Comonadic notions of computation

Tarmo Uustalu[1] Varmo Vene[2]

[1]Institute of Cybernetics, Tallinn

[2]University of Tartu

2nd Lübeck-Tartu Workshop
Tartu, 26 Sept 2006

Motivation

- Moggi and Wadler showed that effectful computations can be structured with monads.
- An effect-producing function from A to B is a map $A \to B$ in the Kleisli category, i.e., a map $A \to TB$ in the base category.
- Some examples applied in semantics:

$$
\begin{array}{lllll}
TA & = & A + 1 & & \text{partiality} \\
TA & = & A + E & & \text{exceptions} \\
TA & = & A^E & & \text{environment} \\
TA & = & A^* & = \mu X.1 + A \times X & \text{non-determinism} \\
TA & = & (A \times S)^S & & \text{state}
\end{array}
$$

- Are all impure features captured by monads?
- What about comonads?

Comonads

Definition

A comonad on category $\mathcal{C}$ is given by

- a functor $D : \mathcal{C} \longrightarrow \mathcal{C}$
- a natural transformation $\varepsilon_A : DA \longrightarrow A$
  - counit of the comonad
- a natural transformation $\delta_A : DA \longrightarrow D^2A$
  - comultiplication of the comonad

s.t. following diagrams commute

$$\begin{tikzcd} & DA \arrow[dl, "\delta_A"'] \arrow[d, equal] \arrow[dr, "\delta_A"] & \\ D^2A \arrow[r, "D\varepsilon_A"'] & DA & D^2A \arrow[l, "\varepsilon_{DA}"] \end{tikzcd} \qquad \begin{tikzcd} DA \arrow[r, "\delta_A"] \arrow[d, "\delta_A"'] & D^2A \arrow[d, "D\delta_A"] \\ D^2A \arrow[r, "\delta_{DA}"'] & D^3A \end{tikzcd}$$

Comonads

Comonads model notions of value in a context;

- DA is the type of contextually situated values of A.

A context-relying function from A to B
is a map $A \to B$ in the coKleisli category,

- i.e., a map $DA \to B$ in the base category.

Product (environment) comonad

- Functor: $DA = A \times E$
- Counit:

$$\begin{array}{rcl} \varepsilon_A & : & A \times E \to A \\ (a, e) & \mapsto & a \end{array}$$

- Comultiplication:

$$\begin{array}{rcl} \delta_A & : & A \times E \to (A \times E) \times E \\ (a, e) & \mapsto & ((a, e), e) \end{array}$$

Comonads

Streams comonad

- Functor: $DA = A^{\mathbb{N}} = \nu X.A \times X$
- Counit:

$$\begin{array}{rcl} \varepsilon_A & : & A^{\mathbb{N}} \rightarrow A \\ \alpha & \mapsto & \alpha(0) \end{array}$$

- Comultiplication:

$$\begin{array}{rcl} \delta_A & : & A^{\mathbb{N}} \rightarrow (A^{\mathbb{N}})^{\mathbb{N}} \\ \alpha & \mapsto & \lambda n.(\lambda m.\alpha(n+m)) \\ [a_0, a_1, a_2, \ldots] & \mapsto & [[a_0, a_1, a_2, \ldots], [a_1, a_2, a_3 \ldots], \ldots] \end{array}$$

Comonads for stream functions

Dataflow computation = discrete-time signal transformations = stream functions.

Example: simple dataflow programs

$$
\begin{aligned}
\mathit{pos} &= 0 \text{ fby } (\mathit{pos} + 1) \\
\mathit{sum}\ x &= x + (0 \text{ fby } (\mathit{sum}\ x)) \\
\mathit{fact} &= 1 \text{ fby } (\mathit{fact} * (\mathit{pos} + 1)) \\
\mathit{fibo} &= 0 \text{ fby } (\mathit{fibo} + (1 \text{ fby } \mathit{fibo}))
\end{aligned}
$$

`pos`	0	1	2	3	4	5	6	...
`sum pos`	0	1	3	6	10	15	21	...
`fact`	1	1	2	6	24	120	720	...
`fibo`	0	1	1	2	3	5	8	...

Stream functions $A^{\mathbb{N}} \to B^{\mathbb{N}}$ are naturally isomorfic to $A^{\mathbb{N}} \times \mathbb{N} \to B$

Comonads for stream functions

General stream functions

- Functor: $DA = A^{\mathbb{N}} \times \mathbb{N}$
- Input streams with past/present/future:

$$a_0, a_1, \ldots, a_{n-1}, \boxed{a_n}, a_{n+1}, a_{n+2}, \ldots$$

- Counit:

$$\begin{array}{rcl} \varepsilon_A & : & A^{\mathbb{N}} \times \mathbb{N} \to A \\ (\alpha, n) & \mapsto & \alpha(n) \end{array}$$

- Comultiplication:

$$\begin{array}{rcl} \delta_A & : & A^{\mathbb{N}} \times \mathbb{N} \to (A^{\mathbb{N}} \times \mathbb{N})^{\mathbb{N}} \times \mathbb{N} \\ (\alpha, n) & \mapsto & (\lambda m.(\alpha, m),\ n) \end{array}$$

Comonads for stream functions

Causal stream functions

- Functor: $DA = A^+ \quad (\cong A^* \times A)$
- Input streams with past and present but no future
- Counit:

$$\begin{array}{rcl} \varepsilon_A & : & A^+ \to A \\ [a_0, \ldots, a_n] & \mapsto & a_n \end{array}$$

- Comultiplication:

$$\begin{array}{rcl} \delta_A & : & A^+ \to (A^+)^+ \\ [a_0, \ldots, a_n] & \mapsto & [[a_0], [a_0, a_1], \ldots, [a_0, \ldots, a_n]] \end{array}$$

Anticausal stream functions

- Input streams with present and future but no past
- Functor: $DA = A^{\mathbb{N}} \quad (\cong A \times A^{\mathbb{N}})$

Comonads for attribute grammars

An attribute grammar is a CF grammar augmented with attributes and semantic equations.

Example: preorder numbering of the nodes

$$
\begin{array}{lcl}
S^\ell & \longrightarrow & E \\
S^b & \longrightarrow & S^b_L S^b_R \\
\\
S^b_L.numin & = & S^b.numin + 1 \\
S^b_R.numin & = & S^b_L.numout + 1 \\
S^\ell.numout & = & S^\ell.numin \\
S^b.numout & = & S^b_R.numout
\end{array}
$$

Tree functions where the output at a position depends on the input at that position and around it (synthesized, inherited attributes).

Comonads for attribute grammars

Purely synthesized AG-s

- Functor: $DA \;=\; \mathsf{Tree}\,A \;=\; \mu X.\, A \times (1 + X \times X)$
- Counit:

$$\begin{array}{rcl} \varepsilon_A & : & \mathsf{Tree}\,A \to A \\ (a, s) & \mapsto & a \end{array}$$

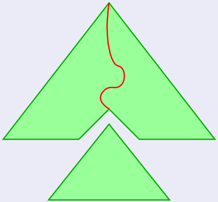
- Comultiplication:

$$\begin{array}{rcl} \delta_A & : & \mathsf{Tree}\,A \to \mathsf{Tree}\,(\mathsf{Tree}\,A) \\ \delta_A(t) & = & \begin{cases} (t, \mathsf{inl}(*)), & \text{if } t = (a, \mathsf{inl}(*)) \\ (t, \mathsf{inr}(\delta_A(t_1), \delta_A(t_2))), & \text{if } t = (a, \mathsf{inr}(t_1, t_2)) \end{cases} \end{array}$$

Comonads for attribute grammars

General AG-s

- Functor: $DA = (2 \times \text{Tree}\, A)^* \times \text{Tree}\, A$
- Path structure from the root to the focus and the local tree below the focus



Pre-[Cartesian closed] co-Kleisli categories

- Extending a pure language (the lambda calculus) with coeffect-constructs, we want the old constructs to remain and not to change their meaning too much.
- If D is a comonad on a Cartesian closed category $\mathcal{C}$, how much of that structure carries over to $\mathbf{CoKl}(D)$?

Products

$$\begin{array}{rcl} A \times^D B & =_{\mathrm{df}} & A \times B \\ \pi_0^D & =_{\mathrm{df}} & \pi_0 \circ \varepsilon \\ \pi_1^D & =_{\mathrm{df}} & \pi_1 \circ \varepsilon \\ \langle k_0, k_1 \rangle^D & =_{\mathrm{df}} & \langle k_0, k_1 \rangle \end{array}$$

Pre-[Cartesian closed] co-Kleisli categories

For (pre-)exponents we need some extra structure on a comonad:

$$
\begin{tikzcd}
D((DA \Rightarrow B) \times A) \arrow[r, "\langle D\pi_0, D\pi_1 \rangle"] \arrow[d, "\mathrm{ev}^D"'] & D((DA \Rightarrow B)) \times DA \arrow[d, "(\varepsilon \times \mathrm{id})"] \\
B & (DA \Rightarrow B) \times DA \arrow[l, "\mathrm{ev}"']
\end{tikzcd}
$$

$$
\frac{D(A \times B) \xrightarrow{k} C}{DA \xrightarrow{\Lambda^D(k)} DB \Rightarrow C}
\qquad
\frac{DA \times DB \xrightarrow{?} D(A \times B) \xrightarrow{k} C}{DA \xrightarrow{\Lambda(k \circ ?)} DB \Rightarrow C}
$$

Pre-[Cartesian closed] co-Kleisli categories

Definition

A comonad D on a [symmetric] [semi]monoidal cat. $\mathcal{C}$ is said to be {lax/strong} [symmetric] [semi]monoidal, if it comes with

- a nat. {transf./iso.} $\mathrm{m} : DA \otimes DB \to D(A \otimes B)$
- [and a nat. {transf./iso.} $\mathrm{e} : I \to DI$]

behaving well wrt. $\alpha, [\mathrm{l}, \mathrm{r},]\ [\gamma,]\ \varepsilon, \delta$.

Pre-exponents

Let D be a comonad on a Cartesian closed cat. $\mathcal{C}$. Assuming that D that is a {lax/strong} [symmetric] [semi]monoidal wrt. the $(1, \times)$ symmetric monoidal structure on $\mathcal{C}$, define this structure on **CoKl**(D):

$$\begin{array}{rcl} A \Rightarrow^D B & =_{\mathrm{df}} & DA \Rightarrow B \\ \mathrm{ev}^D & =_{\mathrm{df}} & \mathrm{ev} \circ \langle \varepsilon \circ D\pi_0, D\pi_1 \rangle \\ \Lambda^D(k) & =_{\mathrm{df}} & \Lambda(k \circ \mathrm{m}) \end{array}$$

Pre-[Cartesian closed] co-Kleisli categories

If D is strong monoidal, then $C \Rightarrow^D -$ is right adjoint to $- \times^D C$ and hence $\Rightarrow^D$ is an exponent functor:

$$\dfrac{\dfrac{D(A \times C) \to B}{DA \times DC \to B}}{DA \to DC \Rightarrow B}$$

However, this seems rare in computational applications, $DA = A^{\mathbb{N}}$ being an atypical example.

Strong symmetric monoidal structure on streams

$$\begin{array}{rcl} \mathrm{m} & : & A^{\mathbb{N}} \times B^{\mathbb{N}} \to (A \times B)^{\mathbb{N}} \\ (\alpha, \beta) & \mapsto & \lambda n.\, (\alpha(n), \beta(n)) \end{array}$$

Pre-[Cartesian closed] co-Kleisli categories

More common is that a comonad is lax symmetric semimonoidal, eg $DA = A^+$, $DA = A^{\mathbb{N}} \times \mathbb{N}$.

Lax symmetric semimonoidal structure on $-^{\mathbb{N}} \times \mathbb{N}$

$$\begin{array}{rcl} \mathrm{m} & : & (A^{\mathbb{N}} \times \mathbb{N}) \times (B^{\mathbb{N}} \times \mathbb{N}) \to (A \times B)^{\mathbb{N}} \times \mathbb{N} \\ ((\alpha, k_1), (\beta, k_2)) & \mapsto & (\lambda n.\,(\alpha(n),\, \beta(n)),\; k_1) \end{array}$$

Then it suffices to have m satisfying $\mathrm{m} \circ \Delta = D\,\Delta$, where $\Delta = \langle \mathrm{id}, \mathrm{id} \rangle : A \to A \times A$ is the semicomonoid structure on the objects of $\mathcal{C}$, to get that $\Rightarrow^{D}$ is a weak exponent functor.

Comonadic semantics

Comonadic semantics is obtained by interpreting the lambda-calculus into $\mathbf{CoKl}(D)$ in the standard way.

Comonadic semantics

$$
\begin{array}{rcccl}
[\![A \times B]\!]^D & =_{\mathrm{df}} & [\![A]\!]^D \times^D [\![B]\!]^D & = & [\![A]\!]^D \times [\![B]\!]^D \\
[\![A \Rightarrow B]\!]^D & =_{\mathrm{df}} & [\![A]\!]^D \Rightarrow^D [\![B]\!]^D & = & D[\![A]\!]^D \Rightarrow [\![B]\!]^D \\
[\![(\underline{x})x_i]\!]^D & =_{\mathrm{df}} & \pi_i^D & = & \pi_i \circ \varepsilon \\
[\![(\underline{x})\mathsf{fst}(t)]\!]^D & =_{\mathrm{df}} & \pi_0^D \circ^D [\![(\underline{x})t]\!]^D & = & \pi_0 \circ [\![(\underline{x})t]\!]^D \\
[\![(\underline{x})\mathsf{snd}(t)]\!]^D & =_{\mathrm{df}} & \pi_1^D \circ^D [\![(\underline{x})t]\!]^D & = & \pi_1 \circ [\![(\underline{x})t]\!]^D \\
[\![(\underline{x})(t_0, t_1)]\!]^D & =_{\mathrm{df}} & \langle [\![(\underline{x})t_0]\!]^D, [\![(\underline{x})t_1]\!]^D \rangle^D & & \\
 & & & = & \langle [\![(\underline{x})t_0]\!]^D, [\![(\underline{x})t_1]\!]^D \rangle \\
[\![(\underline{x})t\ u]\!]^D & =_{\mathrm{df}} & \mathrm{ev}^D \circ^D \langle [\![(\underline{x})t]\!]^D, [\![(\underline{x})u]\!]^D \rangle^D & & \\
 & & & = & \mathrm{ev} \circ \langle [\![(\underline{x})t]\!]^D, ([\![(\underline{x})u]\!]^D)^\dagger \rangle \\
[\![(\underline{x})\lambda x t]\!]^D & =_{\mathrm{df}} & \Lambda^D([\![(\underline{x}, x)t]\!]^D) & = & \Lambda([\![(\underline{x}, x)t]\!]^D \circ \mathrm{m})
\end{array}
$$

Coeffect-specific constructs are interpreted specifically.

Distributive laws

Definition

A distributive law of a monad (T, η, μ) over a comonad (D, ε, δ) is a natural transformation $\lambda_A : DTA \to TDA$ st.

$$\begin{tikzcd} DTA \arrow[r, "\lambda"] \arrow[dr, "\varepsilon"'] & TDA \arrow[d, "T\varepsilon"] \\ & TA \end{tikzcd} \qquad \begin{tikzcd} DTA \arrow[rr, "\lambda"] \arrow[d, "\delta"'] & & TDA \arrow[d, "T\delta"] \\ D^2TA \arrow[r, "D\lambda"] & DTDA \arrow[r, "\lambda"] & TD^2A \end{tikzcd}$$

$$\begin{tikzcd} DA \arrow[d, "D\eta"'] \arrow[dr, "\eta"] & \\ DTA \arrow[r, "\lambda"] & TDA \end{tikzcd} \qquad \begin{tikzcd} DT^2A \arrow[r, "\lambda"] \arrow[d, "D\mu"'] & TDTA \arrow[r, "T\lambda"] & T^2DA \arrow[d, "\mu"] \\ DTA \arrow[rr, "\lambda"] & & TDA \end{tikzcd}$$

Distributive laws

Clocked dataflow computation (partial-stream functions)

$$
\begin{array}{rcl}
TA & = & 1 + A \\
DA & = & A^+ \\
\lambda & : & (1 + A)^+ \to 1 + A^+ \\
as & \mapsto & \begin{cases} \mathrm{inl}(*) & \text{if } \mathrm{last}\,(as) = \mathrm{inl}(*) \\ \mathrm{inr}([a_i \mid \mathrm{inr}(a_i) \leftarrow as]) & \text{otherwise} \end{cases}
\end{array}
$$

Distributive laws

BiKleisli category

Given a monad T and comonad D with a distributive law $\lambda : DTA \to TDA$, the biKleisli category $\mathbf{BiKl}(T, D)$ is defined as:

$$
\begin{array}{lll}
|\mathbf{BiKl}(T, D)| & =_{\mathrm{df}} & |\mathcal{C}| \\
\mathbf{BiKl}(T, D)(A, B) & =_{\mathrm{df}} & \mathcal{C}(DA, TB) \\
\mathrm{id}^{D,T} & =_{\mathrm{df}} & \eta \circ \varepsilon \\
\ell \circ^{D,T} k & =_{\mathrm{df}} & \ell^{\star} \circ \lambda \circ k^{\dagger}
\end{array}
$$

Distributive laws

If $\mathcal{C}$ is Cartesian closed, T is strong, D is lax symmetric semimonoidal, **BiKl**(D, T) carries a pre-[Cartesian closed] structure:

Pre-[Cartesian closed] structure

$$\begin{aligned}
A \times^{D,T} B &=_{\mathrm{df}} A \times B \\
\pi_0^{D,T} &=_{\mathrm{df}} \eta \circ \pi_0 \circ \varepsilon \\
\pi_1^{D,T} &=_{\mathrm{df}} \eta \circ \pi_1 \circ \varepsilon \\
\langle k_0, k_1 \rangle^{D,T} &=_{\mathrm{df}} \sigma_1^{\star} \circ \sigma_0 \circ \langle k_0, k_1 \rangle \\
\\
A \Rightarrow^{D,T} B &=_{\mathrm{df}} DA \Rightarrow TB \\
\mathrm{ev}^{D,T} &=_{\mathrm{df}} \mathrm{ev} \circ \langle \varepsilon \circ D\pi_0, D\pi_1 \rangle \\
\Lambda^{D,T}(k) &=_{\mathrm{df}} \eta \circ \Lambda(k \circ \mathrm{m})
\end{aligned}$$

Conclusions

- The first-order dataflow language agrees perfectly with Lucid and Lustre by its semantics.
- The meaning of higher-order dataflow computation has been unclear. We get a neat semantics from mathematical considerations (cf. Colaço, Pouzet's design with two flavors of function spaces).
- Attribute evaluation and dataflow computation are greatly similar: computation happens on a fixed datastructure, output values are defined uniformly throughout the structure. Therefore the comonadic approach to dataflow computation carries over to attribute evaluation.
- Monadic and comonadic semantics can be combined using distributive laws, giving a setting to uniformly model both effectful and context dependent compuatotions.

Future work

- Dual computational lambda-calculus / comonadic metalanguage.
- General recursion in coKleisli categories.
- Structured recursion/corecursion for dataflow computation.
- Dualization of call-by-name.
- Compilation of comonadic code to automata (cf. Hansen, Costa, Rutten).