Comonadic notions of computation

Tarmo Uustalu\textsuperscript{1} Varmo Vene\textsuperscript{2}

\textsuperscript{1}Institute of Cybernetics, Tallinn
\textsuperscript{2}University of Tartu

FMCS 2006, Kananaskis, 8 June 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 $A \rightarrow B$ in the Kleisli category, i.e., a map $A \rightarrow TB$ in the base category.
- Some examples applied in semantics:

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

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

Definition

A comonad on category \( C \) is given by

- a functor \( D : C \to C \)
- a natural transformation \( \varepsilon_A : DA \to A \)
  - counit of the comonad
- a natural transformation \( \delta_A : DA \to D^2A \)
  - comultiplication of the comonad

s.t. following diagrams commute
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:
  \[
  \varepsilon_A : A \times E \to A \\
  (a, e) \mapsto a
  \]
- Comultiplication:
  \[
  \delta_A : A \times E \to (A \times E) \times E \\
  (a, e) \mapsto ((a, e), e)
  \]
Comonads

Streams comonad

- **Functor:** \( DA = A^\mathbb{N} = \nu X. A \times X \)

- **Counit:**
  \[ \varepsilon_A : A^\mathbb{N} \rightarrow A \]
  \[ \alpha \mapsto \alpha(0) \]

- **Comultiplication:**
  \[ \delta_A : A^\mathbb{N} \rightarrow (A^\mathbb{N})^\mathbb{N} \]
  \[ \alpha \mapsto \lambda n. (\lambda m. \alpha(n + m)) \]
  \[ [a_0, a_1, a_2, \ldots] \mapsto [[a_0, a_1, a_2, \ldots], [a_1, a_2, a_3 \ldots], \ldots] \]
Comonads for stream functions

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

Example: simple dataflow programs

\[
\begin{align*}
pos &= 0 \text{ fby } (pos + 1) \\
sum \, x &= x + (0 \text{ fby } (sum \, x)) \\
fact &= 1 \text{ fby } (fact \ast (pos + 1)) \\
fibo &= 0 \text{ fby } (fibo + (1 \text{ fby } fibo))
\end{align*}
\]

<table>
<thead>
<tr>
<th>pos</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum pos</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>10</td>
<td>15</td>
<td>21</td>
<td>...</td>
</tr>
<tr>
<td>fact</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>24</td>
<td>120</td>
<td>720</td>
<td>...</td>
</tr>
<tr>
<td>fibo</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>...</td>
</tr>
</tbody>
</table>

Stream functions \( A^\mathbb{N} \rightarrow B^\mathbb{N} \) are naturally isomorphic to \( A^\mathbb{N} \times \mathbb{N} \rightarrow B \)
# Comonads for stream functions

## General stream functions

- **Functor:** \( DA = A^\mathbb{N} \times \mathbb{N} \)

- **Input streams with past/present/future:**
  
  \[ a_0, a_1, \ldots, a_{n-1}, \underbrace{a_n}, a_{n+1}, a_{n+2}, \ldots \]

- **Counit:**
  
  \[
  \varepsilon_A : A^\mathbb{N} \times \mathbb{N} \to A \\
  (\alpha, n) \mapsto \alpha(n)
  \]

- **Comultiplication:**
  
  \[
  \delta_A : A^\mathbb{N} \times \mathbb{N} \to (A^\mathbb{N} \times \mathbb{N})^\mathbb{N} \times \mathbb{N} \\
  (\alpha, n) \mapsto (\lambda m.(\alpha, m), n)
  \]
Comonads for stream functions

Causal stream functions

- **Functor:** \( DA = A^+ \) \( (\cong A^* \times A) \)
- **Input streams with past and present but no future**
- **Counit:**
  \[ \varepsilon_A : A^+ \to A \]
  \[ [a_0, \ldots, a_n] \mapsto a_n \]
- **Comultiplication:**
  \[ \delta_A : A^+ \to (A^+)^+ \]
  \[ [a_0, \ldots, a_n] \mapsto [[a_0], [a_0, a_1], \ldots, [a_0, \ldots, a_n]] \]

Anticausal stream functions

- **Input streams with present and future but no past**
- **Functor:** \( DA = A^N \) \( (\cong A \times A^N) \)
Comonads for attribute grammars

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

Example: preorder numbering of the nodes

\[
\begin{align*}
S^\ell & \rightarrow E \\
S^b & \rightarrow S^b S^b \\
S^b_{\text{numin}} & = S^b_{\text{numin}} + 1 \\
S^b_{\text{numout}} & = S^b_{\text{numout}} + 1 \\
S^\ell_{\text{numout}} & = S^\ell_{\text{numin}} \\
S^b_{\text{numout}} & = S^b_{\text{numout}}
\end{align*}
\]

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:** \( DA = \text{Tree } A = \mu X. A \times (1 + X \times X) \)
- **Counit:** \( \varepsilon_A : \text{Tree } A \to A \)
  \[ (a, s) \mapsto a \]
- **Comultiplication:**
  \[ \delta_A : \text{Tree } A \to \text{Tree } (\text{Tree } A) \]
  \[ \delta_A(t) = \begin{cases} 
  (t, \text{inl}(\ast)), & \text{if } t = (a, \text{inl}(\ast)) \\
  (t, \text{inr}(\delta_A(t_1), \delta_A(t_2))), & \text{if } t = (a, \text{inr}(t_1, t_2)) 
\end{cases} \]
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 \( \text{CoKL}(D) \)?

Products

\[
\begin{align*}
A \times^D B &= \text{df} \quad A \times B \\
\pi_0^D &= \text{df} \quad \pi_0 \circ \varepsilon \\
\pi_1^D &= \text{df} \quad \pi_1 \circ \varepsilon \\
\langle k_0, k_1 \rangle^D &= \text{df} \quad \langle k_0, k_1 \rangle
\end{align*}
\]
Pre-[Cartesian closed] co-Kleisli categories

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

$$D((DA \Rightarrow B) \times A) \xrightarrow{\langle D\pi_0, D\pi_1 \rangle} D((DA \Rightarrow B)) \times DA$$

$$B \xleftarrow{ev^D} \quad \xrightarrow{ev} \quad (DA \Rightarrow B) \times DA$$

$$D(A \times B) \xrightarrow{k} C$$

$$DA \xrightarrow{\Lambda^D(k)} DB \Rightarrow C$$

$$DA \xrightarrow{DA \times DB \Rightarrow D(A \times B) \xrightarrow{k} C}$$

$$DA \xrightarrow{DA \times DB \Rightarrow C}$$
Definition

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

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

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

Pre-exponents

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

$A \Rightarrow^D B \equiv_{df} DA \Rightarrow B$

$\text{ev}^D \equiv_{df} \text{ev} \circ \langle \varepsilon \circ D\pi_0, D\pi_1 \rangle$

$\Lambda^D(k) \equiv_{df} \Lambda(k \circ m)$
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:

$$
\begin{align*}
D(A \times C) &\rightarrow B \\
DA \times DC &\rightarrow B \\
DA &\rightarrow DC \Rightarrow B
\end{align*}
$$

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

Strong symmetric monoidal structure on streams

$$
\begin{align*}
m : A^N \times B^N &\rightarrow (A \times B)^N \\
(\alpha, \beta) &\mapsto \lambda n. (\alpha(n), \beta(n))
\end{align*}
$$
Pre-[Cartesian closed] co-Kleisli categories

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

**Lax symmetric semimonoidal structure on $-^N \times N$**

\[
m : (A^N \times N) \times (B^N \times N) \rightarrow (A \times B)^N \times N
\]

\[
((\alpha, k_1), (\beta, k_2)) \mapsto (\lambda n. (\alpha(n), \beta(n)), k_1)
\]

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

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[A \times B]^D$</td>
<td>$\text{df}$ $[A]^D \times [B]^D$</td>
</tr>
<tr>
<td>$[A \Rightarrow B]^D$</td>
<td>$\text{df}$ $[A]^D \Rightarrow [B]^D$</td>
</tr>
<tr>
<td>$[(\times)x]^D$</td>
<td>$\text{df}$ $\pi_i^D$</td>
</tr>
<tr>
<td>$[(\times)\text{fst}(t)]^D$</td>
<td>$\text{df}$ $\pi_0^D \circ^D [(\times)t]^D$</td>
</tr>
<tr>
<td>$[(\times)\text{snd}(t)]^D$</td>
<td>$\text{df}$ $\pi_1^D \circ^D [(\times)t]^D$</td>
</tr>
<tr>
<td>$[(\times)(t_0, t_1)]^D$</td>
<td>$\text{df}$ $\langle [(\times)t_0]^D, [(\times)t_1]^D \rangle^D$</td>
</tr>
<tr>
<td>$[(\times)t\ u]^D$</td>
<td>$\text{df}$ $\text{ev}^D \circ^D \langle [(\times)t]^D, [(\times)u]^D \rangle^D$</td>
</tr>
<tr>
<td>$[(\times)\lambda x t]^D$</td>
<td>$\text{df}$ $\Lambda^D([[(\times, x)t]^D])$</td>
</tr>
</tbody>
</table>

Coeffect-specific constructs are interpreted specifically.
Comonadic semantics

- \( x : C \vdash t : A \) implies \( \llbracket (x) t \rrbracket^D : \llbracket C \rrbracket^D \rightarrow^D \llbracket A \rrbracket^D \), but not all equations of the lambda-calculus are validated.

- Closed terms: Type soundness for \( \vdash t : A \) says that \( \llbracket t \rrbracket^D : 1 \rightarrow^D \llbracket A \rrbracket^D \), i.e., \( D1 \rightarrow \llbracket A \rrbracket^D \), so closed terms are evaluated relative to a coeffect over 1.

In case of general or causal stream functions, this is a list over 1, the time from the start.

If \( D \) is properly (symmetric) monoidal (e.g., \( (\_)^N \)), we have a canonical choice \( e : 1 \rightarrow D1 \).

- Comonadic dataflow language semantics: The first-order 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).
Distributive laws

Definition

A **distributive law** of a monad \((T, \eta, \mu)\) over a comonad \((D, \varepsilon, \delta)\) is a natural transformation \(\lambda_A : DTA \to TDA\) st.
Distributive laws

Clocked dataflow computation (partial-stream functions)

\[
\begin{align*}
TA &= 1 + A \\
DA &= A^+ \\
\lambda : & (1 + A)^+ \to 1 + A^+ \\
as & \mapsto \begin{cases} 
\text{inl}(*), & \text{if last}(as) = \text{inl}(*) \\
\text{inr}([a_i | \text{inr}(a_i) \leftarrow as]), & \text{otherwise}
\end{cases}
\end{align*}
\]
Distributive laws

BiKleisli category

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

$$\text{BiKL}(T, D) = \{ C | C(DA, TB) \}$$

$$\text{id}^{D,T} = \eta \circ \varepsilon$$

$$\ell \circ D,T k = \ell^* \circ \lambda \circ k^\dagger$$
Distributive laws

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

**Pre-[Cartesian closed] structure**

\[
\begin{align*}
A \times^{D, T} B &= \text{df} \quad A \times B \\
\pi_0^{D, T} &= \text{df} \quad \eta \circ \pi_0 \circ \varepsilon \\
\pi_1^{D, T} &= \text{df} \quad \eta \circ \pi_1 \circ \varepsilon \\
\langle k_0, k_1 \rangle^{D, T} &= \text{df} \quad \sigma_1^* \circ \sigma_0 \circ \langle k_0, k_1 \rangle \\
A \Rightarrow^{D, T} B &= \text{df} \quad DA \Rightarrow TB \\
e^{D, T} &= \text{df} \quad \text{ev} \circ \langle \varepsilon \circ D\pi_0, D\pi_1 \rangle \\
\Lambda^{D, T}(k) &= \text{df} \quad \eta \circ \Lambda(k \circ m)
\end{align*}
\]
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).