Sequential products in effect categories

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

Journées ARROWS
Nancy — June 7., 2007

Outline

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:

$$\boxed{(a_1, a_2) \mapsto (f(a_1), a_2)}$$

"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 $\boxed{\to}$: 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

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:

$$\begin{tikzcd}
x_1 \arrow[r, mapsto, "f"] \arrow[d, phantom, ""] & f(x_1) \text{ or } \perp \\
(x_1, x_2) \arrow[u, rightsquigarrow, mapsto] \arrow[d, rightsquigarrow, mapsto] \arrow[r, mapsto] \arrow[ur, phantom, "\equiv"] \arrow[dr, phantom, "\gtrsim"] & (f(x_1), x_2) \text{ or } \perp \arrow[u, rightsquigarrow, mapsto] \arrow[d, rightsquigarrow, mapsto] \\
x_2 \arrow[r, rightsquigarrow, mapsto, "\mathrm{id}"] & x_2 \text{ or } \perp
\end{tikzcd}$$

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:

$$\begin{tikzcd}
(s, x_1) \arrow[r, mapsto, "f"] \arrow[dr, phantom, "\equiv"] & f(s, x_1) = (s', y_1) \\
(s, x_1, x_2) \arrow[u, mapsto, squiggly] \arrow[r, mapsto] \arrow[d, mapsto, squiggly] \arrow[dr, phantom, "\gtrsim"] & (s', y_1, x_2) \arrow[u, mapsto, squiggly] \arrow[d, mapsto, squiggly] \\
(s, x_2) \arrow[r, mapsto, squiggly, "\mathrm{id}"] & (s, x_2)(\neq)(s', x_2)
\end{tikzcd}$$

Outline

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

- "Logical" view:
  several arguments: $x_1, \ldots, x_n$
- "Categorical" view:
  one argument: $\langle x_1, \ldots, x_n \rangle$

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

Substitution is split in two parts:
1. formation of the tuple $t = \langle t_1, \ldots, t_n \rangle$
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:

$$\boxed{\begin{tikzcd} X \arrow[r, "f"] & Y \arrow[r, "g"] & Z \end{tikzcd}} \quad \begin{smallmatrix} \dashleftarrow \\ \longrightarrow \end{smallmatrix} \quad \boxed{\begin{tikzcd} X \arrow[r, "f"] \arrow[rr, bend right, "g.f"'] & Y \arrow[r, "g"] & Z \end{tikzcd}}$$

$$\boxed{X} \quad \begin{smallmatrix} \dashleftarrow \\ \longrightarrow \end{smallmatrix} \quad \boxed{\begin{tikzcd} X \arrow[loop right, "\mathrm{id}_X"] \end{tikzcd}}$$

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

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, $n = 2$)

$$
\boxed{
\begin{tikzcd}
 & Y_1 \\
X \arrow[ur, "f_1"] \arrow[dr, "f_2"'] & \\
 & Y_2
\end{tikzcd}
}
\quad
\underset{\longrightarrow}{\dashleftarrow}
\quad
\boxed{
\begin{tikzcd}
 & & Y_1 \\
X \arrow[urr, "f_1"] \arrow[rr, "\langle f_1, f_2 \rangle"] \arrow[drr, "f_2"'] & & Y_1 \times Y_2 \arrow[u, "q_1"'] \arrow[d, "q_2"] \arrow[ul, phantom, "\equiv"] \arrow[dl, phantom, "\equiv"] \\
 & & Y_2
\end{tikzcd}
}
$$

Multivariate functions

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

$$\begin{tikzcd}
 & Y_1 & \\
X \arrow[ur, "t_1"] \arrow[dr, "t_2"'] & Y_1 \times Y_2 \arrow[u] \arrow[d] \arrow[r, "f"] & Z \\
 & Y_2 &
\end{tikzcd}
\quad
\begin{array}{c}\dashleftarrow \\ \longrightarrow\end{array}
\quad
\begin{tikzcd}
 & Y_1 & \\
X \arrow[ur, "t_1"] \arrow[r] \arrow[dr, "t_2"'] \arrow[rr, bend right=60, "f.\langle t_1, t_2 \rangle"'] & Y_1 \times Y_2 \arrow[u] \arrow[d] \arrow[r, "f"] & Z \\
 & Y_2 &
\end{tikzcd}$$

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

Cartesian categories

Cartesian categories =
the framework for substituting several arguments
$$f(t_1, \ldots, t_n) = f.\langle t_1, \ldots, t_n \rangle$$

Definition

A cartesian category is a category with products.

Outline

Effect categories (1/2)

Definition

An effect category is a decorated category:

- two kinds of functions:
  – (general) functions $\to$
  – 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 $g_1 \lesssim g_2 : Y \to Z$ then $g_1 . f \lesssim g_2 . f$
- $\lesssim$ satisfies replacement **only** for pure functions:
  if $g_1 \lesssim g_2 : Y \to Z$ and v **pure** then $v . g_1 \lesssim v . g_2$

Examples

- partiality
  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
  C is the category with points $S \times X$
  and with all functions
  pure functions are $\mathrm{id}_S \times v \colon (s, x) \mapsto (s, v(x))$
  $f \equiv g$ 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

$$
\boxed{\begin{tikzcd}
 & Y_1 \\
X \arrow[ur, "f_1"] \arrow[dr, rightsquigarrow, "f_2"'] & \\
 & Y_2
\end{tikzcd}}
\quad \overset{\dashleftarrow}{\longrightarrow} \quad
\boxed{\begin{tikzcd}
 & Y_1 \\
X \arrow[ur, "f_1"] \arrow[r, "\langle f_1, f_2 \rangle"] \arrow[dr, rightsquigarrow, "f_2"'] \arrow[ur, phantom, "\equiv", near end, shift right=3] \arrow[dr, phantom, "\gtrsim", near end, shift left=3] & Y_1 \times Y_2 \arrow[u, rightsquigarrow, "q_1"'] \arrow[d, rightsquigarrow, "q_2"] \\
 & Y_2
\end{tikzcd}}
$$

Identities are pure! Hence, we get:

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

"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
  $\langle f_1, f_2\rangle(s, x_1, x_2) = (s', y_1, y_2)$
  where $f_1(s, x_1) = (s', y_1)$ and $f_2(s, x_2) = (s, x_2)$

Sequential product

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

Definition

$$f_1 \ltimes f_2 = (\mathrm{id}_{Y_1} \times f_2).(f_1 \times \mathrm{id}_{X_2})$$

$$\begin{tikzcd}
X_1 \arrow[r, "f_1"] & Y_1 \arrow[r, squiggly, "\mathrm{id}"] & Y_1 \\
X_1 \times X_2 \arrow[u, squiggly, "p_1"] \arrow[d, squiggly, "p_2"'] \arrow[r, "f_1 \times \mathrm{id}"] \arrow[ur, phantom, "\equiv"] \arrow[dr, phantom, "\gtrsim"] & Y_1 \times X_2 \arrow[u, squiggly, "s_1"] \arrow[d, squiggly, "s_2"'] \arrow[r, "\mathrm{id} \times f_2"] \arrow[ur, phantom, "\gtrsim"] \arrow[dr, phantom, "\equiv"] & Y_1 \times Y_2 \arrow[u, squiggly, "q_1"] \arrow[d, squiggly, "q_2"'] \\
X_2 \arrow[r, squiggly, "\mathrm{id}"] & X_2 \arrow[r, "f_2"] & Y_2
\end{tikzcd}$$

A sequential product is a “weak product”

Theorem

For each $f_1 : X_1 \to Y_1$, $f_2 : X_2 \to Y_2$
and pure values $x_1 : U \rightsquigarrow X_1$ and $x_2 : U \rightsquigarrow X_2$:

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

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

$$\begin{tikzcd}
U \arrow[rr, rightsquigarrow, "x_1"] \arrow[d, equal] & & X_1 \arrow[rr, "f_1"] \arrow[drr, phantom, "\gtrsim"] & & Y_1 \\
U \arrow[rr, rightsquigarrow, "\langle x_1, x_2\rangle"] \arrow[d, equal] & & X_1 \times X_2 \arrow[rr, "f_1 \ltimes f_2"] & & Y_1 \times Y_2 \arrow[u, rightsquigarrow, "q_1"'] \arrow[dd, rightsquigarrow, "q_2"] \\
U \arrow[r, rightsquigarrow, "x_1"] & X_1 \arrow[r, "f_1"] & Y_1 \arrow[d, rightsquigarrow, "\langle\,\rangle"'] & \equiv & \\
 & & U \arrow[r, rightsquigarrow, "x_2"] & X_2 \arrow[r, "f_2"] & Y_2
\end{tikzcd}$$

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
$\mathtt{A}\ X\ Y$	$\mathbf{C}(X, Y)$
$\mathtt{arr} :: (X \to Y) \to \mathtt{A}\ X\ Y$	$\mathbf{P}(X, Y) \subseteq \mathbf{C}(X, Y)$
$(\ggg) :: \mathtt{A}\ X\ Y \to \mathtt{A}\ Y\ Z \to \mathtt{A}\ X\ Z$	$g.f$
$\mathtt{first} :: \mathtt{A}\ X\ Y \to \mathtt{A}\ (X, Z)\ (Y, Z)$	$f \times \mathrm{id}$

Outline

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*