Monads and Comonads in Intensional Semantics

Stephen Brookes        Kathryn Van Stone
April 1993
CMU-CS-93-140

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Kleisli categories over monads have been used in denotational semantics to describe functional languages using various notions of computations as values. Kleisli categories over comonads have also been used to describe intensional semantics rather than extensional. This paper explores the possibilities of combining monads and comonads to obtain an intensional semantics using computations as values. We give three alternative ways to combine the two and explore which apply to known monads and comonads of interest. We will also look at various intensional semantics for an example programming language that uses monads for computations and compare them to the original extensional semantics.

93–15189

This research was supported in part by National Science Foundation grant CCR-9006064.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA or the U.S. government.
Keywords: theory, denotational semantics, applicative languages, algebraic approaches to semantics
1. Introduction

Traditionally most denotational semantic interpretations of a functional programming language interpret a program as a function from environments to values. Since these semantics focus exclusively on input-output, or extensional behavior, they cannot easily be used to examine intensional properties of programs, such as order of evaluation or complexity. Also, since they typically use simple values, it is difficult to extend them to reason about programming languages with non-functional elements such as nondeterminism, error handling, and assignments.

When adding such features to a language, originally the denotational semantics was changed on a case-by-case basis, by adding new "values" to represent invalid results or by changing the structure of the values, such as using sets of values for non-deterministic results. Moggi in [14] showed that many of these techniques can be described uniformly using an algebraic structure in category theory called a monad. With a monad, one can develop a formal semantics using category theory for a variety of functional languages that also contain many non-functional elements.

There has been some work examining intensional properties for functional languages. Berry and Curien [5] developed a semantics using the cartesian closed category of concrete data structures and sequential algorithms, which include information on the order of evaluation as well as the final value. Brookes and Geva [6] looked at expanding Berry and Curien's results to parallel computation as well as sequential, and developed general notion of intensional semantics using comonads.

This paper explores the possibilities of combining both methods: using comonads to examine the intensional properties of programs whose extensional properties are modeled by means of a monad. Section 2 defines and gives notation for many of the concepts contained in this paper. It is assumed that the reader has a general knowledge of category theory and domains. Sections 3 and 4 define monads and comonads and give examples that have been useful for examining programs. The monads and comonads are typically defined in as general a fashion as possible, with examples given in specific categories, usually Cont, the category of Scott domains and continuous functions. Section 5 describes the formation of the Kleisli categories from comonads or monads. Sections 6 and 7 describe ways to combine comonads and monads (see [4]) and examines how they work with the examples given earlier. The last section then looks at a simple programming language from [14] and at various semantics using comonads and monads together.

2. Preliminary definitions and notational conventions

2.1. Category theory

There are many books containing the basic categorical concepts used in this paper. The technical report [15] is a good introduction, and [3] is aimed for computer scientists.

A category has binary products if for each pair of objects A and B there is an object $A \times B$ with projection morphisms $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$ such that for all morphisms $f : C \to A$ and $g : C \to B$, there is a unique morphism $(f, g)$ from $C$ to $A \times B$ satisfying $\pi_1 \circ (f, g) = f$ and $\pi_2 \circ (f, g) = g$. For all morphisms $f : A \to C$ and $g : B \to D$, we define $f \times g : A \times B \to C \times D$ to be $(f \circ \pi_1, g \circ \pi_2)$.

Similarly a category has binary coproducts if for every pair of objects $A$ and $B$, there is an object $A + B$ and a pair of injection morphisms $\iota_1 : A \to A + B$ and $\iota_2 : B \to A + B$, such that for all morphisms $f : A \to C$
and $g : B \to C$, there is a unique morphism $[f, g]$ from $A + B$ to $C$ satisfying $[f, g] \circ \iota_1 = f$ and $[f, g] \circ \iota_2 = g$. For all morphisms $f : A \to C$ and $g : B \to D$, we define $f + g : A + B \to C + D$ to be $[\iota_1 \circ f, \iota_2 \circ g]$.

An object $1$ of a category is terminal if for every object $A$, there exists a unique morphism, denoted $!,_A$, from $A$ to $1$.

A category is cartesian closed if it has binary products, a terminal object, and if for each pair of objects $A$ and $B$ there is an exponentiation object $[A - B]$ and a morphism $\text{app}_{A,B} : [A - B] \times A \to C$ such that for all morphisms $f : C \times A \to B$, there is a unique morphism $\text{curry}(f) : C \to [A - B]$ satisfying $f = \text{app}_{A,B} \circ (\text{curry}(f) \times \text{id}_A)$. Here, as usual, $\text{id}_A$ denotes the identity morphism for $A$. Given a morphism $g : C \to [A - B]$ we define $\text{uncurry}(g) : C \times A \to B$ to be $\text{app}_{A,B} \circ (g \times \text{id}_A)$.

A cartesian closed category is set-like if its objects are sets, its morphisms are functions, its products are the usual cartesian products of sets, exponentiation objects $[A - B]$ are subsets of the set of functions from $A$ to $B$, and application is the standard function application.

### 2.2. Partial orders

The definitions for domains and cpo's in this section come primarily from [10]; the discussion on Plotkin orders it taken out of [9].

A poset is a set $D$ with a partial order $\leq_D$, i.e., a binary relation that is reflexive, transitive, and antisymmetric. A subset $X$ of a poset is directed if every finite subset of $X$ has an upper bound in $X$. A partial order $(D, \leq_D)$ is directed complete if for every directed set $X \subseteq D$, the least upper bound of $X$, denoted $\sqcup X$, exists in $D$. A partial order is pointed if it has a least element, usually denoted $\bot$. A cpo is a pointed directed complete partial order.

A subset $X$ of a poset is consistent or bounded if it has an upper bound. A partial order is bounded-complete if every consistent set $X$ has a least upper bound. In a bounded-complete poset, every pair of elements $x, y$ with a lower bound has a greatest lower bound, written as $x \sqcap y$.

An element $k$ in a cpo $D$ is compact (also called finite or isolated) if for every directed set $X$ such that $k \leq_D \sqcup X$, $k$ is less than or equal to some element in $X$. The set of compact elements of $D$ is written as $K(D)$. A cpo $D$ is algebraic if for every $x \in D$, the set $\{k \in K(D) \mid k \leq_D x\}$ is directed and its least upper bound is $x$. A cpo $D$ is $\omega$-algebraic [9] if it is algebraic and the set $K(D)$ is countable. A Scott domain is a bounded-complete, $\omega$-algebraic cpo. In this paper we will often use domain for Scott domain.

A bc-domain is an algebraic bounded-complete cpo. A bc-domain is distributive if for all consistent pairs of elements $y$ and $z$ and all elements $x$ we have that $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$. An algebraic cpo has property I if for each element $x$ there are only finitely many compact elements below it. A dl-domain is a distributive bc-domain that has property I.

A subset $N$ of a poset $A$ is normal if for every $x \in A$, the set $N \cap \{y \in A \mid y \leq_A x\}$ is directed. A poset $A$ is a Plotkin order if for every finite subset $X$ of $A$, there is a finite normal subset $N$ of $A$ with $X \subseteq N$. A bifinite domain is an $\omega$-algebraic cpo $D$ such that $K(D)$ is a Plotkin order.

A function $f$ is monotone if $f(x) \leq f(y)$ whenever $x \leq y$. A function is continuous if it is monotone and preserves least upper bounds of directed sets. A function is strict if it maps least elements to least elements. A continuous function $f : A \to B$ is stable if for all $b \leq f(a)$ the set $\{a' \leq a \mid b \leq f(a')\}$ has a least element, represented as $M(f, a, b)$. Given two functions $f_1$ and $f_2$ from $A$ to $B$, we define the pointwise ordering.
\[ \leq^{A\rightarrow B} \text{ such that } f_1 \leq^{A\rightarrow B} f_2 \text{ if for all } x \in A, f_1(x) \leq^B f_2(x). \] We also define the stable ordering, \( \leq^{A\rightarrow B} \) such that \( f_1 \leq^{A\rightarrow B} f_2 \) if \( f_1 \leq^{A\rightarrow B} f_2 \) and for all \( b \leq f_1(a) \), \( M(f_1, a, b) = M(f_2, a, b) \).

Given two posets \( D_1 \) and \( D_2 \), their separated sum \( D_1 + D_2 \) is a domain consisting of the set \( \{(1, x_1) \mid x_1 \in D_1\} \cup \{(2, x_2) \mid x_2 \in D_2\} \cup \{\perp\} \) with the ordering \( \leq^{D_1 + D_2} \) defined by \( (i, x) \leq^{D_1 + D_2} (i', x') \) whenever \( x \leq^{D_i} x' \) \((i = 1 \text{ or } 2)\) and for all \( y \in D_1 + D_2 \), \( \perp \leq^{D_1 + D_2} y \). The coalesced sum \( D_1 \div D_2 \) of \( D_1 \) and \( D_2 \) is \( (D_1 - \{\perp_{D_1}\}) \cup (D_2 - \{\perp_{D_2}\}) \).

2.3. Specific categories

The categories used for the examples in this paper will generally be drawn from the following list:

Set: The category of sets and functions.

Cont: The category of Scott domains and continuous functions.

Conts: The category of Scott domains and strict continuous functions.

Bif: The category of bifinite domains and continuous functions. The only interest in this category is that bifinite domains are closed under the Plotkin power domain (see section 3.5), but Scott domains are not [9].

dI: The category of dI-domains and stable functions, stably ordered.

All the categories listed above are set-like, except for Conts which is not cartesian closed (since currying does not preserve strictness). Virtually all of the examples that use Cont can be applied to any cartesian closed category of cpo's and continuous functions, and most can be applied to any set-like category.

3. Monads

**Definition 1** A monad on a category \( C \), \( \langle T, \eta, \mu \rangle \) is a functor, \( T : C \rightarrow C \), with two natural transformations, \( \eta : 1 \rightarrow T \) and \( \mu : T^2 \rightarrow T \), satisfying the following conditions for all objects \( A \) of \( C \):

- \( \mu_A \circ \mu_TA = \mu_A \circ T\mu_A \)
- \( \mu_A \circ \eta_TA = \mu_A \circ T\eta_A = 1_{TA} \)

An alternative form is a Kleisli triple, \( \langle T, \eta, \ast \rangle \), where \( T \) is a function from objects to objects, for each object \( A \), \( \eta_A \) is a morphism from \( A \) to \( TA \), and for all morphisms \( f : A \rightarrow TB \), \( f^\ast \) is a morphism from \( TA \) to \( TB \), all satisfying the three conditions below:

- \( f^\ast \circ \eta_A = f \)
- \( \eta_A^\ast = 1_{TA} \)
- \( (g^\ast \circ f)^\ast = g^\ast \circ f^\ast \).
The two forms are equivalent; for any a monad \((T, \eta, \mu)\), there is a Kleisli triple \((T, \eta, \_\_\) where \(T\) is the restriction of the functor \(T\) to objects, \(\eta\) is unchanged, and for all \(f : A \rightarrow TB, f^\_ = \mu_B \circ Tf\). Also for all Kleisli triples \((T, \eta, \_\)\), there is a monad \((T, \eta, \mu)\), where \(Tf = (\eta \circ f)^\_\) and \(\mu = \text{id}\_TA\).

3.1. Identity monad

An obvious monad is the identity monad \((I, \text{id}, \text{id})\), where \(I\) is the identity functor, and the natural transformations are identity morphisms.

3.2. Lifting

In the category \(\text{Cont}\), there is a monad \((L, \text{up, down})\) where

- For all objects \(A, LA = A \cup \{\bot\_LA\}, \text{where} \bot\_LA \notin A\) is a new least element,
- For all morphisms (continuous functions) \(f : A \rightarrow B, Lf(x) = \begin{cases} \bot\_LB & x = \bot\_LA \\ f(x) & \text{otherwise} \end{cases}\)
- For all objects \(A, \text{up}_A : A \rightarrow LA\) is the inclusion function
- For all objects \(A, \text{down}_A : L^2A \rightarrow LA\) is \(\text{down}_A(x) = \begin{cases} \bot\_LA & x = \bot\_L^2A \\ x & \text{otherwise} \end{cases}\)

Lifting can be used (see [14]) to model partiality in programming languages, with the new bottom element representing divergence.

3.3. Coproducts, Disjoint Sums, and Separated Sums

In any category \(C\) with finite coproducts, and a distinguished object \(E\), there is a monad \((E, \eta, \mu)\) such that

- For all objects \(A, EA = A + E\).
• For all \( f : A \to B, Ef : A + E \to B + E \) is \( f \circ \text{id}_E \)
• For all objects \( A, \eta_A : A \to A + B \) is \( \iota_1 \), the corresponding coproduct injection morphism.
• For all objects \( A, \mu_A : (A + E) + E \to A + E \) is \([\text{id}_A, \iota_2]\).

This construction also defines a monad when using separated sums in \( \text{Cont} \), even though they do not satisfy the requirements for a coproduct (the \([-,-]\) constructor is not unique for separated sums).

Coproducts are typically (see [14] and [18]) used to model exceptional handling, where \( E \) represent a set of errors or exceptions.

As an example, in \( \text{Cont} \), let \( E \) be the singleton set \( \{\text{err}\} \). Then the monad \( (E, \eta, \mu) \) looks like

• For all domains \( A, EA = \{(1,a) \mid a \in A\} \cup \{(2, \text{err})\} \cup \{\bot\} \) with the standard separated sum ordering
• For all continuous functions \( f : A \to B \)

\[
Ef(x) = \begin{cases} 
(1, f(a)) & x = (1, a) \\
(2, \text{err}) & x = (2, \text{err}) \\
\bot & x = \bot
\end{cases}
\]

• For all domains \( A, \eta_A(a) = (1, a) \)
• For all domains \( A, \mu_A : (A + E) + E \to (A + E) \) is defined as

\[
\mu_A(x) = \begin{cases} 
y & x = (1, y) \\
(2, \text{err}) & x = (2, \text{err}) \\
\bot & x = \bot
\end{cases}
\]

If we look at separated sums and lifting on predomains (domains that are not necessarily pointed), then lifting is a specific kind of separated sum, namely \( LA = A + \{\} \). In particular any property that is true of all separated sums is also true of all lifted domains, and if a property fails to hold for lifted domains, it also fails to hold for separated sums.

3.4. Products

Let \( C \) be any category with binary products and a terminal object. Then an object \( X \) is a monoid in \( C \), if there exist morphisms \( e : 1 \to X \) and \( m : X \times X \to X \) such that

• \( (m \circ \text{id}_X) \times e = \pi_1 \),
• \( m \circ (e \times \text{id}_X) = \pi_2 \), and
• \( m \circ (m \times \text{id}_X) \circ \alpha_{X,X,X} = m \circ \text{id}_X \times m \), where \( \alpha_{A,B,C} \) is the natural isomorphism from \( A \times (B \times C) \) to \( (A \times B) \times C \).

Given a monoid \( X \), there is then a monad \( (X, \eta, \mu) \) defined as
Figure 2: Identity and Associativity properties of a monoid

- For all objects $A$, $XA = A \times X$.
- For all $f : A \rightarrow B$, $Xf = f \times \text{id}_X$.
- For all objects $A$, $\eta_A : A \rightarrow A \times X$ is $\langle \text{id}_A, e \circ i_A \rangle$.
- For all objects $A$, $\mu_A : (A \times X) \times X \rightarrow A \times X$ is $(\text{id}_A \times m) \circ \alpha_{A,X,X}^{-1}$, where $\alpha^{-1}$ is the natural isomorphism from $(- \times -) \times -$ to $- \times (- \times -)$.

Products can be used to model a simple form of output processing ([18]) or to calculate resources (such as time or space) used in a program ([11]).

As an example, let $C = \text{Cont}$ and let $X = \text{VNat}$, the set of natural numbers (plus $\omega$ to make it directed complete), ordered vertically with least element 0 and greatest element $\omega$. Let $e$ be the constant 0 function, let $m$ be addition. Then $\text{VNat}$ is a monoid, and the monad $(X, \eta, \mu)$ looks like

- For all domains $A$, $XA = A \times \text{VNat}$
- For all functions $f : A \rightarrow B$, $Xf \langle a, n \rangle = \langle f(a), n \rangle$.
- For all domains $A$, $\eta_A(a) = \langle a, 0 \rangle$
- For all domains $A$, $\mu_A(\langle a, n_1 \rangle, n_2) = \langle a, n_1 + n_2 \rangle$.

3.5. Power Sets and Power Domains

In the category $\text{Set}$ the power set constructor forms a monad $(\mathcal{P}, \{ - \}, \cup)$ where

- For all sets $A$, $\mathcal{P}A$ is the power set of $A$.
- For all functions $f : A \rightarrow B$ and all $X \in \mathcal{P}A$, $\mathcal{P}f(X) = \{ f(x) \mid x \in X \}$.
- $\{ - \} : I \rightarrow \mathcal{P}$ forms singleton sets from elements
- $\cup : \mathcal{P}^2 \rightarrow \mathcal{P}$ is set union
Thus we can model nondeterminism ([14]) by having the result of a program be a set of possible final values.

To do the same in Cont, however, we need to form domains out of power sets, in particular we need a partial order on the sets. Since we generally want to make use of the ordering in the underlying set, using set inclusion as the partial order is insufficient. There are three orderings typically used for power set domains (see [9]):

**Hoare or lower ordering:** \( u \subseteq v \) if \( \forall x \in u. \exists y \in v \text{ such that } x \leq y. \)

**Smyth or upper ordering:** \( u \supseteq v \) if \( \forall y \in v. \exists x \in u \text{ such that } x \leq y. \)

**Plotkin or convex ordering:** \( u \supseteq v \) if \( u \supseteq v \) and \( u \subseteq v. \)

These orderings tend to form preorders instead of partial orders but, by a construction taken from [9], we can get back a partial order.

**Definition 2** An ideal over a preorder \( A \) is a nonempty set \( u \subseteq A \) that is directed and downwards closed. An ideal \( u \) is a principal ideal if \( u = \{a \mid b \leq a \} \) for some \( a \in A \). \( \text{Idl}(A) \) is the poset of all ideals in \( A \), ordered by inclusion.

**Theorem 1** [9] For all preorders \( A \), \( \text{Idl}(A) \) is an algebraic cpo, with compact elements being the principal ideals.

Thus for any algebraic cpo \( A \) we can form another algebraic cpo, the power domain

\[ P^t A = \text{Idl}((\mathcal{P}^*_l(K(A))), \subseteq^t) \]

where \( t \in \{\flat, \sharp, \natural\} \) refers to any one of the three preorderings on powersets and \( \mathcal{P}^*_l(X) \) is the set of finite, non-empty subsets of \( X \). Thus an element \( s \in P^t A \) is a (directed, downwards closed) collection of finite sets of compact elements of \( A \).

Although theorem 1 gives us an algebraic cpo, what we are looking for is a Scott domain. If \( A \) is Scott domain, then so are the Smyth and Hoare power domains \( P^l A \) and \( P^r A \). The Plotkin power domain \( P^\natural A \), however, may not even be bounded-complete (for example, consider \( P^\natural(\text{Bool} \times \text{Bool}) \), where \( \text{Bool} \) is the standard flat truth value domain \( \{\text{tt}, \text{ff}, \perp\} \)). While the Plotkin power domain of a Scott domain is not necessarily a Scott domain, the Plotkin power domain of a bifinite domain does remain bifinite (see [9]).

All three power domains can be used to form monads \( \langle P^t, \{\cdot\}^t, [\cdot]^t \rangle \) as follows:

- For all domains \( A \), and \( a \in A \), Let \( \{\cdot\}^t_A : A \rightarrow P^t A \) be the function

\[ \{\cdot\}^t_A = \{ w \in \mathcal{P}^*_l(K(A)) \mid \exists k \in K(A).k \leq A \text{ and } w \subseteq^t \{k\} \} \]

- For all functions \( f : A \rightarrow B \), let \( P^t f : P^t A \rightarrow P^t B \) be defined as

\[ P^t f(s) = \bigcup_{\{a_1, \ldots, a_n\} \in r} ([f(a_1)]^t_B \uplus \cdots \uplus [f(a_n)]^t_B) \]
where for each domain \( A \), \( \otimes_A^j \) is the binary (associative) function from \( P^tA \times P^tA \) to \( P^tA \) defined by

\[
s \otimes_A^j t = \{ w \in P^*_n(K(A)) \mid \exists u \in s. \exists v \in t. w \subseteq^+ u \cup v \}
\]

Combining the above definitions gives us

\[
P^t f (s) = \bigcup \{ w \in P^*_n(K(B)) \mid \forall a \in u. \exists k_a \in K(B). k_a \leq^B f (a) \text{ and } w \subseteq^+ \{ k_a \mid a \in u \} \}
\]

- Given a domain \( A \), \( \otimes_A^j : P^t(P^tA) \rightarrow P^tA \) is defined by

\[
\otimes_A^j (3) = \bigcup \{ \bigcup \{ u \mid u \in \bar{a} \} \mid \bar{a} \in \bar{3} \}
\]

If we only look at compact elements and if we have an \( f : A \rightarrow B \) that preserves compactness, then the functions above simplify to

- For all objects \( A \), \( \otimes_A^j \mid \bigwedge k \{ k \} = \{ k \} \).
- For all objects \( A \), \( \otimes_A^j (\mid v) = \mid (u \cup v) \)
- For all objects \( A \), \( P^t f (\mid \{ a_1, \ldots, a_n \}) = \mid \{ f (a_1) \} \otimes_B^j \ldots \otimes_B^j \{ f (a_1) \} = \mid \{ f (a_1) \}, \ldots, f (a_n) \} \), i.e. \( P^t f (\mid u) = \mid P f (u) \).
- For all objects \( A \), \( \otimes_A^j (\mid \bar{u}) = \bigcup_{\bar{a} \in \bar{u}} u \).

3.6. Exponentiation

Let \( C \) be a cartesian closed category and let \( V \) be any object of \( C \). Then there is a monad \( (V \rightarrow \cdot \text{const.} \cdot \text{diag}) \), where

- For all objects \( A \), \( V^\rightarrow A = \{ V \rightarrow A \} \)
- For all \( f : A \rightarrow B \), \( V^\rightarrow f : \{ V \rightarrow A \} \rightarrow \{ V \rightarrow B \} \) is \( \text{curry}(f \circ \text{app}_{V,A}) \)
- For all objects \( A \), \( \text{const}_A : A \rightarrow \{ V \rightarrow A \} \) is \( \text{curry}(\pi_1) \)
- For all objects \( A \), \( \text{diag}_A : \{ V \rightarrow \{ V \rightarrow A \} \} \rightarrow \{ V \rightarrow A \} \) is \( \text{curry} (\text{app}_{V,A} \circ (\text{app}_{V,V \rightarrow A} \cdot \pi_2)) \)

In \( \text{Cont} \) the monad becomes

- For all domains \( A \), \( V^\rightarrow A \) is the domain of all continuous functions from \( V \) to \( A \).
- For all continuous functions \( f : A \rightarrow B \), \( V^\rightarrow f (\lambda v. a_v) = \lambda v. f (a_v) \).
- For all domains \( A \), \( \text{const}_A (a) = \lambda v. a \).
- For all domains \( A \), \( \text{diag}_A (\lambda v_1, \lambda v_2. a_{v_1, v_2}) = \lambda v. a_{v, v} \)
4. Comonads

A comonad in \( \mathcal{C} \) is a monad in the opposite category \( \mathcal{C}^{\text{op}} \), namely, it is a functor \( G : \mathcal{C} \to \mathcal{C} \), with two natural transformations \( \epsilon : G \to \text{id}_{\mathcal{C}} \) and \( \delta : G^2 \to G \) such that for all objects \( A \) of \( \mathcal{C} \)

- \( \epsilon_A \circ \delta_A = G\epsilon_A \circ \delta_A = \text{id}_{GA} \)
- \( \delta_A \circ \delta_A = G\delta_A \circ \delta_A \)

A comonad also has a form similar to the Kleisli triple: the Kleisli cotriple \( (G, \epsilon^*, \delta^*) \), where \( G \) is a function on objects, and for each object \( A \), \( \epsilon_A \) is a morphism from \( GA \) to \( A \), and for each morphism \( f : GA \to GB \), \( f^* \) is a morphism from \( GA \) to \( GB \), all satisfying the following conditions:

- For all \( f : GA \to GB \), \( \epsilon_B \circ f^* = f \)
- For all objects \( A \), \( \epsilon_A^* = \text{id}_{GA} \)
- For all \( f : GA \to GB \) and \( g : GB \to GC \), \( g^* \circ f^* = (g \circ f^*)^* \).

4.1. Side effects

Let \( \mathcal{C} \) be any cartesian closed category and let \( S \) be an object in \( \mathcal{C} \). Then there is a monad \( (S, \eta, \mu) \) where

- For all objects \( A \), \( SA = [S \to (A \times S)] \)
- For all morphisms \( f : A \to B \), \( Sf : [S \to (A \times S)] \to [S \to (B \times S)] \) is \( \text{curry}(f \times \text{id}_S) \circ \text{app}_{S,A \times S} \)
- For all objects \( A \), \( \eta_A : A \to [S \to (A \times S)] \) is \( \text{curry}(\text{id}_{A \times S}) \)
- For all objects \( A \), \( \mu_A : [S \to ([S \to (A \times S)] \times S)] \to [S \to (A \times S)] \) is \( \text{curry}(\text{app}_{S,A \times S} \circ \text{app}_{S,S \times A \times S}) \)

In Cont the monad becomes:
• For all continuous functions \( f : A \to B \), \( Sf(\lambda s.(a, z)) = \lambda s.(f(a), z) \)

• For all domains \( A \), \( \eta_A(a) = \lambda s.(a, s) \).

• For all domains \( A \), \( \mu_A(\lambda s.(a, z)) = \lambda s.a(z) \)

This monad can be used to model side effects (see [14]), where \( S \) represents some internal state; a computation takes an initial state and returns a value plus a new state.

4.2. Product Comonad

Let \( C \) be a category with binary products and let \( X \) be any object of \( C \). Then there is a comonad \( \langle X, \pi_1, \delta \rangle \) where

• For all objects \( A \), \( XA = A \times X \)

• For all \( f : A \to B \), \( Xf = f \times \text{id}_X \).

• For all \( A \), \( \pi_1 : A \times X \to A \) is the corresponding product projection morphism.

• For all objects \( A \), \( \delta_A : A \times X \to (A \times X) \times X \) is \( (\text{id}_A \times \pi_2) \); in \( \text{Cont} \), \( \delta_A(a, x) = ((a, x). x) \).

4.2.1. Computation paths: Exponentiation comonad

Let \( C \) be a cartesian closed category, and let \( V \) be a monoid in \( C \) with identity element \( e : 1 \to V \) and binary operator \( m : V \times V \to V \). Then there is a comonad \( \langle V, \text{val}, \text{pre} \rangle \) where

• For all objects \( A \), \( VA = [V \to A] \)

• For all \( f : A \to B \), \( Vf : [V \to A] \to [V \to B] \) is \( \text{curry}(f \circ \text{app}_{VA,V}) \)

• For all objects \( A \), \( \text{val}_A : [V \to A] \to A \) is \( \text{app}_{VA,V} \circ (\text{id}_{VA} \circ e \circ V) \)

• For all objects \( A \), \( \text{pre}_A : [V \to A] \to [V - [V \to A]] \) is \( \text{curry}(\text{app}_{VA,V} \circ (\text{id}_{A} \times m) \circ \alpha_{VA,V}^{-1}) \).

To see the above more clearly, in \( \text{Cont} \) it becomes

• For all objects \( A \), \( VA \) is the domain of continuous functions from \( V \) to \( A \)

• For all continuous functions \( f : A \to B \), \( Vf(\lambda v.a_v) = \lambda v.f(a_v) \).

• For all objects \( A \), \( \text{val}_A(\lambda v.a_v) = a_v \).

• For all objects \( A \), \( \text{pre}_A(\lambda v.a_v) = \lambda v_1.\lambda v_2.\alpha_{m(v_1,v_2)} \).
One possible monoid in $\text{Cont}$ has $V = \text{VNat}$, $e = \omega$, and $m$ the greatest lower bound function. With this monoid the monad becomes the path comonad mentioned in [7], where $VA$ can be interpreted as a non-decreasing sequence of elements of $A$. $Vf$ maps $f$ onto the elements of the sequence, $\text{val}$ takes the value of a sequence at $\omega$, which by continuity is also the least upper bound of the elements of the sequence, and $\text{pre}$ returns the sequence of prefixes of a sequence.

### 4.2.2. Strictly increasing paths

When using the path comonad mentioned above, where $V = \text{VNat}$, etc., elements of $VA$ can be thought of as sequences of construction steps used to build a data value. Since the sequence is not necessarily strictly increasing, there may be places in the sequence where the values remain unchanged. To get sequences where the values always increase, we create a variant of the comonad $\langle V, \text{val}, \text{pre} \rangle$ in $\text{Cont}$ using strictly increasing paths. The resulting comonad $\langle V_S, s\text{val}, s\text{pre} \rangle$ is defined as follows:

- For all domains $A$, let $V_SA$ be the set of finite or infinite strictly increasing sequences in $A$. The finite sequences are represented as eventually constant (infinite) sequences. Therefore we have either sequences of the form $(a_n)^{\omega}_{n=0}$, where for each $n$, $a_n <_A a_{n+1}$, or sequences of the form $d_{a_0} a_1 \ldots a_{N-1} \omega$, $a_0 <_A a_1 <_A \ldots <_A a_N$. The ordering $\leq_{V_SA}$ is the least partial ordering such that

  $$a_0 \ldots a_{N-1} a_{N} \leq_{V_SA} a_0 \ldots a_{N-1} a_{N}$$

  whenever $a \in V_SA$ and $a_{N} \leq_A a_{0}$. If we consider a sequence to be a (stable) function from $\text{VNat}$ to $A$, then this ordering of sequences is closely related to the stable ordering on functions.

- For all continuous functions $f : A \rightarrow A'$, let $V_S f$ be the least continuous function such that for all $a, a_0, a_1 \in A$ such that $a_0 <_A a_1$ and for all $a \in V_SA$,

  $$V_S f (a^\omega) = (f(a))^\omega$$

  $$V_S f (a_0 a_1 a) = \begin{cases} (f(a_0))(V_S f (a_1 a)) & f(a_0) \neq f(a_1) \\ V_S f (a_1 a) & f(a_0) = f(a_1) \end{cases}$$

  The definition is similar to the definition of $V$ on functions except that any resulting duplications are removed.

- For all domains $A$, $s\text{val}_A(a) = a_{\omega} = \bigcup_{n=0}^{\omega} a_n$.

- For all domains $A$, $s\text{pre}_A(a) = (a_{\text{min}(i)})^{\omega}_{i=0}$ or equivalently, $s\text{pre}_A(a) = (a_0 \ldots a_{-1} a^\omega_{i=0})^\omega$.

### 5. Kleisli Categories

Given a category $C$ and a monad $\langle T, \eta, \mu \rangle$ there is a category $\mathcal{K}(T)$, the Kleisli category of $T$, defined as follows:

- The objects in $\mathcal{K}(T)$ are the objects in $C$.

- A morphism $f : A \rightarrow B$ in $\mathcal{K}(T)$ is a morphism $f : A \rightarrow TB$ in $C$. 


For all objects $A$ in $\mathcal{K}(T)$, the identity morphism on $A$ is $\eta_A : A \rightarrow TA$.

For all pairs of morphisms $f : A \xrightarrow{T} B$ and $g : B \xrightarrow{T} C$, their composition is $g \circ f = \mu_C \circ Tg \circ f$.

Similarly, for a comonad $(G, \epsilon, \delta)$, there is a Kleisli category $\mathcal{K}(G)$, where

- The objects in $\mathcal{K}(G)$ are the objects in $C$.
- A morphism $f : A \xrightarrow{G} B$ in $\mathcal{K}(G)$ is a morphism $f : GA \rightarrow B$ in $C$.
- For all objects $A$ in $\mathcal{K}(G)$, its identity arrow is $\epsilon_A : GA \rightarrow A$.
- For all pairs of morphisms $f : A \xrightarrow{G} B$ and $g : B \xrightarrow{G} C$, their composition is $g \circ f = g \circ Gf \circ \delta_A$.

[7] and [14] (and section 6.1) show how to obtain semantic interpretations of languages using Kleisli categories. In particular the meaning of a functional program becomes a morphism in the Kleisli category, and the Kleisli category composition rules are used instead of composition in the original category.

6. Double Kleisli Categories

The Kleisli category of a monad is generally used to represent a semantic interpretation of a functional language with added non-functional features. The Kleisli category of a comonad is generally used to look at intensional semantic interpretations. In order to combine the two, we would like to form from a monad $(T, \eta, \mu)$ and a comonad $(G, \epsilon, \delta)$ in $C$ a category $\mathcal{K}(G, T)$ similar to the Kleisli categories, with objects in $C$, and with morphisms of the form $f : GA \rightarrow TB$.

In order to guarantee that we actually obtain a category, however, we need some additional conditions, adapted from [4]:

**Definition 3** Given a monad $(T, \eta, \mu)$ and a comonad $(G, \epsilon, \delta)$, a distributive law of $T$ over $G$ is a natural transformation $\sigma : GT \rightarrow TG$ such that the following four identities hold:

- $\sigma_A \circ G\eta_A = \eta_GA$
- $T\epsilon_A \circ \sigma_A = \epsilon_TA$
- $\sigma_A \circ G\mu_A = \mu_GA \circ T\sigma_A \circ \sigma_TA$
- $T\delta_A \circ \sigma_A = \sigma_GA \circ G\sigma_A \circ \delta_TA$

$\sigma$ is distributive with $\eta$ whenever the first identity holds. Similarly $\sigma$ is distributive with $\epsilon$ when the second identity holds, and so on.

For similar definitions, see [4], which related two monads, or [1], which related two comonads.
Theorem 2. Given a monad \((T, \eta, \mu)\) and a comonad \((G, \epsilon, \delta)\) in a category \(C\) plus a distributive law \(\sigma\) of \(T\) over \(G\), there is a category \(\mathcal{K}(G, T)\) defined as follows:

- The objects of \(\mathcal{K}(G, T)\) are the objects of \(C\).
- A morphism \(f : A \overset{G,T}{\to} B\) in \(\mathcal{K}(G, T)\) is a morphism \(f : GA \to TB\) in \(C\).
- For each object \(A\), the identity morphism \(\text{id}_A\) on \(A\) is \(\eta_A \circ \epsilon_A : GA \to TA\).
- For every pair of morphisms \(f : A \overset{G,T}{\to} B\) and \(g : B \overset{G,T}{\to} C\), their composition, \(g \circ f\), is \(\mu_C \circ Gf \circ \sigma_B \circ Gg \circ \delta_A\).

Proof. The proof involves some straightforward diagram chasing, shown in figures 5 and 6. Figure 5 contains a diagram showing that \(\text{id}^T_B \circ f = f \circ \text{id}^T_A = f\). The upper part of the diagram represents the expansion of \(\text{id}^T_B \circ f\), \(\mu_B \circ T\eta_B \circ T\epsilon_B \circ \sigma_B \circ Gf \circ \delta_A\), and the lower part of the diagram represents the expansion of \(f \circ \text{id}^T_A\), \(\mu_B \circ T\sigma_A \circ G\eta_A \circ T\epsilon_A \circ \delta_A\); in the center is \(f\).

Figure 6 contains a diagram showing that \(h \circ (g \circ f) = (h \circ g) \circ f\). The upper path represents the expansion of \(h \circ (g \circ f)\) and the lower path represents the expansion of \((h \circ g) \circ f\).

6.1. Examples

We now check all the possible combinations of comonads and monads from our lists of examples to see whether the distributivity condition holds. A summary of the results is in table 1 on page 26.
6.1.1. The Product comonad

A distributive law $\sigma$ of any monad $T$ over the product comonad $X$ is a natural transformation from $T_\cdot \times X$ to $T(\cdot \times X)$. For certain types of monads such a distributive law is guaranteed to exist:

**Definition 4** Let $C$ be any category with finite products. A monad $(T, \eta, \mu)$ is strong if there exists a natural transformation $r: T\_ \times _\_ \Rightarrow T(- \times _)\_\_$, called a tensorial strength, satisfying the following conditions:

- $T\pi_1 \circ \tau_{A,1} = \pi_1$.
- $\tau_{A,B} \circ (\eta_A \times \text{id}_B) = \eta_{A \times B}$.
- $\mu_{A \times B} \circ T\tau_{A,B} \circ \tau_{TA,B} = \tau_{A,B} \circ (\mu_A \times \text{id}_B)$.
- $\tau_{A \times B,C} \circ (\tau_{A,B} \times \text{id}_C) \circ \alpha_{TA,B,C} = T\alpha_{A,B,C} \circ \tau_{A,B \times C}$, where again $\alpha: \cdot \times (\cdot \times \cdot) \Rightarrow (\cdot \times \cdot) \times \cdot$.

This definition of strength is equivalent to the definition in [14], which uses a natural transformation $t: \cdot \times T_\cdot \Rightarrow T(\cdot \times \cdot)$. It is clear that given such a $t$ we can get $\tau$ with $\tau_{A,B} = T(\pi_2, \pi_1) \circ t_{B,A} \circ (\pi_2, \pi_1)$ and similarly we can get $t$ from $\tau$.

Unsurprisingly, if $(T, \eta, \mu)$ is strong then there is a distributive law $\sigma$ of $T$ over the product comonad $X$, with $\sigma_A = \tau_{A,X}$. Of course, determining the existence of strength is in general as difficult as determining the existence of a distributive law directly. For set-like categories, however, there is a simple test for strength:

**Theorem 3** Let $C$ be a set-like category, and let $(T, \eta, \mu)$ be a monad over $C$ such that for each pair of objects $A$ and $B$ the function $t_{AB} = \lambda f: A \rightarrow B \rightarrow [TA \rightarrow TB]$ is a morphism in $C$. Then $(T, \eta, \mu)$ is strong.
Figure 6: Proof of associativity of composition in $K(G, T)$
Figure 7: Requirements for $\tau$ to be a tensorial strength
Proof: Given such a category, let \( \tau_{A,B} = \text{uncurry}(t_{A,A\times B} \circ \text{curry}((\pi_2, \pi_1)) \circ (\pi_2, \pi_1) = \lambda (a,b).T(\lambda a.(a,b))a. \)

For naturality, given a function \( f : A \to B \) and a function \( g : C \to D \), we have that

\[
\tau_{C,D} \circ (Tf \times g)(a,c) = \tau_{C,D}(Tf(a),g(c))
\]

\[
= T(\lambda b.(b,g(c)))Tf(a)
\]

\[
= T(\lambda b.(b,g(c)) \circ \lambda a.f(a))a
\]

\[
= T(\lambda a.(a,c))a
\]

\[
= T(f \times g) \circ \tau_{A,B}(a,c)
\]

The other conditions can be shown in a similarly straightforward manner. \( \Box \)

All of the monad examples given are strong in Cont [14]. The resulting \( \sigma \) turn out to be:

Lifting: \( \sigma_A : LA \times X \to L(A \times X) \) is defined by \( \sigma_A(\bot, x) = \bot \) and for \( a \in A \), \( \sigma_A(a,x) = (a,x) \).

Separated Sums: \( \sigma_A : (A+E) \times X \to (A \times X) + E \) is defined by \( \sigma_A(\iota_1(a),x) = \iota_1((a,x),x) \), \( \sigma_A(\iota_2(e),x) = \iota_2(e) \), and \( \sigma_A(\bot, x) = \bot \).

Products: \( \sigma_A : (A \times X') \times X \to (A \times X) \times X' \) is defined by \( \sigma_A((a,x'),x) = ((a,x),x') \), where \( X \) denotes the object from the product comonad, and \( X' \) denotes the object from the product monad.

The product monad is strong in any category with products, where \( \sigma_A = (\pi_1 \times \text{id}_X, \pi_2 \circ \pi_1) \) is the natural isomorphism from \((A \times X') \times X \) to \((A \times X) \times X' \).

Side effects: \( \sigma_A : [S \to (A \times S)] \times X \to [S \to ((A \times X) \times S)] \) is defined by \( \sigma_A(s,(a,z)) = \lambda s.((a,z),z) \).

More generally, in a cartesian closed category we have \( \sigma_A = \text{curry}(\beta_{A,S,X} \circ (\text{app}_{SA,S} \times \text{id}_X) \circ \beta_{SA,X,S}) \), where \( \beta_{A,B,C} \) is the natural isomorphism from \((A \times B) \times C \) to \((A \times C) \times B \).

Power Sets: (In Set) \( \sigma_A : PA \times X \to P(A \times X) \) is defined by \( \sigma_A(u,x) = \{(a,x) \mid x \in u\} \).

Power Domains: \( \sigma_{A}^\dagger : P^I A \times X \to P^I(A \times X) \) (for \( \dagger \in \{\delta, \preceq, \} \)) is defined by

\[
\sigma_{A}^\dagger(s,x) = \bigcup_{u \in S} \{w \mid \forall a \in u. \exists k_a \leq (a,x).w \subseteq \{k_u \mid a \in u\} \}
\]

For compact elements \( (u,k) \), \( \sigma_{A}^\dagger(u,k) = \{(a,k) \mid a \in u\} \).

Exponentiation: \( \sigma_A : [V \to A] \times X \to [V \to (A \times X)] \) is defined by \( \sigma_A(\lambda v.a,v,x) = \lambda v.\langle a,v,x \rangle \). A similar definition holds for strict paths.

For all cartesian closed categories the exponentiation monad is strong, with

\[
\sigma_A = \text{curry}((\text{app}_{A,V} \times \text{id}_X) \circ \beta_{VA,X,V})
\]

6.1.2. Exponentiation as a monad

A comonad \( \langle G, \epsilon, \delta \rangle \) is strong if there exists a natural transformation \( \tau : G \times \_ \to G(\_ \times \_ \) that satisfies analogous conditions to the ones required for a strong monad (see figure 8). For all such comonads, there exists a distributive law \( \sigma \) of the exponentiation monad \( V^{-} \) over \( G \), where \( \sigma_A = \text{curry}(G\text{app}_{A,A} \circ \tau_{V^{-},A,X}) \).

The proof is analogous to the proof for the product monad.
As for strong monads, for set-like categories whenever the function $g_{A,B} = \lambda f . Gf$ is a morphism in that category, then $G$ is strong. The resulting $\sigma$ will be

$$\text{curry}(\text{uncurry}(g_{v-A,A} \circ \text{curry}(\text{app}_{X,A} \circ (\pi_2, \pi_1))) \circ (\pi_2, \pi_1))$$

i.e. for $g \in G[X \rightarrow A]$, $\sigma(g) = \lambda x . G(\lambda a . a(x))g$.

The comonad $\langle V, \text{val}, \text{pre} \rangle$ is strong in both $\text{Cont}$ and $\text{dl}$, with the resulting $\sigma$ being defined by $\sigma(\lambda v . \lambda x . a_{v,x}) = \lambda x . \lambda v . a_{v,x}$. The comonad $\langle V_S, \text{val}, \text{pre} \rangle$, however, is not strong in $\text{Cont}$, for the function $\lambda f . Vsf$ is not monotone when functions are ordered pointwise (and given a tensorial strength $\tau$, we can derive as a morphism $\lambda f . Vsf$, with $\lambda f . Vsf = \text{curry}(V_S\text{app}_{A,B} \circ V_S(\tau_2, \pi_1) \circ \tau_{A,B \rightarrow B} \circ (\pi_2, \pi_1)))$. The product comonad is strong and the $\sigma$ resulting from this construction is identical to the $\sigma$ mentioned in the previous section.
6.1.3. Product as a monad

For all comonads \( \langle G, \varepsilon, \delta \rangle \) there is a distributive law \( \sigma_A = (\pi_1 \circ \varepsilon, \pi_2 \circ \delta) \) of the product monad \( X \) over \( G \). For example there is a distributive law \( \sigma \) of \( X \) over \( V \), where for standard cartesian closed categories

\[
\sigma_A = (\text{curry}(\pi_1 \circ \text{app}_{V(A \times X), V}), \pi_2 \circ \text{app}_{V(A \times X), V} \circ (\text{id}_{V(A \times X)}, \text{ev}_{V(A \times X)}))
\]

or, in \( \text{Cont} \),

\[
\sigma_A(\lambda v. (a_v, x_v)) = (\lambda v. a_v, x_v)
\]

For the strictly increasing paths comonad \( \sigma \) is the same except that duplicates are removed.

6.1.4. Lifting with the path comonads

If we look at the lifting monad \( \langle L, \text{up}, \text{down} \rangle \) with the path comonad \( \langle V, \text{val}, \text{pre} \rangle \), using the monoid \( \text{VNat} \) (\( e = \omega \) and \( m = \min \)) in the category \( \text{Cont} \), we see that an element of \( VLA \) can have one of three forms: it can be a path \( a \in VA \), the element \( \bot^\omega \), or of the form \( \bot^n a \) for some path \( a \in VA \) and some \( n > 0 \). An element of \( LVA \) is either a path \( a \in VA \) or \( \bot \).

In order for a natural transformation \( \sigma \) to be distributive with \( \text{up} \) we must have that if \( a \in VA \), \( \sigma_A(a) = \sigma_A(Va)(a) = \text{up}_V(a) = a \). There is no value, however, that can be assigned to \( \sigma_A(\bot a) \) that satisfies all distributivity requirements without violating monotonicity or naturality. For example, let us look at the domain \( A = \text{VNat} \) and the element \( \bot^\omega \). Since \( \sigma_{\text{VNat}}(\bot^\omega) = 0^\omega \) and \( \bot^\omega \leq 0^\omega \), by monotonicity of \( \sigma_{\text{VNat}} \) we must have that \( \sigma_{\text{VNat}}(\bot^\omega) \leq 0^\omega \) which in \( L\text{VNat} \) means that either \( \sigma_{\text{VNat}}(\bot^\omega) = 0^\omega \) or \( \sigma_{\text{VNat}}(\bot^\omega) = \bot \).

If we let \( \sigma_{\text{VNat}}(\bot^\omega) = \bot \), then \( \sigma \) is not distributive with \( \text{val} \), since \( L\text{val}_{\text{VNat}}(\sigma_{\text{VNat}}(\bot^\omega)) = L\text{val}_{\text{VNat}}(\bot) = \bot \) and \( \text{val}_{\text{VNat}}(\bot^\omega) = 0 \).

So we must have that \( \sigma(\bot^\omega) = 0^\omega \). Let \( f : \text{VNat} \rightarrow \text{VNat} \) be the constant \( 1 \) function. Therefore by naturality of \( \sigma \) we have that

\[
\sigma_{\text{VNat}}(\bot) = \sigma_{\text{VNat}}(Lf(\bot^\omega)) \\
= Lf(\sigma_{\text{VNat}}(\bot^\omega)) \\
= 1^\omega
\]

However we already know that \( \sigma_{\text{VNat}}(1^\omega) = 01^\omega \). Thus we have that \( \bot^\omega \leq 01^\omega \), but \( \sigma_{\text{VNat}}(\bot^\omega) = 1^\omega > \sigma_{\text{VNat}}(01^\omega) \). So \( \sigma \) fails to be monotone.

This proof generalizes to any non-trivial monoid \( V \) but not to the strictly increasing path comonad or to exponentiation in the category \( \text{dIL} \). In fact the natural transformation given by

\[
\sigma_A(a) = \begin{cases} 
\bot & a = \bot^\omega \\
\text{a'} & a = \bot^n a' \text{ for some } n > 0 \\
a & \text{otherwise}
\end{cases}
\]

19
is a distributive law of \( L \) over \( V (V = V_{\text{Nat}}) \) in the category \( \text{dI} \) (\( \sigma_A \) is stable for all \( \text{dI} \) domains \( A \)). When we limit \( n \) to 1, \( \sigma \) is a distributive law of \( L \) over \( V_5 \).

### 6.1.5. Coproducts

There is no distributive law \( \sigma \) of the separated sum monad \( E \) over the exponentiation comonad \( V \) in the category \( \text{Cont} \), for reasons similar to the ones above showing that there is no distributive law of the lifting monad \( L \) over \( V \). Again there is again a distributive law \( \sigma \) of \( E \) over \( V_5 \) and a distributive law \( \sigma \) of \( E \) over \( V \) in \( \text{dI} \) (for the monoid \( V_{\text{Nat}} \)), namely

\[
\sigma_A(a) = \begin{cases} 
\bot & a = \bot \\
1_1((a_i)_i)^{\omega} & \exists n \geq 0. a = 1^n_1((a_i)_i)^{\omega} \\
\varepsilon(a) & \text{otherwise}
\end{cases}
\]

This particular function depends not only on the structure of the comonad, but also on the structure of the monoid and thus does not generalize easily.

For categories with coproducts and one extra (somewhat strong) condition we do get distributivity:

**Theorem 4** Let \( C \) be a category with binary coproducts, \( \langle G, \varepsilon, \delta \rangle \) be any comonad over \( C \), and \( E \) be some object. Suppose that for each object \( A \), the object \( GA + GE \) is naturally isomorphic to \( G(A + E) \), with one half of the isomorphism being \( \phi_A = [G_{1A}, G_{2E}] \). Then there is a distributive law \( \sigma \) of the coproduct monad \( E \) over \( G \), namely

\[
\sigma_A = (id_A + \varepsilon_E) \circ \phi_A^{-1}.
\]

Although the above condition does not hold with the comonad \( V \) in \( \text{Cont} \), it does hold for \( V \) in the category \( \text{Cont}_5 \) of domains and strict continuous functions (using coalesced sums). For set-like categories, the above condition will hold primarily when the structure of \( G(A + E) \) uniformly consists of either elements of \( A \) or elements of \( E \).

### 6.1.6. Power Sets and Power Domains with the Path comonads

In the category \( \text{Set} \) and for all monoids \( V \), there is a distributive law \( \sigma \) of the power set monad \( P \) over the exponentiation comonad \( V \), with \( \sigma_A : [V \rightarrow PA] \rightarrow P[V \rightarrow A] \) defined by

\[
\sigma_A(X) = \{ a \in VA \mid \forall v \in V. a(v) \in X(v) \}
\]

The equivalent function in \( \text{Cont} \) for the Hoare or Smyth power domains, and in \( \text{Bif} \) for the Plotkin power domain is

\[
\sigma_A(s) = \{ w \in P^*_{\text{fin}}(K(VA)) \mid \forall v \in V. \{ a(v) \mid a \in w \} \in s(v) \}
\]

For the Hoare or Smyth power domains, however, \( \sigma \) is not natural, and for the Plotkin power domain, it is not even monotone. In fact we can show that for all three power domains any choice of \( \sigma \) will fail at least one of the conditions needed for distributivity.
Theorem 5 In the category \textsf{Cont}, let \( V \) be the monoid \( \text{VNat} \) with unit \( \omega \) and binary operator \( \text{min} \). Then there is no distributive law of the Hoare power domain monad \( \langle P^b, \bot \rangle \) over the path comonad \( \langle V, \text{val}, \text{pre} \rangle \).

Proof. Assume that there exists an indexed collection of morphisms \( \sigma \) such that for each domain \( A \), \( \sigma_A : VP^bA \to P^bVA \) and that \( \sigma \) is distributive with \( \bot^b \), i.e. for all objects \( A \), \( \sigma_A \circ V\bot \circ \bot_A = \bot \circ \bot_A \). We will show that there are domains \( A \) and \( B \), an \( s \in VP^bA \), and a continuous function \( f : A \to B \) such that \( \sigma \) is not natural.

Let \( A \) be the flat truth value domain \( \text{Bool} \). Let \( B = 3 \), the domain of three points, \( \{0, 1, 2\} \), with \( 0 < 1 < 2 \). Let \( f : \text{Bool} \to 3 \) be the function with \( f(\bot) = 0, f(tt) = 1, f(\text{ff}) = 2 \). It is clear that \( f \) is monotone on a finite domain and therefore is also continuous. Let \( s = (\{tt\})(\{tt, ff\})^\omega \). Then

\[
\begin{align*}
\sigma_B \circ VP^bfs &= \sigma_B(VP^bf((\{tt\})(\{tt, ff\})^\omega)) \\
&= \sigma_B(P^bf((\{tt\})(\{tt, ff\})^\omega)) \\
&= \sigma_B((\{f(tt)\})(\{f(tt), f(ff)\})^\omega) \quad \text{using the definition of } P^b \\
&= \sigma_B((\{1\})(\{1, 2\})^\omega) \\
&= \{2\} = \{1, 2\} \\
&= \{2\} \\
&= \{12^\omega\} \\
&\quad \text{by assumption}
\end{align*}
\]

Now suppose that for some \( s \in P^bV\text{Bool} \) we have that \( P^bVf(s) = \{12^\omega\} \). Given that all elements of \( V3 \) are compact, we have that

\[
\{12^\omega\} = P^bVf(s) = \bigcup_{u \in s} \{Vf(x) \mid x \in u\} = \{w \in P^b_{\text{fin}}(V3) \mid \exists u \in s. w \subseteq^+ \{Vf(x) \mid x \in u\}\}
\]

Thus we know that for every \( u \in s \), \( \{Vf(x) \mid x \in u\} \subseteq^+ \{12^\omega\} \). This means that for all \( b \in u \), \( Vf(b) \leq 12^\omega \). We also know that since \( \{12^\omega\} \subseteq^+ \{12^\omega\} = P^bVf(s) \), there must be an \( u \) such that \( \{12^\omega\} \subseteq^+ \{Vf(x) \mid x \in u\} \), which means that there exists some \( b \in u \) with \( 12^\omega \leq Vf(b) \). Since we also have that \( Vf(b) \leq 12^\omega \), \( Vf(b) \) must equal \( 12^\omega \).

However the only sequence \( b \) in \( \text{Bool} \) such that \( Vf(b) = 12^\omega \) is \( b = tt \text{ ff} \), which is not an element of \( V\text{Bool} \). Thus there is no value \( \sigma_{\text{Bool}} \) can give to \( s \) to make \( \sigma_B \circ VP^bf(s) \) equal to \( P^bVf(\sigma_{\text{Bool}}(s)) \), so \( \sigma \) is not natural. \( \Box \)

Since the proof used valid \( V_S \) paths and did not use specific details of the ordering of the paths, we also have shown that

Corollary 6 In the category \textsf{Cont}, there is no distributive law of the Hoare monad \( \langle P^b, \bot \rangle \) over the strict paths comonad \( \langle V_S, \text{val}, \text{pre} \rangle \).
Also, by a straightforward extension of the proof given above, we can show that there is no distributive law for any monoid domain \( V \) that contains two points \( v_1 \) and \( v_2 \) with \( v_1 < V v_2 \), i.e. any non-trivial monoid domain.

**Theorem 7** In the category \textbf{Cont}, let \( V \) be the monoid \( V\text{Nat} \) as in Theorem 5. Then there is no distributive law of the Smyth power domain \( (P^2, [\bot]^{\downarrow}, [1]^{\downarrow}) \) over the path comonad \( (V, \text{val}, \text{pre}) \).

**Proof.** Assume that there exists an indexed collection of functions \( \sigma \) such that for each domain \( A \), \( \sigma_A : V P^2 A \rightarrow P^2 VA \). We will show that the \( \sigma \)'s cannot simultaneously satisfy monotonicity, naturality, and distributivity with \( [\bot]^{\downarrow} \).

Let \( 2 \) be the two point set \( \{ \bot, T \} \) with \( \bot < T \). Define \( f_0, f_1 : \text{Bool} \rightarrow 2 \) as follows:

\[
\begin{align*}
f_0(\bot) &= \bot \\
f_0(TT) &= T \\
f_0(FF) &= T \\
f_1(\bot) &= \bot \\
f_1(TT) &= \bot \\
f_1(FF) &= T 
\end{align*}
\]

Let \( s = (\{ TT, FF \})(\{ FF \}) \in VP^2 \text{Bool} \). If we assume that \( \sigma \) is distributive with \([\bot]^{\downarrow}\), i.e. for all objects \( A \), \( \sigma_A \circ V [\bot]^{\downarrow} = [\bot]^{\downarrow} V A \), we get that

\[
\begin{align*}
\sigma_2 \circ VP^2 f_0 s &= \sigma_2(VP^2 f_0((\{ TT, FF \})(\{ FF \}))^\omega) \\
&= \sigma_2((\{ T \})^\omega) \\
&= \sigma_2([\{ T \}]^{\downarrow} [\{ T \}]^{\downarrow}) \\
&= [\{ T \}]^{\downarrow} [\{ T \}]^{\downarrow} \\
&= \{ \{ T \} \}
\end{align*}
\]

by assumption

Given any \( s \in P^2 V \text{Bool} \), we see that

\[
\begin{align*}
P^2 V f_0(s) &= \bigcup_{u \in V} \bigcup_{x \in u} V f(x) [\{ T \}]^2 \\
&= \bigcup_{u \in V} [\{ V f(t) \mid t \in u \} \text{ since all elements of } V2 \text{ are compact}
\end{align*}
\]

Thus in order for \( P^2 V f_0(s) \) to be equal to \( \{ \{ T \} \} \) there must be a \( u \in s \) with \( \{ T \} \) \( \subseteq \{ V f(x) \mid x \in u \} \), i.e. for all \( t \in u \), \( T \leq V f(t) \). Since \( T \) is the maximum element of \( V2 \), this is the same as saying that for all \( t \in u \), \( V f(t) = T \).

Now the elements of \( V \text{Bool} \) are all of the form \( \bot \), \( \bot TT \), and \( \bot FF \), where \( n \) is a non-negative integer. If we apply \( V f_0 \) to all of these we find that

\[
\begin{align*}
V f_0(\bot) &= \bot \\
V f_0(\bot TT) &= \bot TT \\
V f_0(\bot FF) &= \bot TT
\end{align*}
\]

22
Therefore for all $s$ with $P^4 Vf_0(s) = \{ \top^\omega \}$, either $\{ \top \top^\omega \}$, $\{ \ell \ell^\omega \}$, or $\{ \ell \ell \}$ must be in $s$. Since those three sets are maximal in the Smyth preorder, $s$ must therefore be either $\{ \top \top \}$, $\{ \ell \ell \}$, or $\{ \ell \ell \}$. 

If we assume $s$ is natural we have that $P^4 Vf_0(\sigma_{\text{Bool}}(s)) = \sigma_2(VP^4 f_0(s))$, but we know that $\sigma_2(VP^4 f_0(s))$ equals $|\{ \top^\omega \}$. Therefore $\sigma_{\text{Bool}}(s)$ must be either $\{ \top \top \}$, $\{ \ell \ell \}$, or $\{ \ell \ell \}$.

Now with $f_1$ we see that

$$\sigma_2 \circ VP^4 f_1(s) = \sigma_2(VP^4 f_1(|\{ \top \top \} | \{ \ell \ell \}))(\{ \ell \ell \}^\omega) \quad = \sigma_2(|\{ \top \top \} | \{ \top \} | \{ \ell \ell \}) \quad = \sigma_2(|\{ \top \top \} | \{ \top \} | \{ \ell \ell \} | \{ \ell \ell \}^\omega) \quad = \{ \top \top \}$$

To get naturality with $f_1$ we need $P^4 Vf_1(\sigma_{\text{Bool}}(s))$ to equal $|\{ \top^\omega \}$ for one of the possible values for $\sigma_{\text{Bool}}(s)$. However

$$P^4 Vf_1(|\{ \top \top \}) = |\{ \top \} \quad Vf_1(|\{ \ell \ell \}) = |\{ \top \} \quad \text{and} \quad P^4 Vf_1(|\{ \ell \ell \} \top \top \}) = |\{ \top \} \top \top \}$$

none of which are equal to $|\{ \top \top \}$. \square

**Corollary 8** In the category $\text{Cont}$, there is no distributive law of the Smyth monad $(P^4, |\{ \top \}| \top \top \}$ over the strict paths comonad $(V_S, \text{vval}, \text{vpre})$.

**Proof.** The proof for theorem 7 can be adapted as a proof of this corollary, since the proof used only strict paths and while the proof did use the path ordering non-trivially, all statements using that ordering (namely that $\top^\omega$, $\top \top^\omega$, and $\ell \ell^\omega$ are maximal in their respective domains), hold for the strict path ordering as well. \square

Even though there is no distributive law $\sigma$, for the Hoare and Smyth domains there are lax distributive laws $\text{c}$ and $\text{c}$, in the sense that for all morphisms $f : A \rightarrow B, s \in VP^4 A$ and $s' \in VP^4 A, \text{c} \circ VP^4 f(s) \leq P^4 Vf \circ \text{c}(s)$ and $\text{c} \circ VP^4 f(s') \leq P^4 Vf \circ \text{c}(s')$ and $\text{c}$ satisfies all the other requirements. Here $\text{c}$, defined as

$$\text{c}(s) = \{ w \in P_{\text{max}}(VA) | \forall v \in V, \{ a(v) | a \in w \} \in s(v) \} \quad \text{(1 \in \{ \top, \bot \})}$$

is the generalization of the distributive law for sets and exponentiation mentioned at the beginning of this section. Unfortunately, the inequality in the naturality of $\text{c}$ translates to an inequality in the associativity law for the double Kleisli construction, so we cannot form a Kleisli category this way.

**Theorem 9** In the category $\text{Bif}$, let $V$ be the monoid $V\text{Nat}$ as in Theorem 5. Then there is no distributive law of the Plotkin monad $(P^4, |\{ \top \}| \top \top \}$ over the path comonad $(V, \text{val}, \text{pre})$.

**Proof.** Assume that there exists a natural transformation $\sigma : VP^4 \Rightarrow P^4 V$ and that $\sigma$ is distributive with both $|\{ \top \}| \top \top \}$ and $\text{val}$. We will show that there exists a bifinite domain $A$ and elements $s_1$ and $s_2$ in $VP^4 A$ such that $s_1 \leq s_2$ but $\sigma_A(s_1) \neq \sigma_A(s_2)$. 

23
Let \( A \) be the bifinite domain consisting of the set \( \{ \bot, a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2 \} \) with the ordering \( \leq^A \) as shown in figure 9. Let \( s_1 = (\{a_1, b_1\})(\{a_2, b_2\})^\omega \), and \( s_2 = (\{c_1, d_1\})(\{c_2, d_2\})^\omega \). Clearly \( s_1 \leq s_2 \).

We will show that \( \sigma_A(s_1) \not\leq \sigma_A(s_2) \).

Let \( f : A \rightarrow V_{\text{Nat}} \) be defined as the least continuous function such that:

\[
\begin{align*}
f(a_i) &= f(b_i) = i \quad i = 1 \text{ or } 2 \\
f(c_i) &= f(d_i) = i + 2 \quad i = 1 \text{ or } 2
\end{align*}
\]

Figure 10 shows the values of \( f \) for each value in \( A \).

We then have that:

\[
\sigma_{V_{\text{Nat}}} \circ V^Pf(s_1) = \sigma_{V_{\text{Nat}}}(V^Pf((\{a_1, b_1\})(\{a_2, b_2\})^\omega))
\]

\[
= \sigma_{V_{\text{Nat}}}((\{1\})(\{2\})^\omega)
\]

\[
= \sigma_{V_{\text{Nat}}}((1^1)(2^1))
\]

\[
= 12^\omega 0^0 V_{\text{Nat}}
\]

assuming distributivity with \( \|\cdot\| \).

\[
= \{12^\omega\}
\]

Suppose there exists an \( s \in P^VA \) such that \( P^Vf(s) = \{12^\omega\} \). Since the range of \( Vf \) consists only of
compact elements, we have that

\[
\{12^w\} = P^k Vf (s) = \bigcup_{a \in u} \{Vf (a) \mid a \in u\}
\]

so for each \( u \in s \) we must have that \( \{Vf (a) \mid a \in u\} \subseteq \{12^w\} \), i.e. for all \( a \in u, a \leq 12^w \). Since the \( a_i \)'s and the \( b_i \)'s are incomparable with each other in \( A \), this means that there exists a \( u_0 \subseteq \{a_1 a_2^w, b_1 b_2^w\} \) with \( u \subseteq u_0 \). This implies that \( s = u_0 \). By the assumption of naturality for \( \sigma \) we have \( \{12^w\} = \sigma \text{Nat} \circ VP^k f s_1 = P^k Vf \circ \sigma_{As1} \) so \( \sigma_{As1} = u_0 \). If we then use the assumption of distributivity with \( \text{val} \) we find that

\[
P^k \text{val} \text{Nat}(u_0) = P^k \text{val} \text{Nat} \circ \sigma_{As1}
\]

\[
= \text{val}_t \text{Nat}s_1
\]

\[
= \{a_2, b_2\}
\]

The only value of \( u_0 \) that satisfies the above equations is \( u_0 = \{a_1 a_2^w, b_1 b_2^w\} = \sigma_{As1} \).

By a similar argument we can also show that \( \sigma_{As2} = \{c_1 c_2^w, d_1 d_2^w\} \). However, \( \{a_1 a_2^w, b_1 b_2^w\} \) and \( \{c_1 c_2^w, d_1 d_2^w\} \) are incomparable, so \( \sigma_A \) is not monotone.

This proof generalizes to any monoid with two distinct and related elements; however, unlike the proofs for \( P^k \) and \( P^t \), it does not apply to the strict path comonad \( V_S \), since \( s_1 \) and \( s_2 \) are incomparable in \( V_S A \). Moreover, the \( \sigma \) given at the beginning of the section, is also not a distributive law over \( V_S \) since it not distributive with \( \{\cdot\}^3 \). We do not currently know if such a distributive law exists.

6.1.7. Side Effects

If we have a comonad \( \langle G, \varepsilon, \delta \rangle \) that is both strong and satisfies the equation

\[
\tau_{A,B} \circ (\text{id}_G \times \varepsilon_B) \circ (G \pi_1, G \pi_2) = \text{id}_{GA \times B}
\]

then there will be a distributive law \( \sigma \) of the side effect monad \( \langle S, \eta, \mu \rangle \) over \( G \), with

\[
\sigma_A = \text{curry}(\text{id}_G \times \varepsilon_S) \circ (G \pi_1, G \pi_2) \circ \text{Gap}_{S, A \times S} \circ \tau_{AS, S}
\]

This condition is rather stringent, essentially requiring that information lost when applying \( \varepsilon \) to \( GB \) is "stored" in \( GA \) so that it can later be recovered by \( \tau \). Even though this condition holds for the product comonad, for the exponentiation comonad (and its variant, the strict path comonad), it is equivalent to requiring that the monoid \( V \) is isomorphic to \( 1 \). In fact the above definition for \( \sigma \) which in \( \text{Cont} \) turns out to be

\[
\sigma_A(\lambda \nu. \lambda s. (a_\nu(s), z_\nu(s))) = \lambda \nu. \lambda s. (a_\nu(s), z_\nu(s))
\]

does not distribute with \( \mu \) for non-trivial monoids. It is highly unlikely that any distributive law exists for \( S \) over any non-trivial exponentiation comonads.

7. Non-distributive double Kleisli categories

An alternative method to using both the comonad and the monad simultaneously is to lift the comonad into the Kleisli category of the monad. One way is to let \( GA = GA \) and then coerce the morphism part of the
Figure 11: The condition $\tau_{A,B} \circ (\text{id}_{GA} \times \epsilon_B) \circ (G \pi_1, G \pi_2) = \text{id}_{G(A \times B)}$

<table>
<thead>
<tr>
<th>Monads</th>
<th>Products</th>
<th>General Path in Comonad</th>
<th>Paths in $\text{Cont}$ $(V = V\text{Nat})$</th>
<th>Strict Paths or Paths in $\text{dI}$</th>
<th>Other Comonad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lifting</td>
<td>yes</td>
<td>--</td>
<td>no</td>
<td>yes</td>
<td>--</td>
</tr>
<tr>
<td>Exceptions</td>
<td>yes</td>
<td>see sec. 6.1.5</td>
<td>no</td>
<td>yes</td>
<td>if $G(A + E) \cong GA + GE$</td>
</tr>
<tr>
<td>Products</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Side Effects</td>
<td>yes</td>
<td>see section 6.1.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power sets</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P^r$</td>
<td>yes</td>
<td>--</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>$P^l$</td>
<td>yes</td>
<td>--</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>$P^h$</td>
<td>yes</td>
<td>--</td>
<td>no</td>
<td>see sec. 6.1.6</td>
<td></td>
</tr>
<tr>
<td>Exponentiation</td>
<td>yes</td>
<td>if strong</td>
<td>yes</td>
<td>$\text{dI}$ only</td>
<td>if strong</td>
</tr>
<tr>
<td>Other monad</td>
<td>if strong</td>
<td>--</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Distributive laws of various monads over comonads

Since there are several combinations of monad and comonads that do not have distributive laws, we need to find a way of combining them without one. One possibility is to lift the comonad using $G'A = GTA:

**Theorem 10** Let $(T, \eta, \cdot^*)$ be a Kleisli triple and $(G, \epsilon, \cdot^*)$ be a Kleisli cotriple on a category $\mathcal{C}$. Then there

\[
\begin{align*}
\hat{G}A &= GA \\
\check{G}f &= \sigma_A \circ Gf : GA \rightarrow TGA \\
\hat{A} &= \eta_A \circ \epsilon_A : GA \rightarrow TA \\
\check{A} &= \eta_{G'A} \circ \delta_A : GA \rightarrow TG^2A \\
\end{align*}

\[
\begin{align*}
\hat{T}A &= TA \\
\check{T}f &= T \circ \sigma_A : GTA \rightarrow TA \\
\hat{\eta}_A &= \eta_A \circ \epsilon_A : GA \rightarrow TA \\
\check{\eta}_A &= \eta_{G'A} \circ \delta_A : GT^2A \rightarrow TA \\
\end{align*}

Figure 12: Lifting the comonad or monad using a distributive law
exists a Kleisli cotriple \((G', \epsilon', \eta')\) on the Kleisli category \(K(T)\) defined as follows:

- For all objects \(A\), \(G'A\) is the object \(GTA\).
- For all objects \(A\), \(\epsilon'_A : GTA \to TA\) is \(\epsilon_TA\).
- For all morphisms \(f : GTA \to TB\), \(f^* : GTA \to TGTB\) is the morphism \(\eta_{GTB} \circ f^*\).

**Proof.** The proof involves using straightforward diagram chasing to show that the three conditions specified in section 4 are satisfied. Figure 13 contains a diagram proving that for all \(f : GTA \to TB\), \(\epsilon'_B \circ f^* = f\).

Given the above, we can construct a lifted comonad \((G', \epsilon', \eta')\) from the lifted Kleisli cotriple \((G', \epsilon', \eta')\). It can easily be shown by simplifying the definition given in section 4 that the resulting comonad is defined as follows:

- For all objects \(A\), \(G'A\) is the object \(GTA\)
- For all \(f : A \to TB\), \(G'f : GTA \to TGTB\) is \(\eta_{GTB} \circ G(\mu_B \circ Tf)\)
- For all objects \(A\), \(\epsilon'_A : GTA \to TA\) is \(\epsilon_TA\)
- For all objects \(A\), \(\delta'_A : GTA \to TGTGA\) (\(= T(G'\lambda A)\)) is \(\eta_{GTGTA} \circ G\eta_{GTA} \circ \delta_TA\)

We can now construct the Kleisli category \(K(G')\) from \(K(T)\). Again it is straightforward to show that the resulting category has the form

- Objects of \(K(G')\) are the same as for \(C\).
Figure 14: Proof that $g \circ Gf \circ \delta_A' = g \circ Gf \circ \delta_{TA}$. The upper part of the diagram represents $g \circ (Gf \circ \delta_A') = \mu_C \circ Tg \circ \mu_{GTB} \circ T\eta_{GTB} \circ TG\mu_B \circ TGf \circ \eta_{GTGTA} \circ G\eta_{GTA} \circ \delta_{TA}$.

- A morphism $f : A \rightarrow B$ is a morphism $f : GTA \rightarrow TB$ in $C$.
- For all objects $A$, the identity morphism on $A$, $id_{A'}$, is $\epsilon_A' = \epsilon_{TA}$.
- For all $f : A \rightarrow B$ and $g : B \rightarrow C$, their composition $g \circ f$ is $g \circ Gf \circ \delta_A' = g \circ Gf \circ \delta_{TA}$.

Figure 14 contains a diagram proving that $g \circ Gf \circ \delta_A' = g \circ Gf \circ \delta_{TA}$.

Using similar definitions, we can also construct a lifted monad $(T'A, \eta'_A, \mu'_A)$ over the Kleisli category $K(G)$ and then form the Kleisli category $K(T')$. It is straightforward to show that they have the following forms:

- For all objects $A$, $T'A$ is the object $TGA$
Fig 15: Typing rules for simple programming language

- For all \( f : GA \to B, T'f : GTGA \to TGB \) is \( T(Gf \circ \delta_A) \circ \epsilon_{TGA} \)
- For all objects \( A, \eta_A' : GA \to TGA \) is \( \eta_{GA} \)
- For all objects \( A, \mu_A' : GTGTGA(= G(T')^2A) \to TGA \) is \( \mu_{GA} \circ T\epsilon_{TGA} \circ \epsilon_{TGTA} \)

\[ \begin{array}{c}
\vdash \rho \text{ type} \\
\vdash \tau \text{ type} \\
\vdash T\tau \text{ type} \\
\vdash \tau \text{ type} \\
x : \tau \vdash x : \tau \\
x : \tau \vdash e : \text{dom}(p) \\
x : \tau \vdash e : \text{ran}(p) \\
x : \tau \vdash e : \tau' \\
x : \tau \vdash \mu(e) : \tau' \\
x : \tau \vdash [e] : T\tau' \\
x : \tau \vdash e_1 : \tau_1 \\
x_1 : \tau_1 \vdash e_2 : \tau_2 \\
x : \tau \vdash \text{let } x_1 = e_1 \text{ in } e_2 : \tau_2
\end{array} \]

8. Examples with monadic languages

To apply some of these constructions, let us look at the "simple programming language" described in [14], with the following syntax:

\[
\begin{align*}
\tau &::= \rho \mid T\rho \\
e &::= x \mid p(e) \mid [e] \mid \mu(e) \mid \text{let } x = e_1 \text{ in } e_2
\end{align*}
\]

Here \( \rho \) ranges over a set of atomic types, \( x \) over a set of variables, and \( p \) over a set of constant function symbols. Note that expressions contain exactly one free variable.

The typing rules for this language are listed in figure 15, consisting of definitions for the judgements \( \vdash \tau \text{ type} \) (denoting a valid type) and \( x : \tau \vdash e : \tau' \). For each constant \( p \) we assume a given domain type \( \text{dom}(p) \) and a range type \( \text{ran}(p) \).
\[
\begin{align*}
\frac{x : \tau \vdash e : \tau'}{x : \tau \vdash e \equiv_{\tau'} e} & \quad \frac{x : \tau \vdash e_1 \equiv_{\tau} e_2}{x : \tau \vdash e_1 \equiv_{\tau} e_3} \quad \frac{x : \tau \vdash e_2 \equiv_{\tau} e_3}{x : \tau \vdash e_1 \equiv_{\tau} e_3} \\
\frac{x : \tau \vdash e_1 \equiv_{\tau} e_2}{x : \tau \vdash e_2 \equiv_{\tau} e_3} & \quad \frac{x : \tau \vdash p(e_1) \equiv_{\text{dom}(p)} p(e_2)}{x : \tau \vdash p(e_1) \equiv_{\text{ran}(p)} p(e_2)}
\end{align*}
\]  

\[
\begin{align*}
\frac{x : \tau \vdash [e_1] \equiv_{\tau} [e_2]}{x : \tau \vdash [e_1] \equiv_{\tau} [e_2]} & \quad \frac{x : \tau \vdash \mu(e_1) \equiv_{\tau} \mu(e_2)}{x : \tau \vdash \mu(e_1) \equiv_{\tau} \mu(e_2)}
\end{align*}
\]  

\[
\begin{align*}
\frac{x : \tau \vdash e : \tau'}{x : \tau \vdash \mu([e]) \equiv_{\tau} e} & \quad \frac{x : \tau \vdash \text{let } x_1 \leftarrow e \text{ in } e_2 \equiv_{\tau} \text{let } x_1 \leftarrow e' \text{ in } e'_2}{x : \tau \vdash \text{let } x_1 \leftarrow e \text{ in } e_2 \equiv_{\tau} \text{let } x_1 \leftarrow e' \text{ in } e'_2}
\end{align*}
\]  

\[
\begin{align*}
\frac{x : \tau \vdash \text{let } x_1 \leftarrow e \text{ in } x_1 \equiv_{\tau} e}{x : \tau \vdash \text{let } x_1 \leftarrow e \text{ in } x_1 \equiv_{\tau} e} & \quad \frac{x : \tau \vdash p(e) \equiv_{\text{ran}(p)} \text{let } x_1 \leftarrow e \text{ in } p(x_1)}{x : \tau \vdash p(e) \equiv_{\text{dom}(p)} \text{let } x_1 \leftarrow e \text{ in } p(x_1)}
\end{align*}
\]  

\[
\begin{align*}
\frac{x : \tau \vdash \text{let } x_1 \leftarrow e_1 \text{ in } x_1 \equiv_{\tau} e_2 \equiv_{\tau} x_2 \equiv_{\tau} e_3}{x : \tau \vdash \text{let } x_1 \leftarrow e_1 \text{ in } x_1 \equiv_{\tau} e_2 \equiv_{\tau} x_2 \equiv_{\tau} e_3}
\end{align*}
\]  

Figure 16: Equivalence rules for simple programming language

\[
\begin{align*}
\frac{x : \tau \vdash e_1 \equiv_{\tau} e_2}{x : \tau \vdash e_1 \equiv_{\tau} e_3} & \quad \frac{x : \tau \vdash e_2 \equiv_{\tau} e_3}{x : \tau \vdash e_1 \equiv_{\tau} e_3} \\
\frac{x : \tau \vdash e_1 \equiv_{\tau} e_2}{x : \tau \vdash e_2 \equiv_{\tau} e_3} & \quad \frac{x : \tau \vdash p(e_1) \equiv_{\text{dom}(p)} p(e_2)}{x : \tau \vdash p(e_1) \equiv_{\text{ran}(p)} p(e_2)}
\end{align*}
\]  

\[
\begin{align*}
\frac{x : \tau \vdash [e_1] \equiv_{\tau} [e_2]}{x : \tau \vdash [e_1] \equiv_{\tau} [e_2]} & \quad \frac{x : \tau \vdash \mu(e_1) \equiv_{\tau} \mu(e_2)}{x : \tau \vdash \mu(e_1) \equiv_{\tau} \mu(e_2)}
\end{align*}
\]  

\[
\begin{align*}
\frac{x : \tau \vdash e : \tau'}{x : \tau \vdash \mu([e]) \equiv_{\tau} e} & \quad \frac{x : \tau \vdash \text{let } x_1 \leftarrow e \text{ in } e_2 \equiv_{\tau} \text{let } x_1 \leftarrow e' \text{ in } e'_2}{x : \tau \vdash \text{let } x_1 \leftarrow e \text{ in } e_2 \equiv_{\tau} \text{let } x_1 \leftarrow e' \text{ in } e'_2}
\end{align*}
\]  

\[
\begin{align*}
\frac{x : \tau \vdash \text{let } x_1 \leftarrow e \text{ in } x_1 \equiv_{\tau} e}{x : \tau \vdash \text{let } x_1 \leftarrow e \text{ in } x_1 \equiv_{\tau} e} & \quad \frac{x : \tau \vdash p(e) \equiv_{\text{ran}(p)} \text{let } x_1 \leftarrow e \text{ in } p(x_1)}{x : \tau \vdash p(e) \equiv_{\text{dom}(p)} \text{let } x_1 \leftarrow e \text{ in } p(x_1)}
\end{align*}
\]  

\[
\begin{align*}
\frac{x : \tau \vdash \text{let } x_1 \leftarrow e_1 \text{ in } x_1 \equiv_{\tau} e_2 \equiv_{\tau} x_2 \equiv_{\tau} e_3}{x : \tau \vdash \text{let } x_1 \leftarrow e_1 \text{ in } x_1 \equiv_{\tau} e_2 \equiv_{\tau} x_2 \equiv_{\tau} e_3}
\end{align*}
\]  

Figure 17: Existence assertion rules for simple programming language

\[
\begin{align*}
\frac{\vdash \tau \text{ type}}{x : \tau \vdash x \downarrow \tau} & \quad \frac{x : \tau \vdash e_1 \equiv_{\tau} e_2}{x : \tau \vdash e_1 \downarrow \tau} \quad \frac{x : \tau \vdash e \equiv_{\tau'}}{x : \tau \vdash [e] \downarrow \tau'}
\end{align*}
\]  

\[
\begin{align*}
\frac{x : \tau \vdash e \downarrow \tau}{x : \tau \vdash x_1 : \tau \vdash \phi} & \quad \frac{x : \tau \vdash [e/x_1] \phi}{\text{where } \phi \text{ is one of } e_1 \equiv_{\tau}, e_2, e_1 : \tau', \text{ or } e_1 \downarrow \tau'}
\end{align*}
\]  

Figure 17: Existence assertion rules for simple programming language
The operational behavior of the language is described by an equivalence relation \( x : \tau \vdash e \equiv_{\tau} e' \) (see figure 16) and an existence relation \( x : \tau \vdash e : \sigma \) (see figure 17). The existence relation determines whether or not an expression can be considered to have the form \([e]\).

The denotation semantic functions considered for this language will all interpret types as objects and typing judgments as morphisms in some category. A semantic function is sound if

1. whenever \( x : \tau \vdash e_1 \equiv_{\tau} e_2 \), the meaning of \( x : \tau \vdash e_1 : \tau' \) is equal to the meaning of \( x : \tau \vdash e_2 : \tau' \)
2. whenever \( x : \tau \vdash e : \tau', \) there exists a unique morphism \( h \) from the meaning of \( \tau \) to the meaning of \( \tau' \) such that the meaning of \( x : \tau \vdash e : \tau' \) equals \( \eta \circ h \).

8.1. Extensional semantics

For a category \( C \) with a monad \( (T, \eta, \mu) \) we define the extensional denotational semantics, taken from [14], for this language as follows:

- For each base type \( \rho \), let \( A_{\rho} \) be some object in \( C \).
- If \( \tau = T\tau' \), let \( A_{\tau} = T A_{\tau'} \).
- For each \( p \), let \( [p] \) be a morphism from \( A_{\text{dom}(p)} \) to \( T A_{\text{ran}(p)} \).
- For each judgment \( x : \tau \vdash e : \tau' \), let \( \mathcal{M}[x : \tau \vdash e : \tau'] : A_{\tau} \to T A_{\tau'} \) be a morphism in the Kleisli category, defined inductively as follows:

\[
\mathcal{M}[x : \tau \vdash x : \tau] = \eta_{A_{\tau}} = \text{id}_{A_{\tau}}
\]
\[
\mathcal{M}[x : \tau \vdash p(e) : \text{ran}(p)] = \mu_{\text{ran}(p)} \circ T[p] \circ \mathcal{M}[x : \tau \vdash e : \text{dom}(p)]
\]
\[
\mathcal{M}[x : \tau \vdash [e] : T\tau'] = \eta_{T A_{\tau}} \circ \mathcal{M}[x : \tau \vdash e : \tau']
\]
\[
\mathcal{M}[x : \tau \vdash \mu(e) : \tau'] = \mu_{A_{\tau}} \circ \mathcal{M}[x : \tau \vdash e : T\tau']
\]
\[
\mathcal{M}[x : \tau \vdash \text{let } x_1 = e_1 \text{ in } e_2 : \tau_2] = \mu_{A_{\tau}} \circ T \mathcal{M}[x_1 : \tau_1 \vdash e_2 : \tau_2] \circ \mathcal{M}[x : \tau \vdash e_1 : \tau_1]
\]

Moggi in [14] showed that if the monad satisfies the mono-requirement, namely if \( \eta_A \) is a monomorphism for every object \( A \), the above semantics is sound. It is easy to see [14] that all the monads given in this paper satisfy the mono-requirement.

8.2. Intensional semantics

Suppose we have a category \( C \) with both a monad \( (T, \eta, \mu) \) and a comonad \( (G, \epsilon, \delta) \). It is easy to show that if a monad \( (T, \eta, \mu) \) satisfies the mono-requirement then so do the lifted monads \( (T, \hat{\eta}, \hat{\mu}) \) and \( (T', \hat{\eta}', \hat{\mu}') \). We can then use the extensional semantics above to define an intensional semantics by starting with the Kleisli category \( K(G) \) instead of \( C \) and using a lifted monad instead of the original. Thus we get the following two intensional semantics:
\[
M[x : \tau \vdash x : \tau]_{G,T} = \hat{\eta}_{A^G,T} \\
= \eta_{A^G,T} \circ \epsilon_{A^G_T} \\
= \text{id}^{A^G_T}_{A^G_T}
\]

\[
M[x : \tau \vdash p(e) : \text{ran}(p)]_{G,T} = \hat{\mu}_{A^G_T} \circ \hat{\eta}_{A^G_T} \circ \sigma_{A^G_T} \circ G.M[x : \tau \vdash e : \text{dom}(p)]_{G,T} \\
= \mu_{\text{ran}(p)} \circ \hat{\eta}_{A^G_T} \circ \sigma_{A^G_T} \circ G.M[x : \tau \vdash e : \text{dom}(p)]_{G,T} \\
= \mu_{A^G_T} \circ M[x : \tau \vdash e : \text{dom}(p)]_{G,T}
\]

\[
M[x : \tau \vdash [e] : T\tau']_{G,T} = \hat{\mu}_{A^G_T} \circ \hat{\eta}_{A^G_T} \circ \sigma_{A^G_T} \circ G.M[x : \tau \vdash e : T\tau']_{G,T} \\
= \mu_{A^G_T} \circ M[x : \tau \vdash e : T\tau']_{G,T}
\]

\[
M[x : \tau \vdash \text{let } x_1 \leftarrow e_1 \text{ in } e_2 : T\tau']_{G,T} = \hat{\eta}_{A^G_T} \circ \hat{\mu}_{A^G_T} \circ \hat{\eta}_{A^G_T} \circ \sigma_{A^G_T} \circ G.M[x : \tau \vdash e_1 : T\tau']_{G,T} \\
= \mu_{A^G_T} \circ M[x : \tau \vdash e_1 : T\tau']_{G,T}
\]

\[
\text{Figure 18: Denotational semantics using the monad } (\hat{T}, \hat{\eta}, \hat{\mu})
\]

1. Using the lifted monad \((\hat{T}, \hat{\eta}, \hat{\mu})\) (thus assuming the existence of a distributive law) we define

   • For each base type \(p\), \(A^G{T} = A_p\), and for type \(\tau = T\tau'\), \(A^G{T} = T\hat{A}^G{T} = TA^G{T}\).

   • For each constant \(p\), let \([p]_{G,T}\) be a morphism from \(GA^G_{\text{dom}(p)}\) to \(TA^G{T}\).

   • For each type judgement \(x : \tau \vdash e : \tau'\), let \(M[x : \tau \vdash e : \tau']_{G,T} : GA^G{T} \rightarrow TA^G{T}\) be defined as shown in figure 18.

2. Using the lifted monad \((\hat{T}', \hat{\eta}', \hat{\mu}')\) we define

   • For each base type \(p\), let \(A'^T = A_p\), and for type \(\tau = T\tau'\), \(A'^T = T'A'^{T'} = TGA'^{T'}\).

   • For each constant \(p\), let \([p]_{T'}\) be a morphism from \(GA'^T_{\text{dom}(p)}\) to \(TGA'^T\).

   • For each type judgement \(x : \tau \vdash e : \tau'\), let \(M[x : \tau \vdash e : \tau']_{T'} : GA'^T \rightarrow TGA'^T\) be defined as show in figure 19.

These two semantics are clearly sound; they are essentially the extensional semantics using different base categories and monads, and all such semantics are sound whenever the monad satisfies the mono-requirement.

One more semantic function can be derived for this language by taking the original semantics and directly adding a comonad to get a semantic definition in the other non-distributive Kleisli category, where a morphism from \(A\) to \(B\) is a morphism from \(GTA\) to \(TB\) in the original category.

• For each base type \(p\), let \(A'^{G'} = A_p\), and for \(\tau = T\tau'\), \(A'^{G'} = TA'^{G'}\).
\[ \mathcal{M}[x : \tau \vdash x : \tau]_T = \eta'_{\mathcal{T}^T} \]
\[ = \eta'_{\mathcal{T}^T_{\text{ran}(p)}} \]
\[ = \text{id}_{\mathcal{T}^T_{\text{ran}(p)}} \]
\[ \mathcal{M}[x : \tau \vdash p(e) : \text{ran}(p)]_T = \mu'_{\mathcal{T}^T_{\text{ran}(p)}} \circ \mathcal{T}[p]_T \circ \mathcal{M}[x : \tau \vdash e : \text{dom}(p)]_T \]
\[ = \mu_{\mathcal{T}^T_{\text{ran}(p)}} \circ \mathcal{T}[p]_T \circ \mathcal{M}[x : \tau \vdash e : \text{dom}(p)]_T \]
\[ \mathcal{M}[x : \tau \vdash [e] : T\tau']_T = \eta'_{\mathcal{T}^T'_{\text{ran}(p)}} \circ \mathcal{M}[x : \tau \vdash e : \tau']_T \]
\[ \mathcal{M}[x : \tau \vdash \mu(e) : \tau']_T = \mu'_{\mathcal{T}^T'_{\text{ran}(p)}} \circ \mathcal{M}[x : \tau \vdash e : \tau']_T \]
\[ \mathcal{M}[x : \tau \vdash \text{let } x_1 \leftarrow e_1 \text{ in } e_2 : \tau_2]_T \]
\[ = \mu'_{\mathcal{T}^T_{\text{ran}(p)}} \circ \mathcal{T}[\mathcal{M}[x_1 : \tau_1 \vdash e_1 : \tau_1]_T \circ \mathcal{M}[x : \tau \vdash e_1 : \tau_1]_T \]
\[ = \mu_{\mathcal{T}^T_{\text{ran}(p)}} \circ \mathcal{T}[\mathcal{M}[x_1 : \tau_1 \vdash e_1 : \tau_1]_T \circ \mathcal{M}[x : \tau \vdash e_1 : \tau_1]_T \]
\[ = \mathcal{M}[x_1 : \tau_1 \vdash e_1 : \tau_1]_T \circ \mathcal{M}[x : \tau \vdash e_1 : \tau_1]_T \]

Figure 19: Denotational semantics using the monad \( \langle T', \eta', \mu' \rangle \)

- For each constant \( p \), let \( [p]_{G'} \) be a morphism from \( GTA'_{\text{dom}(p)} \) to \( TA'_{\text{ran}(p)} \)
- For each judgment \( x : \tau \vdash e : \tau' \), let \( \mathcal{M}[x : \tau \vdash e : \tau']_{G'} : GTA'_{G'} \to TA'_{G'} \) be defined as shown in figure 20

Unfortunately, this semantics is not sound \( (x : \tau \vdash x \downarrow_\tau \) is false). This is because we lack control over the monad part of \( GTA \). If instead we restrict \( GTA \) to the range of \( G\eta_A \) we can get the following "semi-soundness" property:

**Theorem 11** For all expressions \( e, e_1, \) and \( e_2 \) the following two properties hold:

1. If \( x : \tau \vdash e_1 \equiv_e e_2 \) then \( \mathcal{M}[x : \tau \vdash e_1 : \tau']_{G'} \circ G\eta_{A_{G'}} = \mathcal{M}[x : \tau \vdash e_2 : \tau']_{G'} \circ G\eta_{A_{G'}} \)

2. If \( x : \tau \vdash e \downarrow_\tau \) then there exists an \( h : GA'_{G'} \to A'_{G'} \) such that \( \mathcal{M}[x : \tau \vdash e : \tau']_{G'} \circ G\eta_{A_{G'}} = \eta_{A'} \circ h \).

**Proof.** By structural induction on \( e \), using as a substitution lemma

\[ \mathcal{M}[x : \tau \vdash [e/x]e' : \tau'']_{G'} = \mathcal{M}[x : \tau \vdash e' : \tau'']_{G'} \circ G \mathcal{M}[x : \tau \vdash e : \tau']_{G'} \circ \delta_{TAG'} \]

whenever \( x : \tau \vdash e : \tau' \) and \( x : \tau' \vdash e' : \tau'' \). □
\[ \mathcal{M}(x : \tau \vdash x : \tau)_{G'} = \epsilon_{\text{TA}G'} \]
\[ = \text{id}_{G'} \]
\[ \mathcal{M}(x : \tau \vdash p(e) : \text{ran}(p))_{G'} = [p]_{G'} \circ G \mathcal{M}(x : \tau \vdash e : \text{dom}(p))_{G'} \circ \delta_{\text{TA}G'} \]
\[ = [p]_{G'} \circ \delta_{\text{TA}G'} \]
\[ \mathcal{M}(x : \tau \vdash e : T\tau')_{G'} = \eta_{\text{TA}G'} \circ \mathcal{M}(x : \tau \vdash e : \tau')_{G'} \]
\[ = \mu_{\text{A}G'} \circ \mathcal{M}(x : \tau \vdash e : T\tau')_{G'} \]
\[ \mathcal{M}(x : \tau \vdash \text{let } x_1 \leftarrow e_1 \text{ in } e_2 : \tau_2)_{G'} \]
\[ = \mathcal{M}(x_1 : \tau_1 \vdash e_2 : \tau_2)_{G'} \circ G \mathcal{M}(x : \tau \vdash e_1 : \tau_1)_{G'} \circ \delta_{\text{TA}G'} \]
\[ = \mathcal{M}(x_1 : \tau_1 \vdash e_2 : \tau_2)_{G'} \circ \mathcal{M}(x : \tau \vdash e_1 : \tau_1)_{G'} \]

Figure 20: Denotational semantics using the comonad \((G', \epsilon', \delta')\)

### 8.3. Relating intensional and extensional semantics

For all comonads \((G, \epsilon, \delta)\) there is an (adjoint) pair of functors \(F : \mathcal{C} \rightarrow \mathcal{K}(G)\) and \(U : \mathcal{K}(G)\) the Kleisli category \(\mathcal{K}(G)\) to the underlying category, where

- For all objects \(A, FA = A\)
- For all \(f : A \rightarrow B, Ff = \epsilon_A \circ Gf = f \circ \epsilon_A\)
- For all objects \(A, UA = GA\)
- For all \(a : GA \rightarrow B, Ua = Ga \circ \delta_A\).

(see [2] and [6]). There is also a similar pair of functors, of course, for monads.

For some comonads there is a much closer relation:

**Definition 5** A computational comonad \((G, \epsilon, \delta, \gamma)\)\([6]\) is a comonad \((G, \epsilon, \delta)\) plus a natural transformation \(\gamma : I \rightarrow G\) such that

- \(\epsilon_A \circ \gamma_A = \text{id}_A\)
- \(\delta_A \circ \gamma_A = G\gamma_A \circ \gamma_A (= \delta_A \circ \gamma_A \text{ by naturality})\)

Whenever \((G, \epsilon, \delta, \gamma)\) is a computational comonad, then there is another functor, \(H : \mathcal{K}(G) \rightarrow \mathcal{C}\) defined by

- \(HA = A\)
- For \(a : GA \rightarrow A, Ha = a \circ \gamma_A\).
From this we can define a family of relations \( \sim_{A,B} \): if \( a : GA \to B \) and \( f : A \to B \) then \( a \sim_{A,B} f \) whenever \( f = Ha = a \circ \gamma_A \) (the subscripts will be dropped when \( A \) and \( B \) are understood). We can also define two functions \( \text{alg} \) and \( \text{fun} \) as the morphism parts of the functors \( F \) and \( H \) respectively. When we use comonads to represent intensional semantic behavior, \( \text{fun} \) takes an extensional function and give a default intensional algorithm, and \( \text{alg} \) takes an algorithm and returns its extensional behavior (for further details see [7]).

All of the comonads given in the paper, if we assume one very general extra condition, are computational, with \( \gamma \) defined as follows:

**Product comonad** \((X, \pi, \delta)\): Assume that there exists at least one morphism from \( 1 \) to \( X \). Then for all objects \( A \), and for all morphisms \( x : 1 \to X \), \( \gamma_A^x = (\text{id}_A, x \circ \lambda) \). For \( \text{Cont} \), given \( x \in X \) and for all objects \( A \), \( \gamma_A^x = Aa.a \).

**Exponentiation comonad** \((V, \text{val}, \text{pre})\): For all objects \( A \), let \( \gamma_A = \text{curry}(\pi_1) \). For \( \text{Cont} \), \( \gamma_A(a) = \lambda v.a \).

**Strict path comonad** \((V_S, \text{sval}, \text{spre})\): For all objects \( A \), \( \gamma_A = \lambda a.a' \), essentially the same definition as the one for exponentiation.

8.4. Relating \( \mathcal{K}(G, T) \) to \( \mathcal{K}(T) \)

**Theorem 12** Let \( (G, \varepsilon, \delta, \gamma) \) be a computational comonad on \( C \), \( (T, \eta, \mu) \) be a monad on a category \( C \), and \( \sigma \) be a distributive law of \( T \) over \( G \). If for every object \( A \), \( \sigma_A \circ \gammaTA = T\gamma_A \) (figure 22) then the lifted comonad \( (G, \varepsilon, \delta, \tilde{\gamma}) \) is also computational, where for each object \( A \), \( \tilde{\gamma}_A = \eta_{GA} \circ \gamma_A \).

**Proof.** By straightforward diagram chasing. Figure 23 contains the diagram proving naturality of \( \tilde{\gamma} \). The upper path represents the expansion of \( \tilde{\gamma}_B \circ f \), the lower path represents the expansion of \( Gf \circ \tilde{\gamma}_A \). \( \Box \)

From the results of the previous theorem we can construct a lifted pair of functors \( (\tilde{F}, \tilde{H}) \), and a family of relations \( \sim_{A,B} \) relating the Kleisli category \( \mathcal{K}(T) \) to the doubly-lifted category \( \mathcal{K}(G, T) \), as follows:

- For all objects \( A \), \( \tilde{F}A = \tilde{HA} = A \)
- For all \( f : A \to TB \), \( \tilde{F}f = f \circ \varepsilon_A = f \circ \varepsilon_A \)

35
Figure 22: Condition needed to lift computational comonad

\[ \begin{array}{c}
TA \\
\gamma_{TA} \\
GTA \\
\sigma_A \\
TGA
\end{array} \quad \begin{array}{c}
\gamma_{TA} \\
T\gamma_{TA} \\
\end{array} \]

Figure 23: Proof that \( \hat{\gamma}_B \circ f = \hat{G}_f \circ \hat{\gamma}_A \)

\[ \begin{array}{c}
GA \\
\gamma_{GA} \\
G^2A \\
\eta_{G^2A} \\
TG^2A \\
\end{array} \quad \begin{array}{c}
f \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \qua
For all $a : GA \to TB$, $\hat{H}a = a \circ \gamma_A = a \circ \gamma_A$.

For all $a : GA \to TB$ and $f : A \to TB$, $a \sim f$ whenever $f = \hat{H}a = a \circ \gamma_A$.

There are also functions $\text{alg}$ and $\text{fun}$ as before, with $\text{alg}(f) = f \circ \epsilon_A$, and $\text{fun}(a) = a \circ \gamma_A$. Note that all of these relations are the same as for the standard computational comonad, but for different categories.

All the comonad/monad combinations from table 1 that have distributive laws also satisfy the condition of figure 22 except for the power set monad over the exponentiation comonad. It can easily be seen why it fails. If we let $V = \{1, 2\}$ ($e = 2$ and $m = \text{max}$) and $A = \{a, b\}$, then

$$\sigma_A \circ \gamma_{pA}(\{a, b\}) = \sigma_A(\{1 \mapsto \{a, b\}, 2 \mapsto \{a, b\}\})$$

$$= \{1 \mapsto a, 2 \mapsto a, 1 \mapsto a, 2 \mapsto b, 1 \mapsto b, 2 \mapsto b\}$$

but $P_{\gamma_A}(\{a, b\}) = \{\gamma_A(a), \gamma_A(b)\} = \{1 \mapsto a, 2 \mapsto a, 1 \mapsto b, 2 \mapsto b\}$.

Note, however that the lax distributive laws $\epsilon^1$ for the Hoare and Smyth power domains do satisfy the requirement of figure 22. Essentially the closure properties of the two power domains fill in the elements missing when using the plain power set.

### 8.5. Relating $K(T)$ and $K(G')$

If we look at the lifted comonad $G'$, which did not require a distributive law, the obvious candidate for $\gamma' : A \to TGTA$ is $\gamma'_A = \eta_{GTA} \circ \gamma_TA \circ \eta_A$. This $\gamma'$, however, is not natural. Nevertheless we can relate both $K(T)$ and $K(G')$ to $C$ (actually, the subcategory of $C$ generated by the range of $T$). To do this we define a pair of functors $(U_T, H_G)$, with $U_T : K(T) \to C$ and $H_T : K(G') \to C$, defined by

- For all objects $A$, $U_TA = H_G A = TA$
- For all $f : A \to TB$, $U_Tf = \mu_B \circ Tf$
- For all $a : GTA \to TB$, $H_TA = a \circ \gamma_TA$.

Note that $U_T$ is the monad equivalent of the functor $U$ mentioned at the beginning of this section, and $H_T$ is the functor $H$ restricted to objects in the range of $T$. Thus if we let $F_T$ to be the functor $F$ restricted to objects in the range of $T$, we then have that $H_FT = I$ and thus $H_TF_TU_T = U_T$.

From these functors we can define a family of relations $\sim_{A,B}^G$ as follows: given a morphism $f : A \to TB$ in the Kleisli category and a morphism $a : GTA \to TB$ in the lifted Kleisli category, $a \sim_{A,B}^G f$ if and only if $H_TA = U_Tf$, i.e. if $a \circ \gamma_TA = \mu_B \circ Tf$

For $f : A \to TB$ and $a : GTA \to TB$, the relation $\sim_{A,B}^G$ has the following properties:

- $F_TU_Tf = \mu_B \circ Tf \circ \epsilon_TA \sim^G f$. Thus we can define a function $\text{alg}^G(f) = \mu_B \circ Tf \circ \epsilon_TA$ giving us a default intensional morphism for each extensional one.
- It is possible that there is no morphism $f$ such that $a \sim^G f$. For example in the category $\text{Set}$, let $E = \{err_1, err_2\}$, let $B$ and $X$ be sets, and let $x \in X$. Then using the computational product comonad
\((X, \pi_1, \delta, \gamma)\), where for \(b \in B\), \(\gamma^b(b) = (b, x)\), and the coproduct monad \((E, \eta_1, \mu)\), we define the function \(a : (B + E) \times X \to B + E\) to be

\[
a((b, x)) = \begin{cases} 
\eta_1(b) & b \in B, \\
\mu_2(x) & b \in E
\end{cases}
\]

Then for all \(f : B \to TB, \mu_B \circ Ef(\mu_2(\eta_1)) = (id_B, \mu_2)(f + id_E)(\mu_2(\eta_1)) = \mu_2(\eta_1), \) but \(a(\gamma^b(\mu_2(\eta_1))) = \mu_2(\eta_1), \) so \(a \sim T f\).

We can find similar examples for most other non-trivial monads. In fact if for every \(a\) there exists an \(f\) such that \(a \sim T f\), then \(\eta_1 \circ \epsilon_1 = \eta_1 \circ \epsilon\) (to see this let \(a = \eta_1 \circ \epsilon\)) implying that \(T \eta\) is isomorphic to \(TA\).

- If \(a \sim T f\), then \(f = a \circ \gamma_{TA} \circ \eta\). Thus we can define a function \(\text{fun}^G(a) = a \circ \gamma_{TA} \circ \eta\), giving us the extensional part of \(a\). Although this function is defined for all \(a\), from the item above we know that it may not be true that \(a \sim T \text{fun}^G(a)\).

- \(\text{fun}^G(\text{alg}^G(f)) = f\) (although it may not be true that \(\text{alg}^G(\text{fun}^G(a)) = a\)).

- For all \(f' : B \to TC\) and \(a' : GC \to TG\), if \(a \sim T f\) and \(a' \sim T f'\), then \(a' \circ f = a \circ f'\). This means that \(\text{alg}^G(f') \circ \text{alg}^G(f) = \text{alg}^G(f') \circ \text{alg}^G(a') \circ \text{alg}^G(f)\), and when \(a \sim T \text{fun}^G(a)\) and \(a' \sim T \text{fun}^G(a')\), then \(\text{fun}^G(a' \circ f) = \text{fun}^G(a') \circ \text{fun}^G(f(a'))\) and also \(a' \sim T \text{fun}^G(a' \circ a)\).

8.6. Relating \(K(T)\) and \(K(T')\)

These two categories are not related via a Kleisli construction (actually \(K(T')\) is isomorphic to a full subcategory of \(K(T)\)); there is thus no point in looking for a version of \(\gamma\). We can, however, form a family of relations as we did in the previous section. For any pair of morphisms \(f : A \to TB\) and \(a : GA \to TGB\), let \(a \sim I_B f\) if and only if \(a \circ \gamma_A = \gamma_B \circ f\). Note that, unlike the last section, the constructions \(T \gamma B \circ f\) and \(a \circ \gamma A\) are not parts of functors, since the resulting morphisms are of the form \(A \to TGA\) and do not compose properly to form a category.

The family of relations \(\sim I\) has similar properties to \(\sim T\): for all \(f : A \to TB\) and \(a : GA \to TGB\)

- \(T \gamma_B \circ f \circ \epsilon_A \sim I f\), so again we can define a function \(\text{alg}^I(f) = T \gamma_B \circ f \circ \epsilon_A\) on morphisms in \(K(T)\).

- Again there are morphisms \(a\) such that there is no \(f\) with \(a \sim I f\). For example, in \(\text{Set}\), let \((X, \pi_1, \delta, \gamma)\) be the product comonad from from the set \(\{1, 2\}\) with \(\gamma_A = \lambda a.A(a, 1)\) for any set \(A\), and let \(B\) be any set. Let the monad \(\xi\) be the power set monad \(\mathcal{P}\). Let \(a = \lambda (b, n).\{(b, 2)\}\). Then for all morphisms \(f : B \to PB, \mathcal{P} \gamma_B \circ f(b) = \{(b', 1) \mid b' \in f(b)\}, \) but \(a \circ \gamma_B(b) = a((b, 1)) = \{(b, 2)\}. \) In order to guarantee that there is always an \(f\) such that \(a \sim I f\), we must have that \(\gamma_A \circ \epsilon_A = \text{id}_{GA}\), which implies that \(GA\) is isomorphic to \(A\).

- If \(a \sim I f\) then \(f = T \epsilon_B \circ a \circ \gamma_A\). Thus again we can define a function \(\text{fun}^I(a) = T \epsilon_B \circ a \circ \gamma_A\), and again \(\text{fun}^I(\text{alg}^I(f)) = f\).

- For all \(f' : B \to TC\) and \(a' : GC \to TGC\), if \(a \sim I f\) and \(a' \sim I f'\), then \(a' \circ f = a \circ f'\) (when the composition is well defined). Thus we know that \(||f'\circ f\| = ||f'\| \circ ||f\||\) and \(||f' \circ a\| = ||a'\| \circ ||f\||\) whenever \(a'\) and \(a\) have the property that \(a' \sim T \text{fun}^I(a')\) and \(a \sim T \text{fun}^I(a)\).
8.7. Relating the intensional and extensional semantics

Theorem 13 For all computational comonads \( (G, \epsilon, \delta, \gamma) \) and all monads \( (T, \eta, \mu) \) that satisfy the mono-
requirement, whenever \( \text{dom}(p) \) for all constants \( p \), it follows that

\[
A^G_{\tau} = A_{\tau}, \quad \text{and} \quad M[x : \tau \vdash e : \tau']^G_{\tau} \sim M[x : \tau \vdash e : \tau']^T_{\tau}, \quad \text{i.e.} \quad M[x : \tau \vdash e : \tau']^G_{\tau} = M[x : \tau \vdash e : \tau']^T_{\tau} \circ \gamma^A_{\tau}.
\]

Proof. That \( A^G_{\tau} = A_{\tau} \) is obvious, since they have the same definition. The rest is shown by straightforward
induction over the structure of \( e \). \( \square \)

For the \( M \) function, we cannot simply state that \( M[x : \tau \vdash e : \tau']^T_{\tau} \sim M[x : \tau \vdash e : \tau']^T_{\tau} \) since
the types do not match \( (A^T_\tau \neq A_{\tau}) \). All we need to do, however, is add enough \( \gamma \)'s after the evaluation
of \( M[x : \tau \vdash e : \tau'] \) to match up the types and then we do get an equivalence. This is made precise below:

Definition 6 For all types \( \tau \), let \( \Gamma[\tau] : A_{\tau} \to A^T_{\tau} \) be defined inductively by \( \Gamma[\rho] = \text{id}_{A_{\tau}} \) and \( \Gamma[T] = T\gamma^A_{A^T_{\tau}} \circ \Gamma[T] \).

Theorem 14 For all computational comonads \( (G, \epsilon, \delta, \gamma) \) and any monad \( (T, \eta, \mu) \) that satisfies the mono-
requirement, \( \Gamma[T\gamma^A_{\tau}] \circ M[x : \tau \vdash e : \tau']^T_{\tau} = M[x : \tau \vdash e : \tau']^T_{\tau} \circ \gamma^A_{A^T_{\tau}} \circ \Gamma[\tau] \) whenever for all constants \( p \),
\( \Gamma[\text{Tran}(p)] \circ \Gamma[p] = [\rho]^T_{\tau} \circ \gamma^A_{A^T_{\tau}} \circ \Gamma[\text{dom}(p)] \).

Proof. By straightforward induction on \( e \). \( \square \)

For the third semantics, \( M \), the relationship \( M[x : \tau \vdash e : \tau']^G_{\tau} \sim M[x : \tau \vdash e : \tau']^T_{\tau} \) does not hold when \( e = [e'] \), except where \( \eta^A \circ T\eta_A = T\eta_A \). Since \( \mu_A \circ \eta^A = \mu_A \circ T\eta_A \) with certain restrictions we can
relate the two semantics:

Definition 7 For all \( n \geq 0 \), let \( \mu_A^n : T^{n+1}A \to TA \) be defined inductively as follows: \( \mu_A^0 = \text{id}_{TA} \) and for \( n > 0 \),
\( \mu_A^{n+1} = \mu_A^n \circ \mu_{T^{-1}A^n} \).

Definition 8 A morphism \( f : T^{n+1}A \to T^nB \) is simple if there exists an \( f' : A \to TB \), called a simplification
of \( f \), such that \( \mu_A^n \circ f = T^n f' \circ \mu_A^m \).

Note that for \( a : GTA \to TB \), if there exists an \( f \) such that \( f \sim G \), then \( a \circ \gamma_TA \) is simple and \( f \) is
a simplification, i.e. an algorithm that has been built from a morphism \( f : A \to TB \) in a natural way
will be simple. Also note that for all morphisms \( f : A \to TB \), \( n > 0 \) and \( m \geq 0 \), \( f \) is a simplification of
\( \mu_B \circ T^nf \circ \mu_A^m \).

An example (in Set) of a morphism that is not simple is the set complement function \( f : PA \to PA \),
\( f X = A - X \). To see this, suppose that there existed a simplification \( f' : A \to PA \) of \( f \). Then \( f = \bigcup_A \circ Pf' \),
\( \text{i.e. for any set } X \subseteq A, \quad f X = \bigcup\{f'(x) \mid x \in X\} = \{a \in A \mid a \notin X\}. \)
This implies that for all $X \subseteq A$, and all $x \in X, f'(x) \notin X$. By setting $X = A$ we can easily see that no such $f'$ can exist.

**Theorem 15** Let $(G, \varepsilon, \delta, \gamma)$ be a computational comonad and $(T, \eta, \mu)$ be a monad over a category $C$ that satisfies the mono-requirement. Suppose that for every constant $p$, both $\mu_{A_{\text{unint}}}(T[p])$ and $[p]_{G'} \circ \gamma_{TA_{\text{unint}}}$ are simple, with equal simplifications. Then for all $x : \tau \vdash e : \tau'$,

$$\mu_{A_{\text{unint}}}^{-1} \circ T.M[x : \tau \vdash e : \tau'] = \mu_{A_{\text{unint}}}^{-1} \circ T.M[x : \tau \vdash e : \tau'] \circ [p]_{G'} \circ \gamma_{T\alpha_{A_{\text{unint}}}}$$

where $\tau = T\rho$ and $\tau' = T\rho'$ for base types $\rho$ and $\rho'$.

**Proof.** It can be shown by straightforward structural induction that for all $x : T\rho \vdash e : T\rho', \mu_{TA_{\text{unint}}} \circ M[x : T\rho \vdash e : T\rho']$ and $M[x : T\rho \vdash e : T\rho']_{G'} \circ \gamma_{T\alpha_{\text{unint}}}$, are both simple and that $M[x : \rho \vdash \overline{e} : \rho']$ is a simplification for both, where $\overline{e}$ is formed from $e$ by removing all $\mu$'s and $\eta$'s and converting any constants $p$ to new constants $\overline{p}$, whose meaning is a simplification of $\mu_{A_{\text{unint}}} \circ T[p]$ and $[p]_{G'} \circ \gamma_{T\alpha_{\text{unint}}}$ (by assumption they have equal simplifications). The theorem then follows directly. $\Box$

From the proof it is not difficult to see that the two semantics were related by effectively removing all explicit references in the language to the $\mu$ and the $\eta$ construct. For more monads, however, there is no need to use the $[\cdot]$ construct to get a meaningful language. For example typical language constructs that use the power set and power domain monads, such as parallel composition and nondeterministic choice, require only simple morphisms to be meaningful, even when adding intensional behavior. Thus we still have a useful relationship between the intensional and extensional semantics as we had for the other semantic interpretations, stated in theorems 13 and 14.

9. **Conclusions and Future Work**

The combination of a comonad and a monad using a distributive law provides an elegant method for obtaining an intensional semantics from a monadic extensional semantics. It relates as well to the extensional monadic semantics as the standard intensional semantics does to the plain extensional semantics. Unfortunately, there are monads and comonads currently of interest in computer science that do not have distributive laws. Thus we discussed alternative ways to combine comonads and monads without a distributive law and have explored the more complex relationship between extensional and intensional interpretations obtained this way.

There are still other monads and comonads of interest to computer science that were not explored in this paper, such as the monad representing continuations. It may be that later on some combinations will be found that will be interesting enough to explore further. It should also be interesting to explore the uses and limitations of the lax distributive law of the Hoare and Smyth power domain monads over the exponentiation comonads. Also the language given in this paper was extremely simple and not particularly useful in itself; it mostly exists so we can study monads without worrying about products; most of the Kleisli categories built over monads do not have products. There are many other more complex languages that use monads and are worth studying.

**References**


