', C_P', R_P')$ in P with good G'
with W over W', $S \subseteq R_P'$ & $R(R_P' | S) = T$.

Sketch of proof
① Starting with σ_0, show all σ's of form $R(\vec{\sigma})$.
② Starting with S_1, introduce A_P's & C_P's.
③ This data determines R_P's. Verify, starting at ①, that this is a computatn

Cor 1: Refutation lifts to computation.
Cor 2 If $R(X) = T$ iff $X = T$, then refutation lifts to refutation.
Lem: $R : P \to Q$ an ext'n, F an ~~basic~~ instance of a basic finite in P
$\Rightarrow P + F \longrightarrow Q + R(F)$ an ext'n.

Composite of extension are extensions. This is implemented.

List of Participants

CLICS WORKSHOP
AARHUS UNIVERSITY, DENMAR ς
23 - 27. MARCH 1992

Samson Abramsky
Dept. of Computing
Imperial College
180 Queen's Gate
London SW7 2BZ
ENGLAND

Petet Aczel
Department of Computer Science
Manchester University
Manchester, M13 9PL
ENGLAND

Roberto Amadio
CRIN, B.P. 239
F-54506
Vandoeuvre-les-Nancy Cedex
FRANCE

Henrik Reif Andersen
Computer Laboratory,
University of Cambridge,
New Museums Site,
Pembroke Street,
Cambridge,
ENGLAND

Nick Benton
Cambridge University
Computer Laboratory
New Museums Site
Pembroke Street
Cambridge CB2 3QG
ENGLAND

Gavin Bierman
University of Cambridge
Computer Laboratory,
New Museums Site,
Pembroke Street,
Cambridge, CB2 3QG
ENGLAND

Stephen Brookes
Carnegie Mellon University
School of Computer Science
Schenley Park
Pittsburgh
Pennsylvania 15213
USA

Carolyn Brown
Department of Computer Science
Chalmers University of Technology
S-412 96 Gøteborg
SWEDEN

Antonio Bucciarelli
Laboratoire d'Informatique
Ecole Normale Superieure
45, Rue D'Ulm
75005 Paris
FRANCE

Thierry Coquand
CTH, Computer Scince Department
S 41296 Gøteborg,
SWEDEN

Roy Crole
Department of Computing
Imperial College
180 Queen's Gate
London SW7 2BZ
London
ENGLAND

Peter Dybjer
Department of Computer Science
Chalmers University of Technology
S-412 96 Gøteborg
SWEDEN

Matthias Felleisen
Department of Computer Science
Rice University
Houston, TX 77251-1892
USA

Marcelo Fiore
Laboratory for Foundations of
Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ
SCOTLAND

Michael P. Fourman
Department of Computer Science
5198 (sec)
JCMB,
King's Buildings, Mayfield Road
Edinburgh EH9 3JZ
SCOTLAND

Peter Freyd
Department.of Mathematics
University of Pennsylvania
Philadelphia
PA 19104-6395
USA

Jean-Yves Girard
16b rue Sibuet
75012 Paris
FRANCE

Douglas Gurr
10 De Beauvoir Court
North Church Road
Islington, N1 3NX
ENGLAND

Barney Hilken
Department of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL
ENGLAND

Martin Hyland
Department of Pure Mathematics
and Mathematical Statistics
University of Cambridge
16 Mill Lane, Cambridge CB2 1SB
ENGLAND

Bart Jacobs
Department of Pure Mathematics
and Mathematical Statistics
University of Cambridge
16 Mill Lane, Cambridge CB2 1SB
ENGLAND

Barry Jay
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ
SCOTLAND

Peter Johnstone
Department of Pure Mathematics
and Mathematical Statistics
University of Cambridge
16 Mill Lane, Cambridge CB2 1SB
ENGLAND

Andre Joyal
Department of Pure Mathematics and
Mathematical Statistics
University of Cambridge
16 Mill Lane, Cambridge CB2 1SB
ENGLAND

Yves Lafont
Laboratoire d'Informatique
Ecole Normale Superieure
45, Rue D'Ulm
75005 Paris
FRANCE

Francois Lamarche
Laboratoire d'Informatique
Ecole Normale Superieure
45, Rue D'Ulm
75005 Paris
FRANCE

Guiseppe Longo
Laboratoire d'Informatique
Ecole Normale Superieure
45, Rue D'Ulm
75005 Paris
FRANCE

Pasquale Malacaria
Laboratoire d'Informatique
Ecole Normale Superieure
45, Rue D'Ulm
75005 Paris
FRANCE

Tom Melham
University of Cambridge
Computer Laboratory
New Museums Site
Cambridge CB2 3QG
ENGLAND

Maria Claudia Mere
Computer Laboratory
University of Cambridge
New Museums Site
Cambridge CB2 3QG
ENGLAND

Eugenio Moggi
Dip. di Matematica
Univ. de Genova
v. L.B. Alberti 4,
16132 Genova,
ITALY

Philip Mulry
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ
SCOTLAND

David Murphy
GMD 15
Schloss Birlinghoven
Postfach 1316
D-5205 Sankt Augustin 1
GERMANY

Valeria de Paiva
Computer Laboratory
University of Cambridge
New Museums Site
Cambridge
CB2 3QG
ENGLAND

Wesley Phoa
Department of Computer Science
JCMB, The King's Buildings
Mayfield Road
Edinburgh EH9 3JZ
SCOTLAND

Andrew M. Pitts
Computer Laboratory
University of Cambridge
New Museums Site
Pembroke Street
Cambridge CB2 3QG
ENGLAND

Gordon Plotkin
Department of Computer Science
The King's Buildings
Edinburgh EH9 3JZ
SCOTLAND

John Power
Department of Computer Science
The King's Buildings
Edinburgh EH9 3JZ
SCOTLAND

Vaughan Pratt
Department og Computer Science
Stanford University
Stanford, CA 94305
USA

Eike Ritter
Computer Laboratory
University of Cambridge
New Museums Site
Pembroke Street
Cambridge CB2 3QG
ENGLAND

Edmund Robinson
School of Cognitive and
Computing Sciences
University of Sussex
Falmer, Brighton BN1 9QH
ENGLAND

Guiseppe Rosolini
Dip. di Matematica
Universita'
43100 Parma
ITALY

David Rydeheard
Department of Computer Science
The University
Oxford Road
Manchester M13 9PL
ENGLAND

Andrea Schalk
Fachbereich Mathematik
Technische Hochschule Darmstadt
Schlossgartenstrasse 7
D-6100 Darmstadt
GERMANY

Philip J. Scott
Department of Computer Science,
LFCS,
University of Edinburgh
SCOTLAND

Harold Simmons
Department of Computer Science
The University
Manchester M13 9PL
ENGLAND

Ian Stark
Computer Laboratory
Cambridge University
Pembroke Street
Cambridge CB2 3QG
ENGLAND

Thomas Streicher
Fakultaet f. Mathematik und
Informatik
Postfach 2540
Universität Passau
Innstr. 33
W-8390 Passau
GERMANY (West)

Paul Taylor
Dept. of Computing
Imperial College
180 Queen's Gate
London SW7 2BZ
ENGLAND

Robert Tennent
LFCS, Computer Science,
University of Edinburgh,
Edinburgh,
SCOTLAND EH9 3JZ

Steven John Vickers
Department of Computing
Imperial College
180 Queen's Gate
London
SW7 2BZ
ENGLAND

Philip Wadler
Dept. of Computing Science
University of Glasgow
Glasgow, G12 8QQ
SCOTLAND