Categories

Pierre-Louis Curien
πr^2, Preuves, Programmes et Systèmes
INRIA, University Paris 7, **CNRS**

April 21, 2014

What for?

Category theory provides a way to organise mathematical structures in general, and in particular the mathematical structures involved in the design, the understanding, and even the implementation of programming languages.

And this week's lectures are very much about programming languages (high and low-level, pure an "impure").

Examples of categories

- sets and functions between them $\mathbf{Set}$
- sets and partial functions
- sets and relations $\mathbf{REL}$
- posets and monotonous functions between them
- topological spaces and continuous functions
- groups (or rings, fields, Lie algebras..) and morphisms between them
- categories and functors $\mathbf{Cat}$
- functors between two fixed categories and natural transformations between these functors

This morning (lectures 1 and 2): basic definitions and properties

- categories
- functors
- natural transformations
- presheaves and Yoneda lemma + embedding
- 2-categories
- adjoint functors, equivalences of categories
- monads (and comonads), Kleisli categories, algebras and coalgebras
- distributive laws

I'll use string diagrams to provide graphical proofs

Other important notions

- limits and colimits
- Kan extensions
- ends and coends
- enriched categories
- bicategories, n-categories (strict and weak)

Up to ends and coends included, you can consult my notes

Category theory: a programming language-oriented introduction

on my web page (google Pierre-Louis Curien)

Preview: Lecture 3

We shall define cartesian closed categories (CCC), we shall present:

- the syntax of categorical combinators
- the compilation of the λ-calculus into this syntax
- the execution of the categorical **code** in an abstract machine, the CAM (Categorical Abstract Machine) (Cousineau-Curien-Mauny)

We'll show a conservative extension result: every category embeds fully and faithfully in the free CCC over it. Moreover, the proof "contains" a proof of termination of the CAM (**Lafont**)

Preview: Lecture 4

Where categories meet computation: further instances

- normalisation by evaluation
- coherence in monoidal categories
- monads and effects (see Alex's lectures!)

Preview: Lectures 5 and 6

- A short introduction to linear logic
- Denotational semantics for intuitionistic, linear, (and classical, not covered here) logic
- The full abstraction problem
- denotational models that arose in the quest of full abstraction: continuity, stability, sequentiality / game semantics

Category = multi-graph + monoid

A **category** $\mathbf{C}$ is a structure consisting of objects $A : \mathbf{C}$, of morphisms $f : A \to B$, with the following operations and equations on morphisms:

$$\frac{}{id_A : A \to A} \qquad \frac{f : A \to B \quad g : B \to C}{g \circ f : A \to C}$$

$$\frac{f : B \to A \;\; g : C \to B \;\; h : D \to C}{(f \circ g) \circ h = f \circ (g \circ h) : D \to A}$$

$$\frac{f : A \to B}{f \circ id_A = f : A \to B} \qquad \frac{f : A \to B}{id_B \circ f = f : A \to B}$$

Notation: $Obj(\mathbf{C})$ for the collection of objects, $Hom_{\mathbf{C}}(A, B)$ or $\mathbf{C}[A, B]$ for the collection of morphisms from A to B (dependent type!)

Examples of categories (three last coming next)

- a monoid is a one-object category
- a preorder is a category (cf. homotopy type theory, where sets are special groupoids)
- sets and functions between them $\mathbf{Set}$
- sets and partial functions
- sets and relations $\mathbf{REL}$
- posets and monotonous functions between them

⋮

- the dual of a category is a category
- categories and functors $\mathbf{Cat}$ (next)
- functors between two fixed categories and natural transformations between these functors (next after next)

Dual properties

If $\mathbf{C}$, we deine $\mathbf{C}^{op}$ as follows:

$$Obj(\mathbf{C}^{op}) = Obj(\mathbf{C}) \quad \mathbf{C}^{op}[A, B] = \mathbf{C}[B, A]$$

$$id^{op} = id \qquad\qquad f \circ^{op} g = g \circ f \ .$$

Given a property P, one defines its **dual** P^{op} by: P^{op} holds in $\mathbf{C}$ iff P holds in $\mathbf{C}^{op}$.

Example: mono is dual to epi:

f is mono if whenever $f \circ g = f \circ h$ then $g = h$

f is epi if whenever $g \circ f = h \circ f$ then $g = h$

Functors

Let $\mathbf{C}$, $\mathbf{D}$ be categories. A **functor** $F : \mathbf{C} \to \mathbf{D}$ is a graph morphism that preserves the operations:

$$\frac{A : \mathbf{C}}{FA : \mathbf{D}} \qquad \frac{f : A \to B}{Ff : FA \to FB}$$

$$\frac{}{Fid_A = id_{FA} : FA \to FA} \qquad \frac{f : B \to C \quad g : A \to B}{Ff \circ Fg = F(f \circ g) : FA \to FC}$$

Examples of functors

- Forgetful functors, e.g. $U : \mathbf{Top} \to \mathbf{Set}$
- the geometric realisation functor from simplical sets to topological spaces
- the nerve functor from categories to simplicial sets
- the singular homology functor from topological spaces to the category of (chain complexes of) groups

⋮

- Homfunctor $HomLeft_A = \mathbf{C}[\_, A] : \mathbf{C}^{op} \to \mathbf{Set}$:

$$HomLeft(B) = \mathbf{C}[B, A]$$
$$HomLeft(f)(g) = g \circ f \quad (f : C \to B \ , \ g \in HomLeft(B))$$

and similarly $\mathbf{C}[B, \_] : \mathbf{C} \to \mathbf{Set}$

- The projection functors $\mathbf{C} \times \mathbf{D} \to \mathbf{C}$ and $\mathbf{C} \times \mathbf{D} \to \mathbf{D}$
- Functor $\_ \times A : \mathbf{C} \to \mathbf{C}$ in a category $\mathbf{C}$ with products (tomorrow)

Natural transformations

Let $F, G : \mathbf{C} \to \mathbf{D}$. A **natural transformation** $\mu : F \to G$ is a family of morphisms $\{\mu_A : FA \to GA\}_{A:\mathbf{C}}$ (called the components of μ) such that, for all $A, B, f : A \to B$, the following square commutes:

$$Gf \circ \mu_A = \mu_B \circ Ff$$

Example of a natural transformation. Take the functor $List : \mathbf{Set} \to \mathbf{Set}$. Then the family $rev_X : List(X) \to List(X)$ (list reversing) is a natural transformation.
Natural transformations compose vertically:

$$id_A = id_{FA} \qquad (\nu \circ \mu)_A = \nu_A \circ \mu_A$$

Thus we have a category $\mathbf{D}^{\mathbf{C}}$ whose objects are the functors $F : \mathbf{C} \to \mathbf{D}$, and $\mathbf{D}^{\mathbf{C}}[F, G]$ is the collection of natural transformations $\mu : F \to G$.

Natural transformations as 2-morphisms

Natural transformations also compose horizontally.
1) one defines

$$\mu H : F \circ H \to G \circ H \quad \text{by} \quad (\mu \cdot H)_A = \mu_{HA}$$
$$K\mu : K \circ F \to K \circ G \quad \text{by} \quad (K \cdot \mu)_A = K(\mu_A)$$

2) for $F, F' : \mathbf{C} \to \mathbf{C}'$, $G, G' : \mathbf{C}' \to \mathbf{C}''$, $\mu : F \to F'$, $\nu : G \to G'$, we define $\nu \cdot \mu : GF \to G'F'$ as follows (by naturality of ν):

$$(\nu F') \circ (G\mu) \quad = \quad \nu \cdot \mu \quad = \quad (G'\mu) \circ (\nu F)$$

3) for $F, F', F'' : \mathbf{C} \to \mathbf{C}'$, $G, G', G'' : \mathbf{C}' \to \mathbf{C}''$, $\mu : F \to F'$, $\mu' : F' \to F''$, $\nu : G \to G'$ and $\nu' : G' \to G''$, one proves the following, called exchange law:

$$(\nu' \circ \nu) \cdot (\mu' \circ \mu) \quad = \quad (\nu' \cdot \mu') \circ (\nu \cdot \mu)$$

This says that pasting diagrams make sense.

n-categories

2-categories have 0-morphisms, 1-morphisms that compose, 2-morphisms that compose in two ways in a compatible way.

$\mathbf{Cat}$ forms a 2-category

2-categories with one 0-morphism only are (strict) monoidal categories. We'll study them soon

1-categories have 0-morphisms (the objects), and 1-morphisms

3-categories have 0-morphisms, 1-morphisms, 2-morphisms, and 3-morphisms (that compose in 3 different ways) etc...

Weak 2-categories

But once you have 2-morphisms around, you can discuss a weaker form of associativity for 1-morphisms. Instead of requiring

$$(h \circ g) \circ f = h \circ (g \circ f)$$

we ask only the existence of a (natural) invertible 2-morphism

$$\alpha : (h \circ g) \circ f \to h \circ (g \circ f)$$

and the same for the $f \circ id$ and $id \circ f$. These natural transformations have to satisfy some laws, called coherence laws.

This leads to bicategories, a.k.a. weak 2-categories, and then weak n-categories: the problem becomes the unmanageability of the coherence laws when n grows.

Bicategories with one 0-morphism only are monoidal categories.

Presheaves

Let $\mathbf{C}$ be a category. The functors $F : \mathbf{C}^{op} \to \mathbf{Set}$ are called (contravariant) **presheaves** over $\mathbf{C}$.

Examples of presheaves:

• the hom functors $\mathbf{C}[\_, C]$: they are called the representable presheaves.

• If $\mathbf{C}$ is X^* for some alphabet X, with the prefix ordering, then a presheaf over X is a synchronisation forest ($\to$ concurrency theory)

Yoneda lemma

For any presheaf F we have natural bijections:

$$FC \cong \mathbf{Set}^{\mathbf{C}^{op}}[\mathbf{C}[\_, C], F] \ .$$

See my notes for a graphical proof.

Application: $\mathbf{Set}^{\mathbf{C}^{op}}$ has function spaces (anticipating lecture 3). By Yoneda we know that $(G^F)(C)$ must be in bijective correspondence with

$$\mathbf{Set}^{\mathbf{C}^{op}}[\mathbf{C}[\_, C], G^F]$$

which by uncurrying must be in bijective correspondence with

$$\mathbf{Set}^{\mathbf{C}^{op}}[\mathbf{C}[\_, C] \times F, G]$$

Thus we define (cf. Kripke semantics)

$$(G^F)(C) = \mathbf{Set}^{\mathbf{C}^{op}}[\mathbf{C}[\_, C] \times F, G]$$

Yoneda embedding

But the most important application of Yoneda lemma is

The functor $Yon : \mathbf{C} \to \mathbf{Set}^{\mathbf{C}^{op}}$ is full and faithful, where Yon is defined by

$$Yon(C) = \mathbf{C}[\_, C]$$

Proof: By specialising the Yoneda lemma to $F = \mathbf{C}[\_, D] = Yon(D)$, we get a bijection

$$\mathbf{C}[C, D] = FC \cong \mathbf{Set}^{\mathbf{C}^{op}}[\mathbf{C}[\_, C], F] = \mathbf{Set}^{\mathbf{C}^{op}}[\, Yon(C),\, Yon(D)]$$

Adjoint functors: definition 1

Let $\mathbf{C}$, $\mathbf{C}'$ be categories, and let $G : \mathbf{C}' \to \mathbf{C}$. Then we say that G has a **left adjoint** if for every object $C : \mathbf{C}$ there exists an object, denoted as FC, together with a morphism $\eta_C : C \to GFC$, such that for every morphism $g : C \to GC'$ in $\mathbf{C}$ there exists a unique morphism $f : FC \to C'$ in **C'** such that

$$Gf \circ \eta_C = g$$

Example: the forgetful functor U from the category of monoids and monoid morphisms to $\mathbf{Set}$ has a left adjoint (the free monoid construction).

Adjoint functors: definition 2

An **adjunction** between two categories $\mathbf{C}, \mathbf{C}'$ is a quadruple (F, G, η, ϵ), where $F : \mathbf{C} \to \mathbf{C}'$ and $G : \mathbf{C}' \to \mathbf{C}$ are functors and $\eta : id_{\mathbf{C}} \to GF$ (the **unit**) and $\epsilon : FG \to id_{\mathbf{C}'}$ (the **counit**) are natural transformations such that

$$G\epsilon \circ \eta G = id_G \quad \text{and} \quad \epsilon F \circ F\eta = id_L$$

Notation $F \dashv G$

Exercise: give a graphical proof of the equivalence of the two definitions.

Examples of adjoint functors

• Free constructions (cf. above)

• Let $\mathbf{1}$ be the category with one object and one morphism (the identity). We say that $\mathbf{C}$ has a terminal object if the functor $F : \mathbf{C} \to \mathbf{1}$ has a right adjoint.

• Let $\Delta : \mathbf{C} \to \mathbf{C} \times \mathbf{C}$ be the diagonal functor. We say that $\mathbf{C}$ has binary products if Δ has a right adjoint, denoted as $\times$

• Let $\mathbf{C}$ be a category that has binary products. We say that it has exponents if for every object A the functor $\_ \times A$ has a right adjoint, denoted $\_^A$.

• Let $\mathbf{I}$ be a (small) category. We say that it has all limits of shape $\mathbf{I}$ if the diagonal functor $\Delta : \mathbf{C} \to \mathbf{C}^{\mathbf{I}}$ has a right adjoint

Some properties of adjoint functors

A functor is called **full** (resp. **faithful**) if its surjective (resp. injective) on morphisms.

For an adjunction $F \dashv G$ the following holds:

1) G is faithful if and only if every component of the counit is an epi.
2) G is full if and only if every component of the counit is a split mono (i.e., has a left inverse)/
3) G is full and faithful if and only if the counit is invertible.

Dually, F is faithful (resp. full) if every component of the unit is mono (resp. split epi), and is full and faithful iff η is invertible.

Equivalences of categories

The following properties of a functor $F : \mathbf{C} \to \mathbf{C}'$ are equivalent:

1) There exists a functor $G : \mathbf{C}' \to \mathbf{C}$ and two natural equivalences $\iota : GF \to id_{\mathbf{C}}$ and $\iota' : FG \to id_{\mathbf{C}'}$.
2) F is part of an adjunction $F \dashv G$ in which the unit and the counit are natural isomorphisms.
3) F is full and faithful and $\forall\, C' : \mathbf{C}'\ \exists\, C \in \mathbf{C}\ \ (C' \cong FC)$.

Monads

Let $\mathbf{C}$ be a category. A **monad** on $\mathbf{C}$ is a triple (T, η, μ) where $T : \mathbf{C} \to \mathbf{C}$, $\eta : id_{\mathbf{C}} \to T$ (the **unit**) and $\mu : TT \to T$ (the **multiplication**), and where η and μ satisfy the following three equations:

$$\mu \circ (\mu T) = \mu \circ (T\mu) \qquad \mu \circ (\eta T) = id_T \qquad \mu \circ (T\eta) = id_T$$

Dual notion: comonad

Every adjunction $(F : \mathbf{C} \to \mathbf{C}') \dashv (G : \mathbf{C}' \to \mathbf{C})$ gives rise to a monad on $\mathbf{C}$ (and to a comonad on $\mathbf{C}'$).

The monads arising from the free construction adjunctions are (quotient) term monads (unit says that variables are terms, multiplication is substitution)

Examples of monads

- The powerset functor on **Set**

- The list functor on **Set**

- Let S be a fixed set. The functor $(\_ \times S)^S$ gives rise to a monad on **Set** (state monad)

- Let R be a fixed set. The functor R^{R^-} gives rise to a monad on **Set** (continuation monad, cf. Alexandre's lectures)

Kleisli categories

Motivation: interpret programs $\Gamma \vdash M : A$ with effects as morphisms $\Gamma \to TA$ (Moggi).

But how to compose $f : A \to TB$ and $g : B \to TC$? As follows:

$$g \circ_K f = \mu \circ Tg \circ f$$

This gives rise to the Kleisli category $\mathbf{C}_T$ of a monad $T : \mathbf{C} \to \mathbf{C}$:

$$\frac{A : \mathbf{C}}{A : \mathbf{C}_T} \qquad \frac{f \in \mathbf{C}[A, TB]}{f \in \mathbf{C}_T[A, B]}$$

Identities are provided by the unit η.

Strong monads

Motivation: This is not enough to interpret programs with effects. How to interpret $M(N_1, N_2)$ where

$$M : (A \times B) \to C \qquad N_1 : A \qquad N_2 : B$$

We need a morphism $TB \times TC \to T(B \times C)$. Such a morphism (actually two, depending on the order of evaluation: N_1 before N_2, or conversely) can be obtained if we have a natural transformation (called **strength**)

$$s_B : A \times TB \to T(A \times B)$$

satisfying some properties: reformulating it as $s : FT \to TF$, we ask s to be a distributive law.

Distributive laws

Let C be a category. A (monad-functor) **distributive law** on $\mathbf{C}$ is a natural transformation $\lambda : TF \to FT$, where F is an endofunctor on $\mathbf{C}$ and (T, η, μ) is a monad on $\mathbf{C}$, that satisfies the following two equations:

$$\lambda \circ \eta F = F\eta \qquad \lambda \circ \mu F = F\mu \circ \lambda T \circ T\lambda$$

Equipping a functor with a distributive law is exactly what allows to lift it to the category of T-algebras (defined next).

Algebras over a monad

Let $T : \mathbf{C} \to \mathbf{C}$ be a monad. A **T-algebra** with carrier A is a morphism $\alpha : TA \to A$ such that

$$\alpha \circ \eta_A = id_A \qquad \alpha \circ \mu_A = \alpha \circ T\alpha$$

This gives rise to the category $\mathbf{C}^T$ of T-algebras and T-algebras morphisms (morphisms of $\mathbf{C}$ between the carriers that commute with the algebra structure).

Adjunctions arising from monads

Let T be a monad. Both the Kleisli category and the category of T-algebras give rise to adjunctions (whose associated monad is T)

• One can coerce $f \in \mathbf{C}[A, B]$ as $\eta \circ f : A \to TB$. This defines a functor $\mathbf{C} \to \mathbf{C}_T$, which has a right adjoint.

• The forgetful functor $\mathbf{C}^T \to \mathbf{C}$ that maps (A, α) to A has a left adjoint.

Cartesian closed categories (finite products + exponents)

• An object A in a category $\mathbf{C}$ is **terminal** if $\forall B \in \mathbf{C}\ \exists! f : B \to A$. We write $A = 1$ and $f =!$.We denote a terminal object with 1 and with $!_B$ the unique morphism from B to 1.

• Let A, B be two objects in a category $\mathbf{C}$. A **product** of A, B is a triple $(C, \pi : C \to A, \pi' : C \to B)$ such that for any triple $(D, f : D \to A, g : D \to B)$ there exists a unique $h : D \to C$ such that $\pi \circ h = f$ and $\pi' \circ h = g$. We write $C = A \times B$ and $h = \langle f, g \rangle$ (the pair of f, g).

• Let $\mathbf{C}$ be a category that has binary products. An **exponent** of two objects A, B is a pair $(C, ev : C \times A \to B)$ such that for any other pair $(C', f : C' \times A \to B)$ there exists a unique arrow $g : C' \to C$ such that $ev \circ \langle g \circ \pi, \pi' \rangle = f$. We write $C = B^A$ and $g = \Lambda(f)$.

Categorical combinators (syntax and typing)

We have a syntax for objects:

$$A ::= 1 \mid A \times A \mid A \to A$$

and for morphisms:

$$f ::= id \mid f \circ f \mid \langle f, g \rangle \mid \pi \mid \pi' \mid \Lambda(f) \mid ev \mid !$$

Typing rules (we write $A \vdash f : B$ for what we wrote as $f : A \to B$):

$$\frac{}{A \vdash id : A} \qquad \frac{A \vdash f : B \quad B \vdash g : C}{A \vdash g \circ f : C} \qquad \frac{}{A \vdash ! : 1}$$

$$\frac{A \vdash f : B \quad A \vdash g : C}{A \vdash \langle f, g \rangle : B \times C} \qquad \frac{}{A \times B \vdash \pi : A} \qquad \frac{}{A \times B \vdash \pi' : B}$$

$$\frac{A \times B \vdash f : C}{A \vdash \Lambda(f) : B \to C} \qquad \frac{}{(A \to B) \times A \vdash ev : B}$$

Categorical combinators (equations)

$$(f \circ g) \circ h = f \circ (g \circ h)$$
$$f \circ id = f$$
$$id \circ f = f$$

$$\pi \circ \langle f, g \rangle = f$$
$$\pi' \circ \langle f, g \rangle = g$$
$$\langle f, g \rangle \circ h = \langle f \circ h, g \circ h \rangle$$
$$\langle \pi, \pi' \rangle = id$$

$$ev \circ \langle \Lambda(f), g \rangle = f \circ \langle id, g \rangle$$
$$\Lambda(f) \circ g = \Lambda(f \circ \langle g \circ \pi, \pi' \rangle)$$
$$\Lambda(ev) = id$$

$$f =!$$

CCC's as models of λ-calculus

Our goal is now to interpret / compile (simply-typed) λ-calculus in CCC's / categorical combinators.

Actually, we shall:

- first compile λ-calculus into variable-free notation (**De Bruijn**)

- and then compile this algebraic syntax into categorical combinators.

De Bruijn notation

Syntax of terms:

$$M ::= n \mid MM \mid \lambda.M$$

Compiling λ-calculus: The translation is indexed by a list of variables (we write $|\vec{x}|$ for the length of $\vec{x}$)

$$\begin{array}{l} DB_{\vec{x_1},x,\vec{x_2}}(x) = |\vec{x_2}| \quad (x \notin \vec{x_2}) \\ DB_{\vec{x}}(M_1 M_2) = DB_{\vec{x}}(M_1) DB_{\vec{x}}(M_2) \\ DB_{\vec{x}}(\lambda y.M) = \lambda.DB_{\vec{x},y}(M) \end{array}$$

β-reduction is then trickier than in the λ-calculus (if one thinks that avoiding capture of variables is not tricky)

β-reduction in De Bruijn notation

$$(\lambda.M)N = M[0 \leftarrow N]$$

where substitution is defined by

$$m[n \leftarrow N] = \begin{cases} m & \text{if } m < n \\ \tau_0^n(N) & \text{if } m = n \\ m-1 & \text{if } m > n \end{cases}$$

$$(M_1 M_2)[n \leftarrow N] = (M_1[n \leftarrow N])(M_2[n \leftarrow N])$$

$$(\lambda.M)[n \leftarrow N] = \lambda.(M[n+1 \leftarrow N])$$

$$\tau_i^n(j) = \begin{cases} j & \text{if } j < i \\ j+n & \text{if } j \geq i \end{cases}$$

$$\tau_i^n(N_1 N_2) = (\tau_i^n(N_1))(\tau_i^n(N_2))$$

$$\tau_i^n(\lambda.N) = \lambda.(\tau_{i+1}^n(N))$$

From De Bruijn to categorical combinators

$$
\begin{array}{lcl}
[\![A_n, \ldots, A_0 \vdash i : A_i]\!] & = & (\ldots(1 \times A_n) \times \ldots) \times A_{\vdash} \pi' \circ \pi^i : A_i \\
[\![\Gamma \vdash MN : B]\!] & = & ev \circ < [\![\Gamma \vdash M : A \to B]\!], [\![\Gamma \vdash N : A]\!] > \\
[\![\Gamma \vdash \lambda.M : A \to B]\!] & = & \Lambda([\![\Gamma, A \vdash M : B]\!]) \\
 & & \\
[\![M[n \leftarrow N]]\!] & = & [\![M]\!] \circ P^n(\langle id, [\![N]\!] \rangle) \\
[\![\tau_i^n(N)]\!] & = & [\![N]\!] \circ P^i(\pi^n)
\end{array}
$$

where $P(f) = \langle f \circ \pi, \pi' \rangle$

The translation preserves the equations of De Bruijn's presentation and turns an infinite syntax (varying n) into a finite one.

A big step operational semantics

The idea is to formalise the set-theoretical functions underlying the combinators:

$$\frac{}{\langle id|s\rangle = s} \qquad \frac{\langle f|s\rangle = s_1 \;\; \langle g|s_1\rangle = s_2}{\langle g \circ f|s\rangle = s_2}$$

$$\frac{\langle f|s\rangle = s_1 \;\; \langle g|s\rangle = s_2}{\langle\langle f, g\rangle|s\rangle = (s_1, s_2)} \qquad \frac{}{\langle \pi|(s_1, s_2)\rangle = s_1} \qquad \frac{}{\langle \pi'|(s_1, s_2)\rangle = s_2}$$

$$\frac{}{\langle \Lambda(f)|s\rangle = \Lambda(f)s} \qquad \frac{\langle f|(s, t)\rangle = s_1}{\langle ev|(\Lambda(f)s, t)\rangle = s_1}$$

On the way, we have defined a syntax for new syntactical objects s, that we call *values*:

$$s ::= () \parallel \underline{n} \parallel (s, s) \parallel \Lambda(f)s$$

Compiling categorical combinators into machine code

Idea: implementing tail recursion with a stack.

We now view a term as a piece of code: composition becomes code concatenation, and the symbols of the pairing combinator become instructions `PUSH`, `SWAP` and `CONS`, respectively.

$$code(id) = \texttt{SKIP} \quad code(\pi) = \texttt{CAR} \quad code(\pi') = \texttt{CDR} \quad code(ev) = ev$$
$$code(g \circ f) = code(f); code(g) \quad code(\Lambda(f)) = \Lambda(code(f))$$
$$code(\langle f, g \rangle) = \texttt{PUSH}; code(f); \texttt{SWAP}; code(g); \texttt{CONS}$$

The Categorical Abstract Machine

$$
\begin{array}{lcl}
\langle \mathtt{SKIP}; C \mid s \mid S \rangle & \rightarrow & \langle C \mid s \mid S \rangle \\
\langle \mathtt{CAR}; C \mid (s_1, s_2 \mid S \rangle & \rightarrow & \langle C \mid s_1 \mid S \rangle \\
\langle \mathtt{CDR}; C \mid (s_1, s_2 \mid S \rangle & \rightarrow & \langle C \mid s_2 \mid S \rangle \\
\langle \mathtt{PUSH}; C \mid s \mid S \rangle & \rightarrow & \langle C \mid s \mid s \cdot S \rangle \\
\langle \mathtt{SWAP}; C \mid s_1 \mid s_2 \cdot S \rangle & \rightarrow & \langle C \mid s_2 \mid s_1 \cdot S \rangle \\
\langle \mathtt{CONS}; C \mid s_2 \mid s_1 \cdot S \rangle & \rightarrow & \langle C \mid (s_1, s_2) \mid S \rangle \\
\langle \Lambda(C); C' \mid s \mid S \rangle & \rightarrow & \langle C' \mid \Lambda(C)s \mid S \rangle \\
\langle ev; C' \mid (\Lambda(C)s, t) \mid S \rangle & \rightarrow & \langle C; C' \mid (s, t) \mid S \rangle
\end{array}
$$

Normalisation by evaluation

On the board!

But you can access unfinished notes here:

`www.pps.univ-paris-diderot.fr/~curien/NBE.pdf`

There is also a hidden pointer to the book

Domains and Lambda-calculi (Amadio and Curien 1998). Ask me...

Monoidal categories

A **monoidal category** is a category $\mathbf{C}$ equipped with a functor $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$, called the **tensor product**, a distinguished object 1, called the **tensor unit**, and natural isomorphisms, also called the canonical isomorphisms:

$$\alpha : A \otimes (B \otimes C) \to (A \otimes B) \otimes C \qquad \iota_l : 1 \otimes A \to A \qquad \iota_r : A \otimes 1 \to A$$

satisfying the following two so-called coherence equations:

$$\begin{array}{ll} (\alpha - \alpha) & \alpha \circ \alpha = (\alpha \otimes id) \circ \alpha \circ (id \otimes \alpha) \\ (\alpha - \iota) & (\iota_r \otimes id) \circ \alpha = id \otimes \iota_l \ . \end{array}$$

Free monoidal categories (the syntax)

Consider a set X. We build a monoidal category $\mathit{Free}(X)$ as follows. We have a syntax for objects and terms:

$$T ::= x \ \ (\text{where } x \in X) \parallel I \parallel T \otimes T$$
$$M ::= \alpha \parallel \lambda \parallel \rho \parallel \alpha^{-1} \parallel \lambda^{-1} \parallel \rho^{-1} \parallel M \circ M \parallel id \parallel M \otimes M$$

with typing rules:

$$\frac{}{\alpha : (T_1 \otimes T_2) \otimes T_3 \to T_1 \otimes (T_2 \otimes T_3)} \qquad \frac{}{\lambda : I \otimes T \to T} \qquad \frac{}{\rho : T \otimes I \to T}$$

$$\frac{}{\alpha^{-1} : T_1 \otimes (T_2 \otimes T_3) \to (T_1 \otimes T_2) \otimes T_3} \qquad \frac{}{\lambda^{-1} : T \to I \otimes T} \qquad \frac{}{\rho^{-1} : T \to T \otimes I}$$

$$\frac{}{id : T \to T} \qquad \frac{M_1 : T_1 \to T_2 \quad M_2 : T_2 \to T_3}{M_2 \circ M_1 : T_1 \to T_3} \qquad \frac{M_1 : T_1 \to T_1' \quad M_2 : T_2 \to T_2'}{M_1 \otimes M_2 : T_1 \otimes T_2 \to T_1' \otimes T_2'}$$

quotiented by the laws of categories, bifunctors, naturality + coherence equations

The coherence theorem

$Free(X)$ is indeed free: for every function $\rho : X \to \mathbf{C}$ (mapping each x to an object of a monoidal category $\mathbf{C}$) there exists a unique strict monoidal functor $[\![\_]\!]^{\rho}_{\mathbf{C}} : Free(X) \to \mathbf{C}$ that extends it.

Coherence theorem: for any two terms $M, M' : T \to T'$ of the **same type**, we have

$$[\![M]\!]^{\rho}_{\mathbf{C}} = [\![M']\!]^{\rho}_{\mathbf{C}}$$

(for any monoidal category, and any valuation).

Proof: By confluence and strong normalisation! (remark of **Huet**)

Other proof: by a 2-categorical version of the Yoneda embedding (**Yoneda strictifies**, see e.g. Leinster's book on higher operads).

Sequent calculus

Sequents are formal objects of the form $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$, with the understanding that the conjunction of the A's implies the disjunction of the B's.

In presence of an involutive negation ($(A^\perp)^\perp = A$), we can put all formulas on the right.

In sequent calculus, one has only introduction rules (no elimination rules like in natural deduction) for the connectives + the other rules (next slide)

Natural deduction corresponds closely to λ-calculus (Curry-Howard)

There exists also a syntax corresponding to sequent calculus (**Curien-Herbelin**)

The non-logical rules

CONTRACTION

$$\frac{\vdash \Gamma, A, A}{\vdash \Gamma, A}$$

WEAKENING

$$\frac{\vdash \Gamma}{\vdash \Gamma, A}$$

AXIOM

$$\frac{}{\vdash A, A^{\perp}}$$

CUT

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, A^{\perp}}{\vdash \Gamma, \Delta}$$

Linear logic (Girard)

The contraction rule in logic is responsible for the combinatorial explosion of cut-elimination / normalisation.

In contrast, linear λ-terms normalize in linear time.

The idea is to
1) remove contraction and weakening: then there are two disjunctions and two conjunctions
2) reintroduce contraction and weakening in a controlled way, through a modality

Multiplicatives and additives

MULTIPLICATIVES

$$\frac{\vdash A, B, \Gamma}{\vdash A \parr B, \Gamma} \qquad \frac{\vdash A, \Gamma_1 \quad \vdash B, \Gamma_2}{\vdash A \otimes B, \Gamma_1, \Gamma_2}$$

ADDITIVES

$$\frac{\vdash A, \Gamma}{\vdash A \oplus B, \Gamma} \quad \frac{\vdash B, \Gamma}{\vdash A \oplus B, \Gamma} \qquad \frac{\vdash A, \Gamma \quad \vdash B, \Gamma}{\vdash A \& B, \Gamma}$$

In the absence of contraction and weakening, the rules for the two disjunctions ($\parr, \oplus$) are not interderivable (same for the conjunctions $\&, \otimes$).

The units are the 0-ary cases of these 4 connectives (omitted).

Positive and negative connectives

Besides the dichotomy of additives versus multiplicatives (the terminology comes from the coherence space semantics), there is also another dichotomy:

POSITIVE	**NEGATIVE**
$\otimes$ $\oplus$	$\parr$ $\&$
Irreversible	Reversible
Eager	Lazy
Call-by-value	Call-by-name
Active	Passive
Player	Opponent

Formulas as resources

The following is known as Lafont's menu:

Menu (price 17 Euros)

Quiche or Salad
Chicken or Fish
Banana or "Surprise du Chef"*

(*) either "*Profiteroles*" or "*Tarte Tatin*"

$$17E \vdash \left\{ \begin{array}{c} (Q \& S) \\ \otimes \\ (C \& F) \\ \otimes \\ (B \& (P \oplus T)) \end{array} \right.$$

The game semantical reading of proofs

The player is the cook and plays a rule, the opponent is the client and challenges the player by picking one antecedent in the rule.

$$\dfrac{5E \vdash Q\&S \quad 8E \vdash C\&F \quad \dfrac{4E \vdash B \quad \dfrac{4E \vdash T}{4E \vdash P \oplus T}}{4E \vdash B\&(P \oplus T)}}{17E \vdash (Q\&S) \otimes (C\&F) \otimes (B\&(P \oplus T))}$$

The exponentials

We add two more connectives: ! and its dual ?

CONTRACTION

$$\frac{\vdash ?A, ?A, \Gamma}{\vdash ?A, \Gamma}$$

WEAKENING

$$\frac{\vdash \Gamma}{\vdash ?A, \Gamma}$$

DERELICTION

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}$$

PROMOTION

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}$$

The full syntax of formulas

$$A ::= X \mid X^{\perp} \mid A \otimes A \mid 1 \mid A \oplus B \mid 0 \mid A \& B \mid \top \mid A \parr B \mid \perp \mid !A \mid ?A$$

Negation defined by De Morgan's laws:

$$\begin{array}{ll} X^{\perp\perp} = X & (!A)^{\perp} = ?(A^{\perp}) \\ (A \otimes B)^{\perp} = A^{\perp} \parr B^{\perp} & 1^{\perp} = \perp \\ (A \oplus B)^{\perp} = A^{\perp} \& B^{\perp} & 0^{\perp} = \top \end{array}$$

Two important provable isomorphisms:

$$A \otimes (B \oplus C) \equiv (A \otimes B) \oplus (A \otimes C)$$

$$(!A) \otimes (!B) \equiv !(A \& B)$$

Two important derived connectives:

$$A \multimap B = A^{\perp} \parr B \qquad A \Rightarrow B = (!A) \multimap B$$

from where linear logic started.

Cut elimination

$$\dfrac{\dfrac{\Pi_1 \\ \vdots \\ \vdash A^\perp, B^\perp, \Gamma_1}{\vdash A^\perp \parr B^\perp, \Gamma_1} \qquad \dfrac{\Pi_2 \\ \vdots \\ \vdash A, \Gamma_2 \qquad \Pi_3 \\ \vdots \\ \vdash B, \Gamma_3}{\vdash A \otimes B, \Gamma_2, \Gamma_3}}{\vdash \Gamma_1, \Gamma_2, \Gamma_3} \quad \longrightarrow \quad \dfrac{\dfrac{\Pi_1 \\ \vdots \\ \vdash A^\perp, B^\perp, \Gamma_1 \qquad \Pi_2 \\ \vdots \\ \vdash A, \Gamma_2}{\vdash B^\perp, \Gamma_1, \Gamma_2} \qquad \Pi_3 \\ \vdots \\ \vdash B, \Gamma_3}{\vdash \Gamma_1, \Gamma_2, \Gamma_3}$$

$$\dfrac{\dfrac{\Pi \\ \vdots \\ \vdash \Gamma_1, ?B^\perp, ?B^\perp}{\vdash \Gamma_1, ?B^\perp} \qquad \dfrac{\Pi' \\ \vdots \\ \vdash ?\Gamma_2, B}{\vdash ?\Gamma_2, !B}}{\vdash \Gamma_1, ?\Gamma_2} \quad \longrightarrow \quad \dfrac{\dfrac{\Pi \\ \vdots \\ \vdash \Gamma_1, ?B^\perp, ?B^\perp \qquad \dfrac{\Pi' \\ \vdots \\ \vdash ?\Gamma_2, B}{\vdash ?\Gamma_2, !B}}{\vdash \Gamma_1, ?\Gamma_2, ?B^\perp} \qquad \dfrac{\Pi' \\ \vdots \\ \vdash ?\Gamma_2, B}{\vdash ?\Gamma_2, !B}}{\vdash \Gamma_1, ?\Gamma_2, ?\Gamma_2}$$

$$\vdash \Gamma_1, ?\Gamma_2$$

Categorical semantics of linear logic

We limit ourselves to the connectives $\otimes$, $\multimap$, !, and $\&$.
We need (simplified):
• a monoidal closed category $\mathbf{C}$ (i.e. all $\_ \otimes A$ have right adjoints) to interpret $\otimes$ and $\multimap$
• a comonad on $\mathbf{C}$ to interpret !.
• $\mathbf{C}$ must also have finite products to interpret $\&$, and the following compatibilty must hold:

$$(!A) \otimes (!B) \cong !(A \& B) \qquad 1 \cong !\top \, .$$

The key fact connecting intuitionnistic logic and linear logic lies in the following: from these assumptions, it follows that the co-Kleisli category $\mathbf{C}_!$ is cartesian closed.

More on linear logic

• Girard also introduced **proof nets**, a beautiful presentation of proofs as graphs (quotienting irrelevant details of sequent calculus proofs) for the multiplicative-exponential fragment. The case of units and additives is much trickier.

• restrictions of the capabilities of the exponentials to characterise time and space complexity classes (started by Girard under the name of light linear logics)

• differential linear logic (**Ehrhard-Regnier**)

• ludics (Girard)

A short journey in domain theory

All started with the solution of the equation $D \cong (D \to D)$.

For this, to avoid the cardinality problem, **Scott** proposed complete partial orders (cpo's) and **continuous** functions between them.

But then who can do more can do less: the same framework allowed to give meaning to functional programs involving recursive and possibly non-terminating computations.

In order to understand the relations between syntax and semantics, a small toy language called **PCF** was introduced (still by Scott) and two seminal papers by Milner and Plotkin investigated the relation between this syntax and its domain-theoretic interpretation.

Complete partial orders

Given a partial order $(D, \leq)$, a non-empty subset $\Delta \subseteq D$ is called *directed* if

$$\forall\, x, y \in \Delta\ \ \exists\, z \in \Delta\ \ x \leq z \text{ and } y \leq z\ .$$

A partial order $(D, \leq)$ is called a *directed complete partial order* (dcpo) if every directed subset has a least upper bound *(lub)*), denoted $\bigvee \Delta$. If moreover $(D, \leq)$ has a least element (written $\bot$), then it is called a *complete partial order* (cpo).

The CCC of cpo's and continuous functions

Let $(D, \leq)$ and $(D', \leq)$ be partial orders. A function $f : D \to D'$ is called monotonic if

$$\forall\, x, y \in D \;\; x \leq y \Rightarrow f(x) \leq f(y)\ .$$

If D and D' are dcpo's, a function $f : D \to D'$ is called continuous if it is monotonic and preserves directed lub's: $f(\bigvee \Delta) = \bigvee f(\Delta)$

• Cpo's and continuous functions form a cartesian closed category $\mathbf{Cpo}$.
• The partial order on $D \to E$ is the pointwise ordering:

$$f \leq g \quad \text{iff} \quad \forall\, x \;\; f(x) \leq g(x)$$

• Moreover, every continuous function $f : D \to D$ has a (least) fixpoint $(\bigvee f^n(\bot))$

The language PCF

It is simply typed λ-calculus with types

$$\sigma ::= \iota \mid o \mid \sigma \to \sigma$$

to which one adds the following constants:

n	$: \iota$	$(n \in \omega)$
tt, ff	$: o$	
$succ, pred$	$: \iota \to \iota$	
$zero?$	$: \iota \to o$	
if then else	$: o \to \iota \to \iota \to \iota$	
if then else	$: o \to o \to o \to o$	
Y	$: (\sigma \to \sigma) \to \sigma$	for all σ

Small-step operational semantics for PCF

$(\lambda x.M)N \to_{op} M[N/x]$ $\qquad$ $YM \to_{op} M(YM)$

$zero?(0) \to_{op} tt$ $\qquad$ $zero?(n+1) \to_{op} ff$

$succ(n) \to_{op} n+1$ $\qquad$ $pred(n+1) \to_{op} n$

if tt *then* N *else* $P \to_{op} N$ $\qquad$ *if* ff *then* N *else* $P \to_{op} P$

$$\frac{M \to_{op} M'}{MN \to_{op} M'N} \qquad \frac{M \to_{op} M'}{\textit{if } M \textit{ then } N \textit{ else } P \to_{op} \textit{if } M' \textit{ then } N \textit{ else } P}$$

$$\frac{M \to_{op} M'}{f(M) \to_{op} f(M')} \quad \text{(for } f \in \{succ, pred, zero?\}\text{)}$$

The continuous model of PCF

To interpret PCF in **Cpo** we just need to interpret

• the base types:

$$[\![o]\!] = \{\bot, tt, ff\}$$

with $\bot \leq tt$ and $\bot \leq ff$ (and the same for ι)

• and the constants: Y as the least fixed point operator, and the other constants in the obvious way.

These properties make sense in every cpo-enriched category, reformulating the condition on Y as

$$[\![Y]\!] = \bigvee [\![x : \sigma, f : \sigma \to \sigma \vdash f^n(x)]\!] \circ \langle id, \bot \circ !\rangle$$

When they are satisfied, the model is called **standard**.

Algebraic cpo's

The cpo's interpreting PCF types in the continuous model have the further proprety of being **algebraic**. This means that each element x is the directed least upper bound of its compact approximants $d \leq x$ (d is called **compact** if whenever $d \leq \bigvee \Delta$ then $d \leq x$ for some $x \in \Delta$).

Algebraicity gives us a logical reading of domain theory:

- $d \leq x$ as a predicate
- x itself as a set of predicates that hold for x

A turbo introduction to enriched categories

Let $\mathbf{V}$ be a monoidal category (where composition is denoted with $\cdot$, and identies with 1). A **$\mathbf{V}$-enriched category** $\mathbf{C}$ is a structure consisting of

- a collection of objects $A : \mathbf{C}$
- a family of objects $Hom[A, B]$ of $\mathbf{V}$ (for each pair of objects),

together with

- families of morphisms

$$id : I \to Hom[A, A] \quad \text{and} \quad \circ : Hom[B, C] \otimes Hom[B, C] \to Hom[A, C]$$

such that

$$\circ \cdot (id \otimes \circ) = \circ \cdot (\circ \otimes id) \cdot \alpha \quad \text{(pentagon strikes again!)}$$
$$\circ \cdot (id \otimes 1) = \lambda$$
$$\circ \cdot (1 \otimes id) = \rho$$

Adequacy

The following property holds in the continuous model (and in fact in any standard model of PCF) (**computational adequacy**):

$$M \to^* v \text{ if and only if } [\![M]\!] = v$$

for every closed term of base type. The proof uses realisability.

• We next define **operational equivalence**:

$$M =_{op} N \quad \Leftrightarrow \quad (\forall C \ C[M] \to^* v \ \Leftrightarrow \ C[N] \to^* v) \ ,$$

where C ranges over contexts (terms with a hole) of base type.

• Computational adequacy easily implies the following property, called **adequacy**:

$$[\![M]\!] = [\![N]\!] \quad \Rightarrow \quad M =_{op} N \ .$$

If the converse also holds, then we say that the model is **fully abstract**.

Extensional models of PCF

• A category $\mathbf{C}$ with terminal object 1 is said to have **enough points** if (for any A, B and any $f, g \in \mathbf{C}[A, B]$):

$$\forall\, h : 1 \to a \;\; (f \circ h = g \circ h) \;\; \Rightarrow \;\; f = g \;\; .$$

A model of PCF that has enough points is called **extensional**

• A standard model of PCF is called **order-extensional** if the order between the morphisms is the pointwise ordering.

The full abstraction problem for PCF

Milner has shown the following results:

• In an extensional standard fully abstract model of PCF, the interpretations of all types are algebraic, and their compact elements must be definable, i.e., the meaning of some closed term

• Easy consequence: all extensional standard fully abstract models of PCF are isomorphic

• There exists an order-extensional standard fully abstract model of PCF, built (roughly) by quotienting the syntax by the observational equivalence.

So what was the open problem? To find another description of this unique fully abstrat model (it was a vaguely stated problem, but triggered lots of works!)

First candidate: the continuous model

It is not fully abstract, because por is not definable, where $por : [\![o]\!] \times [\![o]\!] \to [\![o]\!]$ is a function such that

$$por(tt, \bot) = tt \qquad por(\bot, tt) = tt$$

This is because the meaning of all closed terms of type $o \times o \to o$ are stable (next slide), while por is not.

But **Plotkin** showed that the continuous model is fully abstract for PCF extended with a constant por with this behaviour.

Second candidate: the stable model (Berry)

A continuous function f between two cpo's is called stable if whenever x and y are upward compatible (i.e. $\exists z \;\; x \leq z, y \leq z$) we have

$$f(x \wedge y) = f(x) \wedge f(y)$$

(one requires our cpo's to have all such greatest lower bounds)

Berry has constructed cartesian closed categories having as objects a certain class of cpo's and as morphisms the stable functions between them.

This model is not fully abstract because (omitted):
- there are stable functions that are **not** sequential
- meaning of all closed terms of type $\kappa_1 \to \ldots \to \kappa_n \to \kappa$ (κ, κ_i base types) are sequential

What next will fail?

Let's go back to Assia's lectures last week. She showed us that the two programs for computing addition behaved quite differently:

$$addl(0, y) = y \qquad addr(x, 0) = x$$
$$addl(Sx, y) = S(addl(x, y)) \qquad addr(x, Sy) = S(addr(x, y))$$

PhD subject proposed by Berry back in 1977:

find a (non-extensional) model of PCF where $[\![addl]\!] \neq [\![addr]\!]$

Sequential algorithms

Some student gladly worked on the subject... This gave rise to the model of **sequential algorithms** (1978). It is a funny model in which not only the left addition and the right addition are different, but there is an element in the model that separates them:

$$add\_taster([\![addl]\!]) = tt \qquad add\_taster([\![addr]\!]) = ff$$

One cannot implement this in PCF. We need some effects (to be able to observe the order of execution and to raise exceptions)

Therefore (the extensional collapse of) the model of sequential algorithms is not fully abstract

But it is fully abstract for a certain extension of PCF with control (1992, **Cartwright-Curien-Felleisen**)

Has the full abstraction problem for PCF been solved?

• In 1994, two groups of researchers (**Abramsky-Jagadeesan-Malacaria**, and **Hyland-Ong + Nickau**) announced their "solution". They built game models (similar in spirit to our 1978 model) that matched the syntax of PCF so closely that not only all elements are definable, but definable by a unique term in (possibly infinite) normal form. The model itself is not fully abstract, but its quotient by the observational order is. But that is (essentially) what Milner had done...

• A bit later, (**Loader**, published in 2001) proved the undecidability of the observational equivalence for finitary PCF.

• My conclusion has been and remains that in a sense the full abstraction problem has been solved, **by the negative**: a denotational model should be there to help us decide!

• But game semantics is great! It was just the advertising that was a bit... flashy. These things happen!

In the meantime...

In 1983 **Girard** reinvinted stability in his paper "System F, fifteen years later", in a very nice setting.

His cpo's are of the form $D(E)$ where E is a set of tokens, or of elementary data pieces, that you can assemble if they are coherent. The category of coherence spaces and stable functions has

• as objects pairs $(E, \frown)$ where E is a set and $\frown$ is a reflexive relation (a graph!)

• as morphisms from $(E, \frown)$ to $(E', \frown')$ the stable functions from $D(E)$ to $D(E')$

where $D(E)$ is the set ol cliques x of E $(e_1, e_2 \in x \Rightarrow e_1 \frown e_2)$.

And this gave birth to...

The function space $(E', \frown')^{(E,coh)}$ has as tokens pairs

$$(x, e')$$

where x is a finite clique of $(E, \frown)$ and e' is a token of $(E', \frown')$.

This led Girard to define

• $!(E, \frown)$ as the coherence space whose tokens are the finite cliques of E and where $x \frown y$ holds iff $x \cup y$ is a clique.

• $(E, \frown) \multimap (E', \frown')$ as a coherence space whose tokens are pairs

$$(e, e')$$

where e is a token of $(E, \frown)$ and e' is a token of $(E', \frown')$.

A new logic was born!

Successful stories...

It turned out that the decomposition $E^D = (!D) \multimap E$ works in all models considered so far in the slides, and also to the next ones to come in the slides.

Girard considered also a variant of $!(E, \frown)$ where the tokens are not finite cliques, but finite multisets of tokens whose underlying set is a clique. Note that this set of tokens is no longer finite when E is finite!

These two variants of ! are known as the set and the multiset variant.

Sequential algorithms and Hyland-Ong games essentially differ by their ! which is "set" in the first case, and "multiset" in the second case.

Other successful stories...

Game semantics in the Hyland-Ong style have been very successful for classifying programming features by properties of the morphisms, also called strategies, of various categories of games.

As far as full abstraction is concerned, Hyland and Ong style has led to a nice fully-abstract model of ALGOL (say, PCF with references).