

Sequential products in effect categories

Jean-Guillaume Dumas, Dominique Duval, Jean-Claude Reynaud
work in progress

Journées ARROWS
Nancy — June 7., 2007

Outline

Introduction

Examples

Cartesian categories

Cartesian effect categories

Conclusion

The problem

In some languages, like C,
the order of evaluation of function arguments is unspecified.

- ▶ when there is no computational effect,
the order of evaluation does not matter
- ▶ when effects do occur,
the order of evaluation becomes fundamental
e.g. `a[i]=++i;`

The problem is to design a formal framework
for **imposing an evaluation order**

Some solutions

The language `Haskell` provides a framework for dealing with computational effects:

- ▶ **Monads** [Moggi 91, Wadler 93]

with generalizations:

- ▶ **Freyd categories** [Power-Robinson 97]
- ▶ **Arrows** [Hughes 00]

Comparisons [Heunen-Jacobs 06]: “all are monoids”:

Monads “are” Arrows “are” Freyd categories

Sequentialization

- ▶ without effects, the function:

$$(1) \quad (a_1, a_2) \mapsto (f_1(a_1), f_2(a_2))$$

can be decomposed as:

$$(2) \quad (a_1, a_2) \mapsto (f_1(a_1), a_2) \mapsto (f_1(a_1), f_2(a_2))$$

- ▶ with effects, (1) is ambiguous, but (2) is not:
“compute first $f_1(a_1)$, then $f_2(a_2)$ ”

So, the issue is about:

$$(a_1, a_2) \mapsto (f(a_1), a_2)$$

“compute $f(a_1)$ and keep the information about a_2 ”

- ▶ **strength** for Monads
- ▶ **premonoidal category** for Freyd categories
- ▶ **first** operator for Arrows

Our solution

Like the other frameworks,
we distinguish two kinds of **functions**:

- ▶ (general) **functions** $\boxed{\rightarrow}$: maybe with effect
- ▶ **pure functions** $\boxed{\rightsquigarrow}$: effect-free

pure functions are functions

cf. [Moggi 91]: values are computations

Unlike the other frameworks,
we distinguish two kinds of **equations**:

- ▶ (strong) **equations** $\boxed{\equiv}$: for equalities
- ▶ **semi-equations** $\boxed{\lesssim}$: some kind of “local comparability”

strong equations are semi-equations

Outline

Introduction

Examples

Cartesian categories

Cartesian effect categories

Conclusion

Two examples

► Partiality

- can be handled with the monad $X \mapsto X + 1$
- our semi-equations form an partial order relation

► State

- can be handled with the monad $X \mapsto (S \times X)^S$
- our semi-equations form an equivalence relation

Partiality

Two kinds of **functions**:

- ▶ general functions may be partial
- ▶ pure functions are total functions

Two kinds of **equations**:

- ▶ an equation $f \equiv g$ is an equality (of domains and values)
- ▶ a semi-equation $f \lesssim g$ is a (usual) inequality:
 $\mathcal{D}(f) \subseteq \mathcal{D}(g)$ and $f(x) = g(x)$ for all $x \in \mathcal{D}(f)$.

Key property:

$$\begin{array}{ccc} x_1 & \xrightarrow{f} & f(x_1) \text{ or } \perp \\ \uparrow \text{ } \uparrow & \equiv & \uparrow \text{ } \uparrow \\ \overline{(x_1, x_2)} & \longrightarrow & \overline{(f(x_1), x_2) \text{ or } \perp} \\ \downarrow \text{ } \downarrow & \gtrsim & \downarrow \text{ } \downarrow \\ x_2 & \xrightarrow{\text{id}} & x_2 \text{ or } \perp \end{array}$$

State

Two kinds of **functions**:

- ▶ general functions may use and modify the state
- ▶ pure functions neither use nor modify the state

Two kinds of **equations**:

- ▶ an equation $f \equiv g$ is an equality
- ▶ a semi-equation $f \lesssim g$ (or $f \cong g$)

only means that the resulting values are equal:

$f(s, x) = (s', y), g(s, x) = (s'', y)$ with the same y .

Key property:

$$\begin{array}{ccc} (s, x_1) & \xrightarrow{f} & f(s, x_1) = (s', y_1) \\ \uparrow \text{~~~~~} & \equiv & \uparrow \text{~~~~~} \\ (s, x_1, x_2) & \xrightarrow{\quad} & (s', y_1, x_2) \\ \downarrow \text{~~~~~} & \lesssim & \downarrow \text{~~~~~} \\ (s, x_2) & \xrightarrow{\text{id}} & (s, x_2) (\neq) (s', x_2) \end{array}$$

Outline

Introduction

Examples

Cartesian categories

Cartesian effect categories

Conclusion

Multivariate functions: $f(x_1, \dots, x_n)$

- ▶ “Logical” view:
several arguments: x_1, \dots, x_n
- ▶ “Categorical” view:
one argument: $\langle x_1, \dots, x_n \rangle$

$$f(x_1, \dots, x_n) = f(\langle x_1, \dots, x_n \rangle)$$

Substitution is split in two parts:

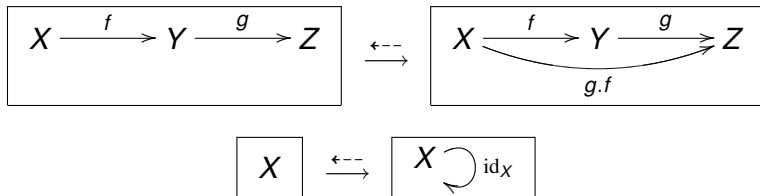
1. formation of the tuple $t = \langle t_1, \dots, t_n \rangle$
2. substitution of one argument $f(t)$

Categories

Categories =
the framework for substituting **one** argument
 $f(t) = f.t$

Definition

A **category** is a graph with composition:



generalizing monoids: $h.(g.f) \equiv (h.g).f$, $f.\text{id} \equiv f$, $\text{id}.f \equiv f$.

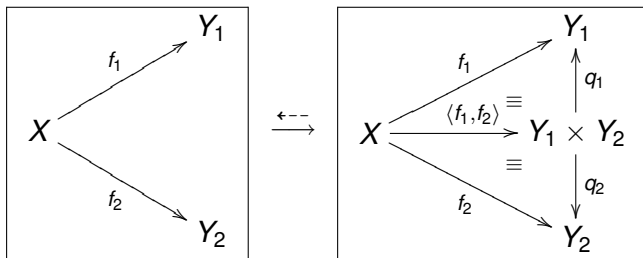
Words

drawings	graphs	categories	computer sc.
point	vertex	object	type
arrow	edge	morphism	function

All functions are **univariate**!

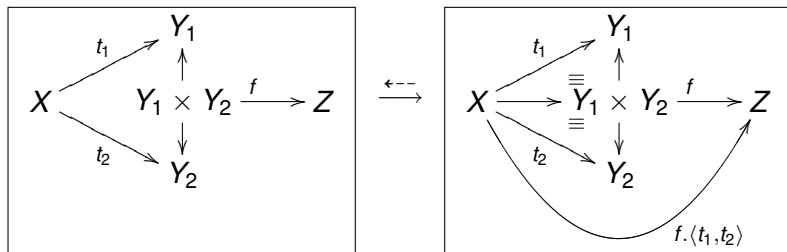
(Categorical) Products

An abstraction of the cartesian product of sets (here, $n = 2$)



Multivariate functions

1. formation of the **tuple** $t = \langle t_1, \dots, t_n \rangle$
2. substitution of **one** argument $f(t)$



$$f(t_1, \dots, t_n) = f.\langle t_1, \dots, t_n \rangle$$

Cartesian categories

Cartesian categories =
the framework for substituting **several** arguments

$$f(t_1, \dots, t_n) = f.\langle t_1, \dots, t_n \rangle$$

Definition

A **cartesian category** is a category with products.

Outline

Introduction

Examples

Cartesian categories

Cartesian effect categories

Conclusion

Effect categories (1/2)

Definition

An **effect category** is a **decorated** category:

- ▶ two kinds of functions:
 - (general) **functions** \rightarrow
 - **pure functions** \rightsquigarrow
every pure function is a function
identities are pure, composition of pures is pure
- ▶ two kinds of equations:
 - (strong) **equations** \equiv
 - **semi-equations** \lesssim
every equation is a semi-equation
on pure functions, \lesssim and \equiv coincide

and...

Effect categories (2/2)

... and in addition:

- ▶ \lesssim satisfies **substitution**:
if $g_1 \lesssim g_2 : Y \rightarrow Z$ then $g_1.f \lesssim g_2.f$
- ▶ \lesssim satisfies **replacement only** for **pure functions**:
if $g_1 \lesssim g_2 : Y \rightarrow Z$ and v **pure** then $v.g_1 \lesssim v.g_2$

Examples

► partiality

\mathbf{C} is the category of partial functions

pure functions are total functions

$f \equiv g$ means $f = g$ (equality of domains and values)

$f \lesssim g$ means $\mathcal{D}(f) \subseteq \mathcal{D}(g)$ and $f(x) = g(x)$ for all $x \in \mathcal{D}(f)$.

► state

S is the set of states

\mathbf{C} is the category with points $S \times X$

and with all functions

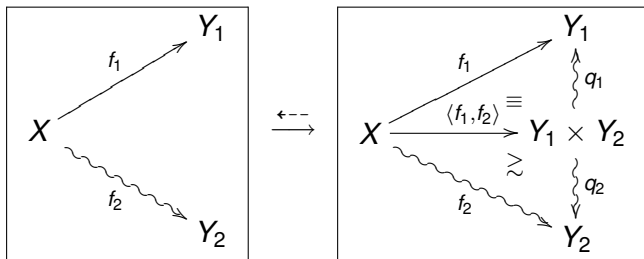
pure functions are $\text{id}_S \times v: (s, x) \mapsto (s, v(x))$

$f \equiv g$ means $f = g$

$f \lesssim g$ means f and g return the same value $y \in Y$,
maybe **not** the same state!

Semi-products

A semi-product is a **decorated** product
it defines $\langle f_1, f_2 \rangle$ **only** when f_2 is **pure**



Identities are pure! Hence, we get:

$$\langle f, \text{id} \rangle : (a_1, a_2) \mapsto (f(a_1), a_2)$$

“compute $f(a_1)$ and keep the information about a_2 ”

Examples

- ▶ **partiality**

$$\begin{aligned}\langle f_1, f_2 \rangle(x_1, x_2) &= (f(x_1), x_2) \text{ when } x_1 \in \mathcal{D}(f) \\ &= \perp \text{ when } x_1 \notin \mathcal{D}(f)\end{aligned}$$

- ▶ **state**

$$\begin{aligned}\langle f_1, f_2 \rangle(s, x_1, x_2) &= (s', y_1, y_2) \\ \text{where } f_1(s, x_1) &= (s', y_1) \text{ and } f_2(s, x_2) = (s, x_2)\end{aligned}$$

Sequential product

“compute first $f_1(a_1)$, then $f_2(a_2)$ ”

Definition

$$f_1 \times f_2 = (\text{id}_{Y_1} \times f_2) \cdot (f_1 \times \text{id}_{X_2})$$

$$\begin{array}{ccccc}
 X_1 & \xrightarrow{f_1} & Y_1 & \overset{\sim}{\xrightarrow{\text{id}}} & Y_1 \\
 \uparrow p_1 & & \uparrow s_1 & & \uparrow q_1 \\
 & \equiv & & \gtrsim & \\
 X_1 \times X_2 & \xrightarrow{f_1 \times \text{id}} & Y_1 \times X_2 & \xrightarrow{\text{id} \times f_2} & Y_1 \times Y_2 \\
 \uparrow p_2 & & \uparrow s_2 & & \uparrow q_2 \\
 & \gtrsim & & \equiv & \\
 X_2 & \overset{\sim}{\xrightarrow{\text{id}}} & X_2 & \xrightarrow{f_2} & Y_2
 \end{array}$$

A sequential product is a “weak product”

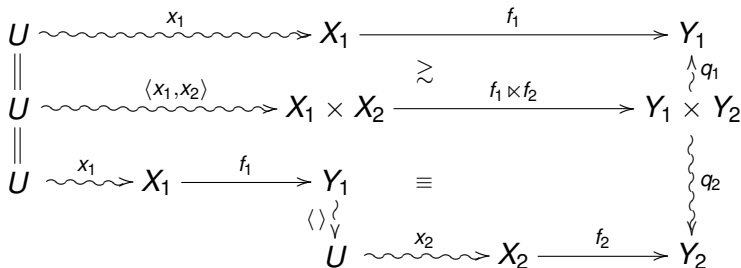
Theorem

For each $f_1 : X_1 \rightarrow Y_1$, $f_2 : X_2 \rightarrow Y_2$

and pure values $x_1 : U \rightsquigarrow X_1$ and $x_2 : U \rightsquigarrow X_2$:

$$q_1.(f_1 \times f_2).\langle x_1, x_2 \rangle \lesssim f_1.x_1$$

$$q_2.(f_1 \times f_2).\langle x_1, x_2 \rangle \equiv f_2.x_2.\langle \rangle.f_1.x_1$$



Decorated results and proofs

By forgetting the decorations:

- ▶ every decorated result remains a result
- ▶ every decorated proof remains a proof

By adding decorations:

- ▶ some results can be decorated, maybe in several ways,
- ▶ and for these results, some proofs can be decorated

Cartesian effect categories vs. Arrows

Arrows generalize Monads: [Hugues 00] for Haskell

Theorem

Every cartesian effect category determines an Arrow

Arrows	Effect category
$\mathbf{A} \ X \ Y$ $\text{arr} :: (X \rightarrow Y) \rightarrow \mathbf{A} \ X \ Y$ $(\gg\gg) :: \mathbf{A} \ X \ Y \rightarrow \mathbf{A} \ Y \ Z \rightarrow \mathbf{A} \ X \ Z$ $\text{first} :: \mathbf{A} \ X \ Y \rightarrow \mathbf{A} \ (X, Z) \ (Y, Z)$	$\mathbf{C}(X, Y)$ $\mathbf{P}(X, Y) \subseteq \mathbf{C}(X, Y)$ $g.f$ $f \times \text{id}$

Outline

Introduction

Examples

Cartesian categories

Cartesian effect categories

Conclusion

Conclusion

- ▶ a new categorical framework for imposing an order of evaluation
- ▶ another application of **decorated categories** cf. exceptions [Duval-Reynaud 05] (decorated doctrines? [Lawvere])
- ▶ with one more level of abstraction: decorations are obtained from **morphisms between logics**, in the context of **diagrammatic logics** [Duval-Lair 02]

Références

► around Haskell

- [Moggi 91] Notions of Computation and Monads, *Information and Computation* 93, p.55–92.
- [Wadler 93] Monads for functional programming, *Program Design Calculi* Springer-Verlag.
- [Power Robinson 97] Premonoidal Categories and Notions of Computation, *Mathematical Structures in Computer Science* 7, p.453–468.
- [Hughes 00] Generalising monads to arrows, *Science of Computer Programming* 37, p.67–111.
- [Heunen Jacobs 06] Arrows, like Monads, are Monoids, *Electronic Notes in Theoretical Computer Science* p.219–236.

► decorated logic

- [Duval Lair 03] Diagrammatic Specifications, *Mathematical Structures in Computer Science* 13, p.857–890.
- [Duval Reynaud 05] Dynamic logic and exceptions: an introduction, *MAP'05, Dagstuhl Seminars*