

Precise comonads for dataflow computation and tree transformations

*Not obsessed with monads so much more,
still obsessed with comonads...*

Tarmo Uustalu, Tallinn
Joint work with Varmo Vene, Tartu

EffTT, Tallinn, 13–14 December 2007

Dataflow computation

- Dataflow computations = discrete-time signal transformations = stream functions.
- The output value at a time instant (stream position) is determined by the input value at the **same** instant (position) plus **further** input values.
- General dataflow, dependence on past and future / causal dataflow, dependence on past alone.
- Lucid, French synchronous languages (Lustre, Lucid Synchrone).
- (Related to Mealy machines.)

Example dataflow programs

pos = 0 fby (pos + 1)
sum x = x + (0 fby (sum x))
fact = 1 fby (fact * (pos + 1))
fibo = 0 fby (fibo + (1 fby fibo))

5	5	5	5	5	5	5	5	...
17 fby 5	17	5	5	5	5	5	5	...
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	...

('fby' ('followed by')) means unit delay)

Tree transformations

- Attribute evaluation = tree relabelling transformations.
- The label at a position in the output tree is determined by the label in the **same** position in the input tree plus **further** labels of the input tree (below or anywhere).
- Purely synthesized, dependence on nodes below / general attribute grammars, dependence on nodes both below and above-aside.
- (Related to relabelling tree transducers.)

Example attribute grammar

$$S^{\ell} \longrightarrow E$$

$$S^b \longrightarrow S_L^b S_R^b$$

$$S^{\ell}.avl = tt$$

$$S^b.avl = S_L^b.avl \wedge S_R^b.avl \wedge S^b.locavl$$

$$S^{\ell}.locavl = tt$$

$$S^b.locavl = |S_L^b.height - S_R^b.height| \leq 1$$

$$S^{\ell}.height = 0$$

$$S^b.height = \max(S_L^b.height, S_R^b.height) + 1$$

Context-dependent computation

- Common to both dataflow computation and tree transformations is computation in a datastructure (container).
- The shape of the datastructure is kept, the computation for every position is
 - **local**, although context-dependent,
 - **uniform**, follows the same rule.

This talk

- Moggi's, late 1980s: analysis of different notions of effectful (cbv) computation in terms Kleisli categories of strong monads.
- Brookes, Geva and Stone, early 1990s: coKleisli categories of “computational” comonads for “intensional semantics” .
- Our 2 cent: CoKleisli categories of **symmetric (semi)monoidal comonads** are a setting to analyse notions of context-dependent computation such as dataflow computation and tree transformations ...
- ... and sometimes you may want a comonad on a **functor category** instead of your base category for things to work as they should.

Outline

- Comonads and context-dependent computation
(cf monads and effectful computation)
- Symmetric (semi)monoidal monads and
context-dependent computation with products and
function spaces
(cf strong monads and effectful computation with
products and function spaces)
- Semantics of context-dependent languages
(cf Kleisli semantics of effectful languages)
- Refined comonads on functor categories
- Examples: dataflow computation, attribute evaluation

Comonads

- Comonads are the dual of monads.
- A *comonad* is
 - a functor $D : \mathcal{C} \rightarrow \mathcal{C}$ (the *underlying functor*),
 - a natural transformation $\varepsilon : D \rightarrow \text{Id}_{\mathcal{C}}$ (the *counit*),
 - a natural transformation $\delta : D \rightarrow DD$ (the *comultiplication*)

satisfying these conditions:

$$\begin{array}{ccc} DA & \xrightarrow{\delta_A} & DDA \\ \delta_A \downarrow & \searrow & \downarrow D\varepsilon_A \\ DDA & \xrightarrow{\varepsilon_{DA}} & DA \end{array} \qquad \begin{array}{ccc} DA & \xrightarrow{\delta_A} & DDA \\ \delta_A \downarrow & & \downarrow D\delta_A \\ DDA & \xrightarrow{\delta_{DA}} & DDDA \end{array}$$

- In other words, a comonad is a comonoid in $[\mathcal{C}, \mathcal{C}]$ (a monoid in $[\mathcal{C}, \mathcal{C}]^{\text{op}}$).

CoKleisli category of a comonad

- A comonad D on a category \mathcal{C} induces a category **CoKI**(D) called the *coKleisli category* of D defined by
 - an object is an object of \mathcal{C} ,
 - a map of from A to B is a map of \mathcal{C} from DA to B ,
 - $\text{id}_A^D =_{\text{df}} DA \xrightarrow{\varepsilon_A} A$,
 - if $k : A \rightarrow^D B$, $\ell : B \rightarrow^D C$, then
$$\ell \circ^D k =_{\text{df}} DA \xrightarrow{k^\dagger} DB \xrightarrow{\ell} C$$
 where
$$k^\dagger =_{\text{df}} DA \xrightarrow{\mu_A} DDA \xrightarrow{Dk} DB.$$
- From \mathcal{C} there is an identity-on-objects *inclusion functor* J to **CoKI**(D), defined on maps by
 - if $f : A \rightarrow B$, then
$$Jf =_{\text{df}} DA \xrightarrow{\varepsilon_A} A \xrightarrow{f} B = DA \xrightarrow{Df} DB \xrightarrow{\varepsilon_B} B.$$
- The functor J has a left adjoint $U : \mathbf{CoKI}(D) \rightarrow \mathcal{C}$ given by $UA =_{\text{df}} DA$, if $k : A \rightarrow^D B$, then $Uk =_{\text{df}} DA \xrightarrow{k^\dagger} DB$.

Comonadic notions of computation

- We think of \mathcal{C} as the category of pure functions and of DA as the type of effectful computations of values of type A (values in context).
- **CoKI**(D) is the category of context-dependent functions.
- $\varepsilon_A : DA \rightarrow A$ is the identity on A seen as trivially context-dependent (discarding the context).
- $Jf : DA \rightarrow B$ is a general pure function $f : A \rightarrow B$ regarded as trivially context-dependent.
- $\delta_A : DA \rightarrow DDA$ blows the context of a value up (duplicates the context).
- $k^\dagger : DA \rightarrow DB$ is a context-dependent function $k : DA \rightarrow B$ extended into one that can output a value in a context (e.g., for a postcomposed context-dependent function).

Simplest (computational) examples

- Product comonad, for dependency on an environment:
 - $DA =_{\text{df}} A \times E$ where E is an object of \mathcal{C} ,
 - $\varepsilon_A =_{\text{df}} A \times E \xrightarrow{\text{fst}} A$,
 - $\delta_A =_{\text{df}} A \times E \xrightarrow{\langle \text{id}, \text{snd} \rangle} (A \times E) \times E$.
- This is the dual of the coproduct monad for exceptions.
- It is not very interesting, as $\mathbf{CoKI}(D) \cong \mathbf{KI}(T)$ for $TA =_{\text{df}} E \Rightarrow A$.

- Exponent comonad:

- $DA =_{\text{df}} E \Rightarrow A$ where (E, e, m) is a monoid in \mathcal{C} ,

- $\varepsilon_A =_{\text{df}} (E \Rightarrow A) \xrightarrow{\text{ur}^{-1}} (E \Rightarrow A) \times 1$

- $\delta_A =_{\text{df}} \Lambda(\Lambda(((E \Rightarrow A) \times E) \xrightarrow{\text{id} \times e} (E \Rightarrow A) \times E \xrightarrow{\text{ev}} A, (E \Rightarrow A) \times E \xrightarrow{a} (E \Rightarrow A) \times (E \times E) \xrightarrow{\text{id} \times m} (E \Rightarrow A) \times E \xrightarrow{\text{ev}} A)),$

- Interesting special cases are $(E, e, m) =_{\text{df}} (\text{Nat}, 0, +)$ and $(E, e, m) =_{\text{df}} (\text{Nat}, 0, \max)$.

- Costate comonad:
 - $DA =_{\text{df}} (P \Rightarrow A) \times P$ where P is an object of \mathcal{C} ,
 - $\varepsilon_A =_{\text{df}} (P \Rightarrow A) \times P \xrightarrow{\text{ev}} A$,
 - $\delta_A =_{\text{df}} (P \Rightarrow A) \times P \xrightarrow{\text{coev} \times \text{id}} (P \Rightarrow ((P \Rightarrow A) \times P)) \times P$.
- This comonad arises from the adjunction $P \times - \dashv P \Rightarrow -$. Composition the other way around gives the state monad $TA =_{\text{df}} P \Rightarrow (A \times P)$.

Comonads for dataflow computation

- We are interested in general/causal/anticausal stream functions $\text{Str}A \rightarrow \text{Str}B$ where

$$\text{Str}A =_{\text{df}} \nu X. A \times X$$

which we would like to see as context-dependent functions from A to B .

- Streams are naturally isomorphic to functions from natural numbers:

$$\text{Str}A =_{\text{df}} \nu X. A \times X \cong \text{Nat} \Rightarrow A$$

- *General* stream functions $\text{Str}A \rightarrow \text{Str}B$ are thus in natural bijection with maps $(\text{Nat} \Rightarrow A) \times \text{Nat} \rightarrow B$.

- The functor

$$DA =_{\text{df}} (\text{Nat} \Rightarrow A) \times \text{Nat}$$

is a comonad (streams with a position comonad), a special case of the costate comonad, so maps $(\text{Nat} \Rightarrow A) \times \text{Nat} \rightarrow B$ are coKleisli maps.

- The coKleisli identities and composition agree with the stream function identities and composition.
- Important operations supported are $\text{fby} : A \times DA \rightarrow A$ and $\text{next} : DA \rightarrow A$ for unit delay and anticipation.

- Comonad for general dataflow, concretely:

$$DA =_{\text{df}} (\text{Nat} \Rightarrow A) \times \text{Nat}$$

$$\begin{aligned} \varepsilon_A &: (\text{Nat} \Rightarrow A) \times \text{Nat} &\rightarrow A \\ & (f, n) &\mapsto f\ n \end{aligned}$$

$$\begin{aligned} \delta_A &: (\text{Nat} \Rightarrow A) \times \text{Nat} &\rightarrow (\text{Nat} \Rightarrow ((\text{Nat} \Rightarrow A) \times \text{Nat})) \times \text{Nat} \\ & (f, n) &\mapsto (\lambda m. (f, m), n) \end{aligned}$$

$$\begin{aligned} \text{fby}_A &: A \times ((\text{Nat} \Rightarrow A) \times \text{Nat}) &\rightarrow A \\ & (a_{00}, (f, 0)) &\mapsto a_{00} \\ & (a_{00}, (f, n + 1)) &\mapsto f\ n \end{aligned}$$

$$\begin{aligned} \text{next}_A &: (\text{Nat} \Rightarrow A) \times \text{Nat} &\rightarrow A \\ & (f, n) &\mapsto f(n + 1) \end{aligned}$$

- A position in a stream splits it into two parts: elements before and after (and including) that position:

$$(\text{Nat} \Rightarrow A) \times \text{Nat} \cong \text{List}A \times \text{Str}A \cong \text{List}A \times (A \times \text{Str}A)$$

- Accordingly, **causal** stream functions are coKleisli maps of the comonad

$$DA =_{\text{df}} \text{List}A \times A \cong \text{NEList}A =_{\text{df}} \mu X. A \times (1 + X)$$

(cofree recursive comonad on $HX =_{\text{df}} 1 + X$, nonempty list comonad).

- and **anticausal** stream functions are coKleisli maps of the comonad

$$DA =_{\text{df}} \text{Str}A \cong \text{Nat} \Rightarrow A$$

(stream comonad) which is a special case of the exponent comonad $DA =_{\text{df}} S \Rightarrow A$ with $(S, e, m) =_{\text{df}} (\text{Nat}, 0, +)$.

- The nonempty list comonad supports fby, the stream comonad supports next.

- Comonad for causal dataflow, concretely:

$$DA =_{\text{df}} \text{NEList } A$$

$$\begin{aligned} \varepsilon_A : \quad & \text{NEList } A && \rightarrow & A \\ & (a_0, \dots, a_{n-1}, a_n) && \mapsto & a_n \end{aligned}$$

$$\begin{aligned} \delta_A : \quad & \text{NEList } A && \rightarrow & \text{NEList } (\text{NEList } A) \\ & (a_0, \dots, a_{n-1}, a_n) && \mapsto & ((a_0), \dots, (a_0, \dots, a_{n-1}), \\ & && & (a_0, \dots, a_{n-1}, a_n)) \end{aligned}$$

$$\begin{aligned} \text{fby}_A : \quad & A \times \text{NEList } A && \rightarrow & A \\ & (a_{00}, (a_0)) && \mapsto & a_{00} \\ & (a_{00}, (a_0, \dots, a_n, a_{n+1})) && \mapsto & a_n \end{aligned}$$

- Comonad for anticausal dataflow, concretely:

$$DA =_{\text{df}} \text{Str}A$$

$$\begin{aligned} \varepsilon_A : \quad & \text{Str}A && \rightarrow & A \\ & (a_n, a_{n+1}, \dots) && \mapsto & a_n \end{aligned}$$

$$\begin{aligned} \delta_A : \quad & \text{Str}A && \rightarrow & \text{Str}(\text{Str}A) \\ & (a_n, a_{n+1}, \dots) && \mapsto & ((a_n, a_{n+1}, \dots), (a_{n+1}, a_{n+2}, \dots), \dots) \end{aligned}$$

$$\begin{aligned} \text{next}_A : \quad & \text{Str}A && \rightarrow & A \\ & (a_n, a_{n+1}, \dots) && \mapsto & a_{n+1} \end{aligned}$$

Comonads for relabelling tree transformations

- Let $H : \mathcal{C} \rightarrow \mathcal{C}$. Define

$$\text{Tree}A =_{\text{df}} \mu X. A \times HX$$

- We are interested in relabelling functions $\text{Tree}A \rightarrow \text{Tree}B$. (Alt. we can define $\text{Tree}^\infty A =_{\text{df}} \nu X. A \times HX$ and interest ourselves in relabelling functions $\text{Tree}^\infty A \rightarrow \text{Tree}^\infty B$.)
- Comonad for general relabelling functions:

$$DA =_{\text{df}} \text{Tree}'A \times A \cong \text{Path}A \times \text{Tree}A \cong \text{Path}A \times A \times H(\text{Tree}A)$$

where

$$\text{Path}A =_{\text{df}} \text{List}(A \times H'(\text{Tree}A))$$

(Huet's zipper).

- E.g., for $HX =_{\text{df}} 1 + X \times X$, $H'X \cong 2 \times X$ and $\text{Path}A \cong \text{List}(A \times 2 \times \text{Tree}A)$.

- Comonad for bottom-up relabelling functions:

$$DA =_{\text{df}} \text{Tree}A$$

- The important operations are those for navigation in a zipper.

Comonad for general relabelling of containers

- Streams and trees are a special case of containers, i.e., functors

$$FA =_{\text{df}} \prod_{s \in S} (P_s \Rightarrow A)$$

- Shape-preserving functions $FA \rightarrow FB$ are families of maps $(P_s \Rightarrow A \rightarrow P_s \Rightarrow B)_{s \in S}$,
i.e., maps $\prod_{s \in S} ((P_s \Rightarrow A) \times P_s) \rightarrow B$.
- The functor

$$DA =_{\text{df}} \prod_{s \in S} ((P_s \Rightarrow A) \times P_s) \cong F'A \times A$$

is the comonad here.

Symmetric monoidal comonads

- A *strong/lax symmetric monoidal functor* between symmetric monoidal categories $(\mathcal{C}, I, \otimes)$ and $(\mathcal{C}', I', \otimes')$ is
 - a functor on $F : \mathcal{C} \rightarrow \mathcal{C}'$
 - together with an isomorphism/map $e : I' \rightarrow FI$
 - and a natural isomorphism/transformation with components $m_{A,B} : FA \otimes' FB \rightarrow F(A \otimes B)$

satisfying

$$\begin{array}{ccc}
 FA \otimes' I' & \xrightarrow{\text{id} \otimes' e'} & FA \otimes' FI & \xrightarrow{m_{A,I}} & F(A \otimes I) & & FA \otimes' FB & \xrightarrow{m_{A,B}} & F(A \otimes B) \\
 \text{ur}'_{FA} \downarrow & & & & \downarrow F_{ur_A} & & c'_{FA,FB} \downarrow & & \downarrow F_{c_{A,B}} \\
 FA & \xlongequal{\quad\quad\quad} & FA & & & & FB \otimes' FA & \xrightarrow{m_{B,A}} & F(B \otimes A)
 \end{array}$$

$$\begin{array}{ccc}
 (FA \otimes' FB) \otimes' FC & \xrightarrow{m_{A,B} \otimes \text{id}} & F(A \otimes B) \otimes' FC & \xrightarrow{m_{A \otimes B, C}} & F((A \otimes B) \otimes C) \\
 a'_{FA,FB,FC} \downarrow & & & & \downarrow F_{a_{A,B,C}} \\
 FA \otimes' (FB \otimes' FC) & \xrightarrow{\text{id} \otimes m_{B,C}} & FA \otimes' F(B \otimes C) & \xrightarrow{m_{A, B \otimes C}} & F(A \otimes (B \otimes C))
 \end{array}$$

- A *symmetric monoidal natural transformation* between two (strong or lax) symmetric monoidal functors (F, e, m) , (G, e', m') is a natural transformation $\tau : F \rightarrow G$ satisfying

$$\begin{array}{ccc}
 I' & \xrightarrow{e} & FI \\
 \parallel & & \downarrow \tau_I \\
 I' & \xrightarrow{e'} & GI \\
 & & \downarrow \tau_{A \otimes B} \\
 & & GA \otimes' GB \xrightarrow{m'_{A,B}} G(A \otimes B) \\
 & & \uparrow \tau_A \otimes' \tau_B \\
 & & FA \otimes' FB \xrightarrow{m_{A,B}} F(A \otimes B)
 \end{array}$$

- A *strong/lax symmetric monoidal comonad* on a symmetric monoidal category $(\mathcal{C}, I, \otimes)$ is a comonad (D, ε, δ) where D is a strong/lax symmetric monoidal functor (with I, \otimes preserved by e, m) and ε, δ are symmetric monoidal natural transformations.

Examples revisited

- $DA =_{\text{df}} A \times E$ is lax symmetric monoidal as soon as E carries a commutative monoid structure.
- $DA =_{\text{df}} S \Rightarrow E$ is strong symmetric monoidal.
- Hence $DA =_{\text{df}} \text{Str}A \cong \text{Nat} \Rightarrow A$ is strong symmetric monoidal too.
- $DA =_{\text{df}} \text{List}A \times A$ is only lax symmetric semimonoidal. . .

$$\begin{array}{lcl} e & : & 1 \rightarrow \text{NEList}1 \\ & & () \mapsto ? \end{array}$$

$$\begin{array}{lcl} m_{A,B} & : & \text{NEList}A \times \text{NEList}B \rightarrow \text{NEList}(A \times B) \\ & & ((a_0, \dots, a_n), (b_0, \dots, b_n)) \mapsto ((a_0, b_0), \dots, (a_n, b_n)) \\ & & ((a_0, \dots, a_n), (b_0, \dots, b_m)) \mapsto \text{perhaps } ((a_n, a_m)) \end{array}$$

CoKleisli categories and Cartesian closed structure

- Let D be a comonad on a Cartesian closed category \mathcal{C} .
- How much of the structure of \mathcal{C} does **CoKI**(D) inherit?
- Since $J : \mathcal{C} \rightarrow \mathbf{CoKI}(D)$ is a right adjoint and preserves limits, **CoKI**(D) inherits the products of \mathcal{C} . Explicitly, we can define

$$\begin{aligned} 1^D &=_{\text{df}} 1 \\ !^D &=_{\text{df}} ! \\ A \times^D B &=_{\text{df}} A \times B \\ \text{fst}^D &=_{\text{df}} \text{fst} \circ \varepsilon \\ \text{snd}^D &=_{\text{df}} \text{snd} \circ \varepsilon \\ \langle k_0, k_1 \rangle^D &=_{\text{df}} \langle k_0, k_1 \rangle \end{aligned}$$

- If D is $(1, \times)$ strong/lax symmetric semimonoidal, then we can also define

$$\begin{aligned}
 A \Rightarrow^D B &=_{\text{df}} DA \Rightarrow B \\
 \text{ev}^D &=_{\text{df}} \text{ev} \circ \langle \varepsilon \circ D\text{fst}, D\text{snd} \rangle \\
 \Lambda^D(k) &=_{\text{df}} \Lambda(k \circ m)
 \end{aligned}$$

$$D((DA \Rightarrow B) \times A) \xrightarrow{\langle \varepsilon \circ D\text{fst}, D\text{snd} \rangle} (DA \Rightarrow B) \times DA \xrightarrow{\text{ev}} B$$

$$\frac{DC \times DA \xrightarrow{m} D(C \times A) \xrightarrow{k} B}{DC \xrightarrow{\Lambda(k \circ m)} DA \Rightarrow B}$$

- Using a strength (if available) is not a good idea: We have no multiplication

$$DC \times DA \xrightarrow{\text{sl}} D(C \times DA) \xrightarrow{D\text{sr}} DD(C \times A) \xrightarrow{?} D(C \times A)$$

and applying ε or $D\varepsilon$ gives a solution where the order of arguments of a function is important and contexts do not combine:

$$DC \times DA \xrightarrow{\text{id} \times \varepsilon} DC \times A \xrightarrow{\text{sl}} D(C \times A)$$

or

$$DC \times DA \xrightarrow{\varepsilon \times \text{id}} C \times DA \xrightarrow{\text{sr}} D(C \times A)$$

- If D is strong semimonoidal (in which case it is automatically strong symmetric semimonoidal as well), then $A \Rightarrow^D -$ is right adjoint to $- \times^D A$ and hence \Rightarrow^D is an exponent functor:

$$\frac{\frac{D(C \times A) \rightarrow B}{DC \times DA \rightarrow B}}{DC \rightarrow DA \Rightarrow B}$$

- This is the case, e.g., if $DA \cong \nu X.A \times (E \Rightarrow X)$ for some E (e.g., $DA \cong \text{Str}A \cong \nu X.A \times (1 \Rightarrow X)$).

- Often however (if we do not take care), D is only lax symmetric (semi)monoidal.
- Then it suffices to have (e and) m satisfying

$$\begin{array}{ccc}
 DA & \xlongequal{\quad} & DA \\
 \downarrow !_{DA} & & \downarrow D!_A \\
 1 & \xrightarrow{e} & D1
 \end{array}
 \qquad
 \begin{array}{ccc}
 DA & \xlongequal{\quad} & DA \\
 \downarrow \Delta_{DA} & & \downarrow D\Delta_A \\
 DA \times DA & \xrightarrow{m_{A,A}} & D(A \times A)
 \end{array}$$

to get $e \circ !_{D1} = \text{id}_{D1}$ and $m_{A,B} \circ \langle D\text{fst}, D\text{snd} \rangle = \text{id}_{D(A \times B)}$.

- Then \Rightarrow^D is a weak exponent operation on objects.

CoKleisli semantics

- As in the case of Kleisli semantics, we interpret the simply typed lambda-calculus into **CoKI**(D) in the standard way, using its Cartesian (pre)closed structure, getting

$$\begin{aligned} \llbracket K \rrbracket^D &=_{\text{df}} \text{an object of } \mathbf{CoKI}(D) \\ &= \text{that object of } \mathcal{C} \end{aligned}$$

$$\begin{aligned} \llbracket 1 \rrbracket^D &=_{\text{df}} 1^D \\ &= 1 \end{aligned}$$

$$\begin{aligned} \llbracket A \times B \rrbracket^D &=_{\text{df}} \llbracket A \rrbracket^D \times^D \llbracket B \rrbracket^D \\ &= \llbracket A \rrbracket^D \times \llbracket B \rrbracket^D \end{aligned}$$

$$\begin{aligned} \llbracket A \Rightarrow B \rrbracket^D &=_{\text{df}} \llbracket A \rrbracket^D \Rightarrow^D \llbracket B \rrbracket^D \\ &= D\llbracket A \rrbracket^D \Rightarrow \llbracket B \rrbracket^D \end{aligned}$$

$$\begin{aligned} \llbracket \underline{C} \rrbracket^D &=_{\text{df}} \llbracket C_0 \rrbracket^D \times^D \dots \times^D \llbracket C_{n-1} \rrbracket^D \\ &= \llbracket C_0 \rrbracket^D \times \dots \times \llbracket C_{n-1} \rrbracket^D \end{aligned}$$

$$\begin{aligned}
\llbracket (\underline{x}) x_i \rrbracket^D &=_{\text{df}} \pi_i^D \\
&= \pi_i \circ \varepsilon \\
\llbracket (\underline{x}) \text{let } x \leftarrow t \text{ in } u \rrbracket^D &=_{\text{df}} \llbracket (\underline{x}, x) u \rrbracket^D \circ^D \langle \text{id}^D, \llbracket (\underline{x}) t \rrbracket^D \rangle^D \\
&= \llbracket (\underline{x}, x) u \rrbracket^D \circ \langle \varepsilon, \llbracket (\underline{x}) t \rrbracket^D \rangle^\dagger \\
\llbracket (\underline{x}) () \rrbracket^D &=_{\text{df}} !^D \\
&= ! \\
\llbracket (\underline{x}) \text{fst}(t) \rrbracket^D &=_{\text{df}} \text{fst}^D \circ^D \llbracket (\underline{x}) t \rrbracket^D \\
&= \text{fst} \circ \llbracket (\underline{x}) t \rrbracket^D \\
\llbracket (\underline{x}) \text{snd}(t) \rrbracket^D &=_{\text{df}} \text{snd}^D \circ^D \llbracket (\underline{x}) t \rrbracket^D \\
&= \text{snd} \circ \llbracket (\underline{x}) t \rrbracket^D \\
\llbracket (\underline{x}) (t_0, t_1) \rrbracket^D &=_{\text{df}} \langle \llbracket (\underline{x}) t_0 \rrbracket^D, \llbracket (\underline{x}) t_1 \rrbracket^D \rangle^D \\
&= \langle \llbracket (\underline{x}) t_0 \rrbracket^D, \llbracket (\underline{x}) t_1 \rrbracket^D \rangle \\
\llbracket (\underline{x}) \lambda x t \rrbracket^D &=_{\text{df}} \Lambda^D(\llbracket (\underline{x}, x) t \rrbracket^D) \\
&= \Lambda(\llbracket (\underline{x}, x) t \rrbracket^D \circ \text{m}) \\
\llbracket (\underline{x}) t u \rrbracket^D &=_{\text{df}} \text{ev}^D \circ^D \langle \llbracket (\underline{x}) t \rrbracket^D, \llbracket (\underline{x}) u \rrbracket^D \rangle^D \\
&= \text{ev} \circ \langle \llbracket (\underline{x}) t \rrbracket^D, (\llbracket (\underline{x}) u \rrbracket^D)^\dagger \rangle
\end{aligned}$$

- Constructs specific to a particular notion of context are interpreted specifically.
- E.g., for the constructs of a general/causal/anticausal dataflow language we can use the appropriate comonad and define:

$$\begin{aligned} \llbracket (\underline{x}) t_0 \text{ fby } t_1 \rrbracket^D &=_{\text{df}} \text{fby} \circ \langle \llbracket (\underline{x}) t_0 \rrbracket^D, (\llbracket (\underline{x}) t_1 \rrbracket^D)^\dagger \rangle \\ \llbracket (\underline{x}) \text{ next } \rrbracket^D &=_{\text{df}} \text{next} \circ (\llbracket (\underline{x}) t \rrbracket^D)^\dagger \end{aligned}$$

- Again, we have welldefinedness / soundness of typing, in the form $\underline{x} : \underline{C} \vdash t : A$ implies $\llbracket (\underline{x})t \rrbracket^D : \llbracket \underline{C} \rrbracket^D \rightarrow^D \llbracket A \rrbracket^D$. Moreover, all equations of the lambda-calculus are validated for a strong semimonoidal comonad, but not in the lax situation.
- For a closed term $\vdash t : A$, soundness of typing 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 a contextuated value of the unit type.
- If D is monoidal (not just semimonoidal), we have a canonical choice $e : 1 \rightarrow D1$.
- In case of general or causal stream functions, an element of $D1$ is a list over 1, i.e., a natural number, the time elapsed.

Is this semantics right?

- Right wrt. what? We could compare the comonadic generic denotational semantics with some other generic semantics, . . . if we had we one (e.g., operational).
- Or we can compare the comonadic denotational semantics of a specific language to its standard denotational semantics.
- First-order dataflow languages: The comonadic and standard (stream-function) semantics agree fully.
- Higher-orderness: How to combine dataflow constructs and higher-orderness has been unclear. We get a neat semantics of the “natural” higher-order extension of the first-order languages from mathematical considerations (cf. Colaço, Pouzet’s design with two flavors of function spaces).

Issues

- Inaccuracy: Dataflow computation and tree transformations can be analyzed in terms of strong monoidal comonads on $[\mathcal{I}, \mathcal{C}]$ with \mathcal{I} some small category.
- Coproducts and recursion: General recursion vs. guarded recursion for cofree recursive comonads.
- “Dual” Lawvere theories and arrow types/Freyd categories: preclosed rather than premonoidal structure is of interest (with closedness a la Eilenberg-Kelly).
- Comonad resolutions other than coKleisli.
- Operational semantics.
- Combining effects and context-dependence: distributive laws and biKleisli categories, e.g., for clocked dataflow.

The inaccuracy problem

- That the comonads for causal and general dataflow are not strong symmetric monoidal and the coKleisli categories not cartesian closed is . . . (perhaps) wrong: the function space is “too large” for poor reasons.
- We haven’t exploited that stream functions don’t change the shape of a given element (contextually situated value) of $DA =_{\text{df}} \text{List}A \times A$ or $DA =_{\text{df}} \text{List}A \times \text{Str}A$.
- For taking advantage of this, a comonad on a functor category can be used.

A “precise” comonad for causal dataflow

- Instead of a comonad on a base category \mathcal{C} , define one on $[\omega, \mathcal{C}]$ where ω is the poset of natural numbers.

$$(DA)_n =_{\text{df}} \prod_{j=0}^n A_j$$

$$\begin{aligned} (\varepsilon_A)_n : \quad (DA)_n &\rightarrow A_n \\ (a_0, \dots, a_n) &\mapsto a_n \end{aligned}$$

$$\begin{aligned} (\delta_A)_n : \quad (DA)_n &\rightarrow \prod_{j=0}^n (DA)_j \\ (a_0, \dots, a_n) &\mapsto ((a_0), \dots, (a_0, \dots, a_n)) \end{aligned}$$

$$\begin{aligned} (\text{fby}_A)_n : \quad A_0 \times (DA)_n &\rightarrow A_n \\ (a_{00}, (a_0)) &\mapsto a_{00} \\ (a_{00}, (a_0, \dots, a_n, a_{n+1})) &\mapsto A_{n \rightarrow n+1} a_n \end{aligned}$$

- This comonad is unproblematically strong symmetric monoidal in the right way:

$$e_n : 1 \rightarrow (D1)_n$$

$$() \mapsto \underbrace{((), \dots, ())}_{n+1 \text{ times}}$$

$$(m_{A,B})_n : (DA)_n \times (DA)_n \rightarrow (DA)_n$$

$$((a_0, \dots, a_n), (a'_0, \dots, a'_n)) \mapsto ((a_0, a'_0), \dots, (a_n, a'_n))$$

- Of course there is more to worry about, e.g., DA must be functorial, ε_A , δ_A etc. must be natural for any functor A .

$$(DA)_{n \rightarrow n+1} : (DA)_n \rightarrow (DA)_{n+1}$$

$$(a_0, \dots, a_n) \mapsto (a_0, A_{0 \rightarrow 1} a_0, \dots, A_{n \rightarrow n+1} a_n)$$

Related: Semantics of intuitionistic linear and modal logic

- Strong symmetric monoidal comonads (and strong monads) are central in the semantics of intuitionistic linear logic and modal logic to interpret the ! and \Box (\Diamond) operators.
- Linear logic: Benton, Bierman, de Paiva, Hyland; Bierman; Benton; Mellies; Maneggia; etc.
- Modal logic: Bierman, de Paiva.
- Applications to staged computation and semantics of names: Pfenning, Davies, Nanevski.

Conclusions

- Not just “intensional semantics”, but also several important and classical “context-dependent” notions of computation can be analyzed systematically in terms of comonads.
For corresponding extensions of the lambda calculus, a “dual” Moggi-style semantics applies.
- Systematic approach, generic analysis of different notions of computation.
- Known language designs/semantics for these notions quality-checked against category-theoretic criteria of canonicity.
- New designs/semantics, e.g., categorically-motivated semantics of higher-orderness for dataflow languages, neater than the earlier proposals.