Monads and Algebras - An Introduction

Uday S. Reddy

April 27, 1995
1 Algebras of monads

Consider a simple form of algebra, say, a set with a binary operation. Such an algebra is a pair \((X, \ast : X \times X \to X)\). Morphisms of these algebras preserve the binary operation: \(f(x \ast y) = f(x) \ast f(y)\).

We first notice that the domain of the operation is determined by a functor \(F : \text{Set} \to \text{Set}\) (the diagonal functor \(FX = X \times X\)). In general, given an endofunctor \(F : C \to C\) on a category \(C\), we can speak of the “algebras” for \(F\), which are pairs \((X, \alpha : FX \to X)\) of an object \(X\) of \(C\) and an arrow \(\alpha : FX \to X\). Morphism preserve the respective operations, i.e.,

\[
\begin{array}{ccc}
FX & \xrightarrow{Ff} & FY \\
\alpha & & \beta \\
X & \xrightarrow{f} & Y
\end{array}
\]

Now, in algebra, it is commonplace to talk about “derived operators.” Such an operator is determined by a term made up of variables (over \(X\)) and the operations of the algebra.

For example, for the simple algebra with a binary operation, terms such as \(x, x \ast (y \ast z)\), and \((x \ast y) \ast (z \ast w)\) determine derived operators. From a categorical point of view, such derived operators are just compositions of the standard operators and identity arrows:

\[
\begin{array}{ccc}
X & \xrightarrow{\text{id}_X} & X \\
X \times (X \times X) & \xrightarrow{id_X \times \alpha} & X \times X & \xrightarrow{\alpha} & X \\
(X \times X) \times (X \times X) & \xrightarrow{\alpha \times \alpha} & X \times X & \xrightarrow{\alpha} & X
\end{array}
\]

To formalize such derived operators, we think of another functor \(T : C \to C\) into which we can embed the domains of all the derived operators. In particular, we should have embeddings \(I \to T\) and \(FT \to T\). If \(C\) has colimits, we can just take the initial solution of the recursive definition:

\[
TX = X + FTX
\]

(1)

to obtain the functor \(T\). We may think of \(T\) as a “closed” functor that includes all possible compositions of \(F\)’s and identity functors. For example, for the simple algebra with a binary operation, \(TX\) is the set of all binary trees over \(X\). We can then define the collection of all derived operators by a map \(\bar{\alpha} : TX \to X\) given by the inductive definition \(\bar{\alpha} = [\text{id}, \alpha \circ F\bar{\alpha}]\).

We are moving from algebras of functors \(F\) towards algebras of “closed” functors \(T\). Soon, we will dispense with functors \(F\) altogether and add appropriate structure to the “closed” functors \(T\) to form monads. In this setting, it is useful to think of \(TX\) as forming an abstract form of “terms” over \(X\). The map \(\bar{\alpha}\) is then viewed as a “valuation” map, which gives values to such terms in \(X\).

So far, we have just restated algebras of functors as algebras of “closed” functors. If all we can say in terms of “closed” functors is a restatement of what we could say in terms of functors, there is not much point to the exercise. Is there something to be gained by going to “closed” functors? Indeed, it becomes apparent as soon as we consider algebras with equational axioms.

Let us refine our simple algebras by assuming an associativity axiom for the binary operation:

\[
x \ast (y \ast z) = (x \ast y) \ast z
\]

(2)

As such, the axiom is an equivalence between two derived operators. When we use algebras of functors, we have to add the axiom explicitly after the fact. On the other hand, when we use algebras of “closed” functors, we can refine the “closed” functors by taking into account
equational axioms such as associativity. For example, instead of taking $TX$ to be the set of binary trees over $X$, we can take it to be set of binary trees “modulo associativity,” i.e., (nonempty) lists, over $X$. $T$ is still a “closed” functor even though it is not freely generated by a functor $F$. Algebras of the form $\langle X, \bar{\alpha} : TX \to X \rangle$ will automatically satisfy the associativity law (2) because the domain of $\bar{\alpha}$ already incorporates associativity.

Let us look more closely at the structure required of “closed” functors. We need two natural transformations $\eta : I \to T$ and $\mu : TT \to T$. These transformations need not be embeddings any more because we expect that our abstract terms will be obtained by identifying many concrete terms. The transformation $\eta_X$ regards an element of $X$ as an abstract term. Think of it as a “variable term” $\langle x \rangle$. The transformation $\mu_X$ is similar to a substitution operator. It converts an abstract term over abstract terms, say $t[u/x]$ to an abstract term. Write the resulting term as $t[u/x]$. When the “closed” functor is freely generated, as in (2), a variable term $\langle x \rangle$ will be just $x$, and a composite term $t(x := u)$ will be just $t[u/x]$. But, in the more general context, they can be different. Nevertheless, they must satisfy the usual laws of substitution:

$t\langle x := y \rangle = t[y/x]$


$t\langle x := t \rangle = t$


$t\langle x := u(y := v) \rangle = t\langle x := u \rangle \langle y := v \rangle$

It turns out that every term has exactly one “variable” (which could be of an appropriate product type to accommodate multiple indeterminates). So, we can dispense with variable names and use the notation $\langle \bullet \rangle$ for a variable term, $t(u)$ for a substitution term and $t[u]$ for a term of terms. In this notation, the laws of substitution should read as:

$t(\langle \bullet \rangle) = t$

$t(\bullet(t)) = t$

$t(u(v)) = t(u)v$

These considerations lead to the definition of monads:

**Definition 1** A monad on a category $C$ is a triple $T = \langle T, \eta, \mu \rangle$ of an endofunctor and two natural transformations as follows:

$$
\begin{align*}
TT & \xrightarrow{\mu} T \\
T & \xrightarrow{\eta} I \\
I & \xrightarrow{\eta} T \\
\end{align*}
$$

This data must satisfy the following “unit laws” and the “associativity law”

$$
\begin{align*}
T = TI & \xrightarrow{T\eta} TTT \\
TT & \xrightarrow{T\mu} T \\
TTT & \xrightarrow{T\mu} TT
\end{align*}
$$

The natural transformations $\eta$ and $\mu$ are called the **unit** and the **multiplication** of the monad respectively.

The algebras for a monad will be pairs $\langle X, \alpha : TX \to X \rangle$ of objects $X$ and arrows $\alpha$. Regard $TX$ as a set of abstract terms denoting derived operators, and $\alpha$ as a valuation map that assigns values to such terms. These data must satisfy certain axioms too.
1. Evaluating a variable term should give back the value of the variable. So, the composite

\[ X \xrightarrow{\eta_X} TX \xrightarrow{\alpha} X \]

must be the identity.

2. A term obtained by substitution of variables, say, \( t[u/x] \) can be evaluated in two ways. We can evaluate \( u \) and \( t \) in order. Or, we can take the composite term \( t(x := u) \) and evaluate it at once. These two evaluations must be equal.

\[ TTX \xrightarrow{T\alpha} TX \]
\[ \mu_X \downarrow \alpha \]
\[ TX \xrightarrow{\alpha} X \]

A morphism of \( T \)-algebras is an arrow \( h : \langle X, \alpha \rangle \to \langle Y, \beta \rangle \) such that

\[ TX \xrightarrow{Th} TY \]
\[ \alpha \downarrow \beta \]
\[ X \xrightarrow{h} Y \]

(Note that this is unchanged from algebras of functors.) It is common practice to denote an algebra by its structure map \( TX \xrightarrow{\alpha} X \), because \( X \) is implicit in its type. A morphism of algebras is denoted by the notation:

\[
\begin{pmatrix}
    TX \\
    \alpha \\
    X
\end{pmatrix}
\xrightarrow{h}
\begin{pmatrix}
    TY \\
    \beta \\
    Y
\end{pmatrix}
\]

The category of \( T \)-algebras and algebra morphisms is denoted \( C^T \) and called the Eilenberg-Moore category of \( T \).

A free algebra for \( T \) is an algebra of the form \( \langle TX, \mu_X : TTX \to TX \rangle \). The laws of algebras

\[ TX \xrightarrow{T\eta_X} TTX \]
\[ \mu_X \downarrow \]
\[ TX \]
\[ TTX \xrightarrow{T\mu_X} TTX \]
\[ \mu_X \downarrow \]
\[ TX \]

trivially follow from the properties of the monad \( T \). In a set-theoretic context, the elements of a free algebra are simply the abstract terms obtained from \( T \).

In presenting the category of free \( T \)-algebras, one uses a simplification. Note that any morphism \( h : \langle TX, \mu_X \rangle \to \langle TY, \mu_Y \rangle \) is uniquely determined by its “action on \( X \),” viz., \( \eta_X ; h : X \to TX \to TY \). Given such an action \( f : X \to TY \), we can recover \( h \) by \( h = Tf ; \mu_Y : TX \to TTY \to TY \). Thus, in presenting the category of free algebras, we can dispense with the algebra structure and take arrows to be the arrows (in \( C \)) of the form \( X \to TY \).

Formally one defines a category \( C_T \), called the Kleisli category of \( T \), as follows:

- **Objects** are the objects \( X \) of \( C \), understood as standing for free algebras \( \langle TX, \mu_X \rangle \).
- **Arrows** \( f : X \to Y \) are the arrows \( f : X \to TY \) of \( C \), understood as standing for morphisms \( f^* = Tf ; \mu_Y : \langle TX, \mu_X \rangle \to \langle TY, \mu_Y \rangle \).
The identity arrows are \( \eta_X : X \to TX \) and the composition of \( f : X \to TY \) and \( g : Y \to TZ \) is \( f \circ g \). The Kleisli category \( C_T \) is equivalent to the full subcategory of free algebras in \( C^T \).

In general, it would appear that Kleisli categories are useful for modelling syntax, i.e., the surface form of semantic descriptions or programs, as evidenced by [Mog91, Wad92]. On the other hand, Eilenberg-Moore categories play a role in structuring semantics, in particular, the various structures used for semantic models.

We give several examples of monads and their algebras:

**Example 2** There is a monad \( \text{list} : \text{Set} \to \text{Set} \) which assigns to each set \( X \) the set of lists (or sequences) over \( X \). The unit \( \eta_X \) maps an element \( x \in X \) to the singleton sequence \( \langle x \rangle \) and the multiplication \( \mu_X \) maps a sequence of sequences \( \langle s_1, \ldots, s_n \rangle \) to their concatenation \( s_1 \cdots s_n \).

Put another way, an abstract term for the list monad is a formal product of the form \( x_1 \cdots x_n \). A variable term is just a product of a single variable \( x \). The substitution \( x_1 \cdots x_n \langle x_1 := s_1, \ldots, x_n := s_n \rangle \) is the concatenation \( s_1 \cdots s_n \).

A list-algebra is a pair \( \langle X, \Pi : \text{list} X \to X \rangle \) where \( \Pi \) is a valuation function that assigns to each formal product \( x_1 \cdots x_n \) a value in \( X \). The laws of list-algebras are then

\[
\Pi(x) = x \\
\Pi(x_1 \cdots x_n \langle x_1 := s_1, \ldots, x_n := s_n \rangle) = \Pi((\Pi s_1) \cdots (\Pi s_n))
\]

Writing \( e \) for \( \Pi(\langle \rangle) \) and \( x \cdot y \) for \( \Pi(x, y) \), we obtain the following laws as a consequence of the corresponding properties for list concatenation:

\[
x \cdot e = x = e \cdot x \\
x \cdot (y \cdot z) = (x \cdot y) \cdot z
\]

Thus, list-algebras are precisely monoids \( \langle X, \cdot, e \rangle \) and morphisms are monoid morphisms. The free list-algebras are free monoids, i.e., lists with concatenation as the binary operation. The Kleisli category maps are functions of the form \( X \to \text{list} Y \).

One can similarly define a nonempty list monad \( \text{list}^+ : \text{Set} \to \text{Set} \). The algebras of this monad are semigroups \( \langle X, \cdot \rangle \).

There is a multiset monad \( \text{Set} \to \text{Set} \) with singleton multisets as the unit and multiset union as the multiplication. The following notation is useful in this context. Recall that a multiset over \( X \) is a function \( \phi : X \to \mathbb{N} \) that is “finitary” in the sense that \( \phi(x) = 0 \) for all but finitely many \( x \). The set of such finitary functions is called the “\( \mathbb{N}\)-span” of \( X \), denoted \( \mathbb{N}(X) \). It will be convenient to write functions \( \phi \) as finite formal sums \( k_1 x_1 + \cdots + k_n x_n \) where \( k_i \in \mathbb{N} \) and \( x_i \in X \). (The elements \( x \) not listed in the sum are understood to have coefficients 0.) In this notation, a singleton multiset is just \( x \) regarded as a formal sum and multiset union is:

\[
k_1(\sum_i l_i x_i) + \cdots + k_n(\sum_i l_i x_i) \Rightarrow \sum_i (\sum_{j=1}^n k_j l_{ji}) x_i
\]

where, on the right, the outer \( \sum \) is formal and the inner \( \sum \) is on naturals. These notations can be used for all \( K\)-span monads where \( K \) has addition.

The \( \mathbb{Z}\)-span monad \( \mathbb{Z}(\cdot) : \text{Set} \to \text{Set} \) has abelian groups as algebras.

**Example 3** Let \( M = \langle M, \cdot, e \rangle \) be a monoid. An action of \( M \) (on a set) is a pair \( \langle X, \alpha : X \times M \to X \rangle \) such that

\[
\alpha(x, e) = x \\
\alpha(x, ab) = \alpha(\alpha(x, a), b)
\]

A morphism of \( M\)-actions \( f : \langle X, \alpha \rangle \to \langle Y, \beta \rangle \) is a function such that

\[
f(\alpha(x, a)) = \beta(f(x), a)
\]
Thus, we have a category $M-\text{Act}$ of $M$-actions.

Actions of $M$ can be viewed as automata. $M$ consists of abstract “instruction sequences” for the automaton and $X$ is its state set. The action $\alpha$ is then a transition function that specifies the effect of running an instruction sequence $a$ in a state $x$.

$M$-actions can be viewed as algebras of a monad $− \times M : \text{Set} \to \text{Set}$. The unit $η_X : X \to X \times M$ of this monad sends $x$ to $(x, c)$ and the multiplication $μ_X : (X \times M) \times M \to X \times M$ sends $((x, a), b)$ to $(x, ab)$. The unit laws and the associativity law of the monoid follow from the unit laws and the associativity law of the monoid $M$.

$M$-actions are now precisely algebras of this monad.

Note that monoid is the minimum structure needed on $M$ to make a monad in this fashion. Semigroups won’t do, for example. The definitions of monads and monoids on the one hand, and algebras and monoid actions on the other, have a great deal of resemblance. The main difference is that monoid is a structure on sets while a monad is a structure on functors.

If $R$ is a ring, we can define a monad $R \otimes − : \text{Ab} \to \text{Ab}$ with unit $x \mapsto 1 \otimes x$ and multiplication $r_1 \otimes (r_2 \otimes x) \mapsto r_1 r_2 \otimes x$. The algebras of this monad are left $R$-modules.

Example 4 Consider the finite powerset monad $F : \text{Set} \to \text{Set}$ where $FX$ is the set of finite sets over $X$, the unit $η_X$ sends $x \in X$ to a singleton $\{x\} \in FX$, and the multiplication $μ_X : FFX \to FX$ is the union operation $\{a_1, \ldots, a_n\} \mapsto a_1 \cup \cdots \cup a_n$. So, an abstract term of $F$ is a finite set $\{x_1, \ldots, x_n\}$. A variable term is a singleton set and abstract substitution $\{x_1, \ldots, x_n\} \{x_1 := a_1, \ldots, x_n := a_n\}$ is the union $a_1 \cup \cdots \cup a_n$.

If $\langle X, α : FX \to X \rangle$ is an algebra for this monad, write $⊥$ for $α(∅)$ and $x ∨ y$ for $α\{x, y\}$. From the properties of finite sets, it follows that $∨$ is associative, commutative, idempotent and has $⊥$ as the unit. Thus, algebras of $F$ are pointed semilattices $(X, ∨, ⊥)$.

Defining $x \leq y \iff x ∨ y = y$, one verifies that $≤$ is a partial order. The element $⊥$ is the least element in this partial order and $∨$ is the join operation.

Similarly, the nonempty finite powerset monad $F^+ : \text{Set} \to \text{Set}$ gives rise to semilattices (with possibly no bottom elements).

Example 5 The powerset monad $P : \text{Set} \to \text{Set}$ is defined similarly, with the unit giving singletons and the multiplication giving unions.

The algebras of this monad are of the form $\langle X, ∨_X : PX \to X \rangle$ where the operation $∨$ assigns to every subset $a \subseteq X$, an element of $∨_X a \in X$. Reasoning as above, we can define a partial order $x \leq y \iff ∨_X \{x, y\} = y$ and notice that $∨$ is the supremum operation with respect to this order. Thus, the algebras of the powerset monad are complete semilattices. The morphisms are sup-preserving functions, also called additive functions. (Complete semilattices are also complete lattices, but their morphisms preserve only sups.)

The free algebras of the powerset monad are of the form $\langle PX, \cup \rangle$. They are complete lattices, but have additional structure: they are atomic and form boolean algebras. One calls them complete atomic boolean algebras (CABAs). However, the morphisms of $P$-algebras do not preserve this additional structure.

The Kleisli maps $f : X \to_K Y$ are functions $f : X \to PY$. Any such function $f$ is essentially a binary relation between $X$ and $Y$. Thus, the Kleisli category $\text{Set}_P$ is isomorphic to $\text{Rel}$, the category of sets and relations.

Example 6 The subsingletons monad $P^1 : \text{Set} \to \text{Set}$ assigns to a set $X$ the set of subsets of $X$ containing at most one element. Thus, an element of $P^1X$ is either $∅$ or a singleton $\{x\}$. The unit is $η_X(x) = \{x\}$ and the multiplication is $μ_X(∅) = μ_X(∅) = ∅$, $μ_X(\{x\}) = \{x\}$. All this is just a specialization of what was said above for the powerset monad.

Equivalently, this monad may be viewed as the lift monad $(-)_{∅} : \text{Set} \to \text{Set}$ which simply adds an additional element $∅ = ⊥$ to a given set $X$. 

6
The algebras of this monad are of the form $\langle X, \alpha \rangle$ whose algebras are pairs $\langle X, \alpha : X_\bot \to X \rangle$. Given that $\alpha(x)$ must be $x$ for all $x \in X$, all that is left to be specified is $\alpha(\bot)$. Thus, the algebras are pointed sets, i.e., sets $X$ with distinguished points $\bot$. The morphisms preserve these points (called strict functions). Incidentally, the free algebras are also pointed sets. So, the Kleisli category is equivalent to the Eilenberg-Moore category.

The Kleisli maps $f : X \to_K Y$ are functions $f : X \to Y_\bot$ which are essentially partial functions $f : X \to Y$. Thus, the Kleisli category is equivalent to $\mathbf{PfIn}$, the category of sets and partial functions.

Example 7 The category $\text{Poset}$ has posets for objects and monotone functions for morphisms. Consider the functor $\hat{\mathbf{P}} : \text{Poset} \to \text{Poset}$ that assigns to a poset $X$ the poset of all downward-closed subsets (downsets) of $X$, ordered by inclusion. This functor can be extended to a monad by associating the unit $\eta : x \mapsto \down{\{x\}}$ (where $\down a = \{ z \in X : \exists x \in a. z \leq x \}$) and the multiplication $\mu : a \mapsto \bigcup a$. This is called the downsets monad on $\text{Poset}$.

The algebras of this monad are of the form $\langle X, \bigvee_X : \hat{\mathbf{P}}X \to X \rangle$ which are again complete semilattices. To see that $\bigvee_X$ is a supremum operation, consider $\bigvee_X a$. If $x \in a$ then $\down{\{x\}} \subseteq \down a$ which implies $x \leq \bigvee_X a$. If $z$ is an upper bound of $a$, then $\down a \subseteq \down z$. So, $\bigvee_X a \subseteq z$. Thus, complete semilattices can be viewed as algebras in two different ways.

The free algebras are of the form $\langle \hat{\mathbf{P}}X, \bigcup \rangle$. These are complete lattices, but have additional structure. They are prime-algebraic with complete primes of the form $\down \{x\}$ for $x \in X$. (Note that, for every $a \in \hat{\mathbf{P}}X$, $a = \bigcup \{ \down \{x\} : x \in a \}$.)

The Kleisli maps $X \to \hat{\mathbf{P}}Y$ can be characterized as relations $F \subseteq X \times Y$ satisfying:

$$x' \geq_X x \land x \in X \Rightarrow y \in Y \land y' \Rightarrow x' \land F y \land y \geq Y y'$$

Other similar examples include

- the finite lower set monad $\hat{\mathbf{F}} : \text{Poset} \to \text{Poset}$ whose algebras are pointed join-semilattices,
- the nonempty finite lower set monad $\hat{\mathbf{F}}^+ : \text{Poset} \to \text{Poset}$ whose algebras are join-semilattices, and
- the subsingleton lower set monad $\hat{\mathbf{F}}^1 : \text{Poset} \to \text{Poset}$ whose algebras are pointed posets (posets with least elements).

In each case, the algebra maps preserve the stated structure (strict semilattice morphisms, semilattice morphisms, strict monotone functions respectively).

At this point, it is useful to relate various notions of “algebraicity” for complete semilattices or, more generally, for complete partial orders (cpo’s). Let $X$ be a cpo.

1. A compact element in $X$ is an element $k \in X$ such that for all directed sets $a \subseteq X$, $k \leq \bigvee a$ implies $k \leq x$ for some $x \in a$. $X$ is an algebraic cpo if every element $x \in X$ is the lub of the compact elements below it.

2. A complete prime in $X$ is an element $k \in X$ such that whenever $k \leq \bigvee a$ (for any $a$ with a lub), $k \leq x$ for some $x \in a$. $X$ is said to be prime-algebraic if every element $x \in X$ is the lub of the complete primes below it.

3. If $X$ is a pointed cpo with a least element $\bot_X$, an atom of $X$ is a complete prime $k$ that covers $\bot_X$, i.e., $k \neq \bot_X$ and $x \leq k \implies x = \bot_X \lor x = k$. $X$ is said to be atomic if every element is the lub of atoms below it.
Evidently, atoms are complete primes which are in turn compact elements. So, there are proper inclusions between atomic cpos, prime-algebraic cpos and algebraic cpos.

**Example 8** Consider the functor \( \hat{D} : \text{Poset} \rightarrow \text{Poset} \) that assigns to a partial order \( X \) the poset of all directed downsets (also called *ideals*) over \( X \), ordered by inclusion. The unit \( \eta_X \) sends \( x \) to the principal ideal \( \{ x \} = \{ y : y \leq x \} \) and the multiplication \( \mu_X \) sends a directed down-closed family \( a \) of ideals to its union. It may be verified that these data give a monad.

The algebras for the directed sets monad are complete partial orders or *cpo’s* (not necessarily pointed). They are posets \( X \) with an operation \( \bigvee : \hat{D}X \rightarrow X \) for lubs of directed downsets. The morphisms preserve directed sups (called *Scott-continuous functions*).

The free algebras for the directed sets monad are of the form \( \langle \hat{D}(X), \bigvee \rangle \). They are ideal completions of posets or, equivalently, *algebraic cpo’s*. (“Domains” are algebraic cpo’s with a countable basis.)

The Kleisli maps \( X \rightarrow_K Y \) are monotone functions \( X \rightarrow \hat{D}Y \) for posets \( X \) and \( Y \). Such maps are called *approximable maps* and characterized as relations \( F \subseteq X \times Y \) satisfying the following conditions:

1. For all \( x \in X \), there is \( y \in Y \) such that \( x \, F \, y \).
2. If \( x' \geq x \), \( x \, F \, y \) and \( y \geq y' \) then \( x' \, F \, y' \).
3. If \( x \, F \, y_1 \) and \( x \, F \, y_2 \), there exists \( y \geq y_1, y_2 \) such that \( x \, F \, y \).

In place of directed sets, one can use various kinds of subsets which, after adding downward-closure, give rise to monads. Here are some examples:

<table>
<thead>
<tr>
<th>Subset system</th>
<th>Condition on a subset ( a \subseteq X )</th>
<th>Algebras</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \omega )-chains</td>
<td>( a ) linearly ordered</td>
<td>( \omega )-cpos</td>
</tr>
<tr>
<td>bounded sets</td>
<td>( a ) has an upper bound</td>
<td>bounded complete cpos</td>
</tr>
<tr>
<td>consistent sets</td>
<td>every finite ( a' \subseteq a ) has an upper bound</td>
<td>consistently complete cpos</td>
</tr>
<tr>
<td>pairwise consistent sets</td>
<td>every pair in ( a ) has an upper bound</td>
<td>coherent cpos</td>
</tr>
</tbody>
</table>

The morphisms in each case preserve the lubs of appropriate subsets. (They are not merely Scott-continuous.) More on the structure of subset systems may be found in [Mes83].

**Example 9** A *web* \( A = (|A|, \sqsubseteq_A) \) is a set with a reflexive-symmetric binary relation called “coherence.” The morphisms of webs preserve this relation, i.e., \( x \sqsubseteq y \implies f(x) \sqsubseteq f(y) \).

This gives a category *Web*.

Define a functor \( \mathbb{P}_{coh} : \text{Web} \rightarrow \text{Web} \) which assigns to a web \( A \) the set of pairwise coherent subsets of \(|A|\) (or, simply, *coherent sets* of \( A \)). The coherence relation for \( \mathbb{P}_{coh}A \) is

\[
\begin{align*}
& a_1 \sqsubseteq a_2 \iff a_1 \cup a_2 \in \mathbb{P}_{coh}A \iff \forall x \in a_1. \forall y \in a_2. \ x \sqsubseteq_A y \\
& \text{This functor gets the structure of a monad by adding the unit of singletons and the multiplication of coherent unions. Thus, we obtain the coherent sets monad on webs.}
\end{align*}
\]

The free algebras of \( \mathbb{P}_{coh} \) are easy to recognize. They are of the form \( \langle \mathbb{P}_{coh}A, \bigcup^{coh} \rangle \) and are called *coherent spaces*. They are partially ordered by inclusion and form coherent cpos. In addition, they have the property of being *atomic*. A morphism of coherent spaces is determined by a web map \( f : A \rightarrow \mathbb{P}_{coh}B \) or, equivalently, a relation \( F \subseteq |A| \times |B| \) that preserves coherence.

A *partially ordered web*, or *poweb*, is a pair \( (|A|, \sqsubseteq_A) \) where \(|A|\) is a poset \( (|A|, \leq_A) \). This is subject to the axiom:

\[
x' \leq x \sqsubseteq y \geq y' \implies x' \sqsubseteq y'
\]
In particular, note that \( x \leq y \) implies \( x \sqsubseteq y \). Such ordered webs are also called prime event structures. The morphisms of powebs are monotone coherence-preserving functions. This gives a category \( \text{Poweb} \).

The coherent downsets monad \( \mathbb{P}_{\text{coh}} : \text{Poweb} \to \text{Poweb} \) assigns to a poweb \( A \) the set of coherent downsets of \( A \), ordered by inclusion. The coherence relation for \( \mathbb{P}_{\text{coh}}A \) is as for webs above.

The algebras of \( \mathbb{P}_{\text{coh}} \) are powebs \((|A|, \sqsubseteq_A)\) equipped with an operation \( \bigvee_A : \mathbb{P}_{\text{coh}}A \to A \) (which is a morphism of powebs). Now, we claim

\[
x \sqsubseteq y \iff x \uparrow y
\]

If \( x \sqsubseteq y \) then \( \bigvee\{x, y\} \) exists. So, \( x \uparrow y \). Conversely, if \( x \uparrow y \) then there is some \( z \) such that \( x \leq z \) and \( z \geq y \), and the condition of powebs implies that \( x \sqsubseteq y \). The algebras of \( \mathbb{P}_{\text{coh}} \) are coherent cpo’s, i.e., coherent cpo’s where every element \( x \sqsubseteq y \) and \( y \sqsupseteq y' \) implies \( x' \sqsubseteq y' \).

**Example 10** A consistency structure on a set \(|A|\) is a family of finite subsets \( \Gamma_A \) such that

1. \( x \in A \) implies \( \{x\} \in \Gamma_A \), and
2. \( a \in \Gamma_A \) and \( a' \subseteq a \) implies \( a' \in \Gamma_A \).

Note that consistency structures generalize webs by moving from binary relations to “multiary” relations. A morphism of consistency structures are functions \( f : A \to B \) such that \( a \in \Gamma_A \) implies \( f[a] = \{ f(x) : x \in a \} \in \Gamma_B \). Thus, we have a category \( \text{Con} \).

A subset \( a \subseteq |A| \) is said to be consistent if all its finite subsets are in \( \Gamma_A \). Let \( \mathbb{P}_{\text{con}}A \) denote the set of all consistent sets of \( A \). We can associate a consistency structure with \( \mathbb{P}_{\text{con}}A \) by defining \( \Gamma \) to be set of finite sets \( \{a_1, \ldots, a_n\} \) such that \( a_1 \cup \cdots \cup a_n \) is a consistent set. We have a functor \( \mathbb{P}_{\text{con}} : \text{Con} \to \text{Con} \) which extends to a monad.

The free algebras of this monad, of the form \( (\mathbb{P}_{\text{con}}A, \bigcup_{\text{con}}) \), are called qualitative domains. They are atomic, consistently complete cpos.

A partially ordered consistency structure is a pair \((|A|, \Gamma_A)\), where \(|A|\) is a poset, such that

\[
a \cup \{x\} \in \Gamma_A \land x \geq x' \implies a \cup \{x, y\} \in \Gamma_A
\]

Such structures are also called prime information systems. The morphisms are monotone consistency-preserving functions. This gives a category \( \text{Pocon} \).

There is a consistent downsets monad \( \mathbb{P}_{\text{con}} : \text{Pocon} \to \text{Pocon} \). Its algebras are partially ordered consistency structures \((|A|, \Gamma_A)\) equipped with an operation \( \bigvee_{\text{con}} : \mathbb{P}_{\text{con}}A \to A \). As for coherent downsets above, it is possible to argue that, for all finite sets \( a \subseteq |A| \),

\[
a \in \Gamma_A \iff a \uparrow
\]

Hence, the algebras are precisely consistently complete cpo’s. The free algebras are prime-algebraic, consistently complete cpo’s.
Example 11 The category \textbf{Preord} has objects preordered sets \( X = \langle X, \preceq_X \rangle \) and morphisms monotone functions. Given a preordered set \( X \), define an equivalence relation \( \approx_X \) by \( x \approx y \iff x \preceq y \land y \preceq x \). The quotient \( X/\approx_X \) is a partial order.

This gives a monad \( T: \textbf{Preord} \to \textbf{Preord} \) with \( TX = X/\approx_X \). Note that \( TTX \simeq TX \). A monad with this property is called an \textit{idempotent} monad. The unit \( \eta_X \) sends \( x \in X \) to the equivalence class \( [x] \in X/\approx_X \). The multiplication \( \mu_X \) is the isomorphism \( TTX \simeq TX \).

This example is interesting in that the unit \( \eta \) is not an embedding.

The notion of comonads and coalgebras is defined by duality. If \( C \) a category, a \textit{comonad} on \( C \) is a triple \( \mathcal{L} = \langle L, \epsilon, \delta \rangle \) of an endofunctor and two natural transformations as follows:

\[
\begin{array}{c}
L = LI \xleftarrow{\delta} L \xrightarrow{\epsilon} LL = L \\
L \xrightarrow{\delta} L \xleftarrow{\epsilon} L
\end{array}
\]

The natural transformations \( \epsilon \) and \( \delta \) are called the \textit{counit} and \textit{comultiplication} respectively. A \textit{coalgebra} for \( L \) is a pair \( \langle X, \alpha : X \to LX \rangle \) such that

\[
\begin{array}{c}
X \xleftarrow{\epsilon_X} LX \\
\alpha \downarrow \\
X
\end{array}
\quad
\begin{array}{c}
LX \xleftarrow{\delta_X} LX \\
\alpha \downarrow \\
LX
\end{array}
\]

Morphisms of coalgebras are defined in the expected fashion:

\[
\begin{array}{c}
LX \xrightarrow{Lh} LY \\
\alpha \downarrow \\
X \xrightarrow{h} Y
\end{array}
\quad
\begin{array}{c}
LX \xrightarrow{\beta} LY \\
\alpha \downarrow \\
X
\end{array}
\]

Example 12 Consider the category \textbf{CSLat} of complete semilattices and additive functions. Define a functor \( \mathbb{P}: \textbf{CSLat} \to \textbf{CSLat} \) that assigns to a complete semilattice \( X \) the complete semilattice \( \mathbb{P}X \) of all subsets of \( X \), ordered by inclusion. The counit \( \epsilon_X : \mathbb{P}X \to X \) sends a subset \( a \subseteq X \) to \( \bigvee a \). The comultiplication \( \delta_X : \mathbb{P}X \to \mathbb{P}P \mathbb{P}X \) sends \( a \) to the set of singletons \( \{ \{ x \} : x \in a \} \).

A coalgebra for \( \mathbb{P} \) is a pair \( \langle X, \alpha : X \to \mathbb{P}X \rangle \) of a complete semilattice \( X \) and an additive function \( \alpha \). The counit law for coalgebras means that \( \bigvee_X \alpha(x) = x \). So, \( \alpha(x) \) is a set of approximations of \( x \). The comultiplication law means that

\[
\{ \{ y \} : y \in \alpha(x) \} = \{ \alpha(y) : y \in \alpha(x) \}
\]

Thus, for all \( y \) in the image of \( \alpha \), \( \alpha(y) = \{ y \} \). Working this out, one finds that \( \alpha \) maps \( x \in X \) to the set of \textit{atoms} below \( x \). Now, the counit law \( \bigvee_X \alpha(x) = x \) says precisely that \( X \) is a complete \textit{atomic} lattice.
The morphisms $h : (X, \alpha) \to (Y, \beta)$ of coalgebras preserve the atoms, i.e.,

$$\beta(h(x)) = \{ h(a) : a \in \alpha(x) \}$$

In particular, if $x$ is an atom, $\alpha(x) = \{ x \}$. So, $h(x)$ is an atom as well. Thus, $h$ is completely determined by its action on the atoms.

Since complete atomic lattices are isomorphic to powersets $\mathbb{P}X$ with atoms $X$, and the morphisms are functions on the atoms, the category of coalgebras is equivalent to $\textbf{Set}$. 
2 Monads and adjunctions

The concept of monads is closely related to that of adjunctions, which is at the heart of category theory. We examine this connection.

An adjunction is a pair of functors between two categories

\[ C \xrightarrow{F} A \xleftarrow{G} \]

together with a natural isomorphism:

\[ \phi_{x,a} : A(Fx, a) \cong C(x, Ga) : \phi_{x,a}^{-1} \]  

(We are lower case letters for objects: \( x, y, z \) for the objects of \( C \) and \( a, b, c \) for the objects of \( A \).) In this situation, one writes \( F \dashv G : A \to C \) and calls \( F \) the left adjoint of \( F \) and \( G \) the right adjoint of \( F \). Each functor in the adjunction determines the other uniquely up to isomorphism.

Typically, \( A \) is a category of some “structures” over the objects of \( C \). Then, \( F \) is the “free” functor that assigns to an object \( x \) of \( C \), the free structure over \( x \). The functor \( G \) is a “forgetful” functor that assigns to a structure \( a \), the underlying object of \( a \) (“forgets” the structure of \( a \)).

For instance, consider the adjunction between sets and monoids:

\[ \text{Set} \xrightarrow{F} \text{Mon} \]

where \( F \) is the free monoid functor \( X \mapsto \langle \text{list } X, \cdot, \epsilon \rangle \) and \( G \) is the underlying set functor \( \langle A, \cdot_A, e_A \rangle \mapsto A \). We have a natural isomorphism:

\[ \phi_{X,A} : \text{Mon}(FX, A) \cong \text{Set}(X, GA) : \phi_{X,A}^{-1} \]

which means that the set of monoid morphisms \( FX \to A \) is (naturally) one-to-one with the set of functions \( X \to GA \). Let us see the details. Given a function \( f : X \to A \), there exists a unique monoid morphism \( h = \phi_{X,A}^{-1}(f) : \langle \text{list } X, \cdot, \epsilon \rangle \to \langle A, \cdot_A, e_A \rangle \) such that the restriction of \( h \) to \( X \) is \( f \). It is given by

\[ h(x_1, \ldots, x_n) = f(x_1) \cdot_A \cdots \cdot_A f(x_n) \]

Note that the composite \( GF : \text{Set} \to \text{Set} \) assigns to a set \( X \) the set of lists over \( X \) (the underlying set of the free monoid). We know that this functor extends to a monad. This happens in general.

Whenever \( F \dashv G : C \to A \) is an adjunction, the composite functor \( T = GF : C \to C \) gives a monad on \( C \). Dually, the composite functor \( L = FG : A \to A \) gives a comonad on \( A \). This situation may be pictured as follows:

\[ \begin{array}{ccc}
C & \xrightarrow{F} & A \\
\downarrow & & \downarrow \\
T = GF & \to & L = FG \\
\text{(monad)} & & \text{(comonad)}
\end{array} \]

Let us see the details of this phenomenon. First, recall that there is a notion of a unit of an adjunction. By setting \( a = Fx \) in (3), we get a family of arrows

\[ \eta_x = \phi_{x,a}(\text{id}_{Fx}) : x \to C GfX \]
where are easily seen to be natural in $x$. We take this to be the unit of the monad $T$. Dually, there is a counit for the adjunction obtained by setting $x = Ga$ in (3):

$$\varepsilon_a = \phi_{x,a}^{-1}(\text{id}_{Ga}) : FGa \to A \ a$$

We take this to be the counit of the comonad $L$. For the multiplications and the comultiplications, respectively, of $T$ and $L$, we define

$$\mu = G\varepsilon F : GFG\overset{\sim}{\to} GF$$
$$\delta = F\eta G : FGFG\overset{\sim}{\to} FG$$

These data do give monads and comonads . . .

It is reported that Huber [Hub61] proved this result because “he was having so much trouble demonstrating that the associative identity was satisfied” and noticed that all his monads were associated with adjunctions. Indeed, it simplifies our work considerably to look for an adjunction that induces a monad to obtain the relevant data of the monad. The following example would be quite tricky to do without the use of adjunctions.

**Example 13** A meet-semilattice is a pair $(A, \land_A)$ where $\land_A$ is a meet operation (associative, commutative and idempotent binary operation). The morphisms preserve meets. This gives a category $\text{MSLat}$. We claim that there is an adjunction between posets and meet-semilattices $\text{Poset} \xarr{F} \xarr{G} \text{MSLat}$. This means there is a natural isomorphism:

$$\phi_{X,A} : \text{MSLat}(FX, \langle A, \land_A \rangle) \cong \text{Poset}(X, G\langle A, \land_A \rangle)$$

Take $G$ to be the forgetful functor that maps $(A, \land_A)$ to the underlying poset $(A, \leq_A)$. The partial order of $A$ is defined by $x \leq_A y \iff x \land_A y = x$. For the free functor $F$, define the upper set of $a \subseteq X$ by $\uparrow_X a = \{ z \in X : \exists x \in a, z \geq_X x \}$. Then, we can take $FX$ to be the meet-semilattice $\langle \hat{F}^+X, \cup \rangle$ where $\hat{F}^+X$ is the set of upper sets of nonempty finite subsets of $X$. Note that the union of upper sets is an upper set. The induced partial order on $\hat{F}^+X$ is reverse inclusion, $\supseteq$.

The natural isomorphism $\phi$ maps a meet-semilattice homomorphism $h : \langle \hat{F}^+X, \cup \rangle \to (A, \land_A)$ to its restriction $f : X \to A$ given by $f(x) = h(\uparrow\{x\})$. Note that $f$ is monotone: $x \leq y \implies \uparrow\{x\} \supseteq \uparrow\{y\} \implies h(\uparrow\{x\}) \leq h(\uparrow\{y\})$. Any such function $f$ uniquely extends to a homomorphism $h(\uparrow a) = \bigwedge\{ f(x) : x \in a \}$.

The adjunction induces a monad on $\text{Poset}$, the nonempty finite upper set monad $\hat{F}^+ : \langle X, \leq_X \rangle \mapsto \langle \hat{F}^+X, \supseteq \rangle$. We obtain the relevant data from the adjunction. The unit $\eta_X : X \to \hat{F}^+X$ is $\phi(\text{id}_{FX})$. It is given by

$$\eta_X(x) = \text{id}(\uparrow\{x\}) = \uparrow\{x\}$$

The counit of the adjunction $\varepsilon_A : \langle \hat{F}^+GA, \cup \rangle \to A$ is $\phi^{-1}(\text{id}_{GA})$. It is given by

$$\varepsilon_A(\uparrow a) = \bigwedge_A \{ \text{id}_{GA}(x) : x \in a \} = \bigwedge_A a$$

So, the multiplication of the monad $\mu_X : \hat{F}^+\hat{F}^+X \to X$ is $G\varepsilon_{FX}$. Since $\hat{F}^+X$ is ordered by reverse inclusion, an upper set of a finite set $s$ is as actually downset. Denote it by $\downarrow s = \{ a : \exists a' \in s, a \subseteq s \}$. Then $\mu_X(\downarrow s) = \bigcup s$. This completes the definition of the $\hat{F}^+$ monad. Note that the algebras of this monad are precisely meet-semilattices.

Other similar examples include
• the **subsingleton upper set** monad \( \mathcal{F}^1 \) whose algebras are “topped” posets (posets with top elements),

• the **finite upper set** monad \( \mathcal{F} \) whose algebras are “topped” meet-semilattices

• the **upset** monad \( \mathcal{P} \) whose algebras are inf-complete semilattices.

We see that the intuitive picture given at the beginning of this section is needlessly one-sided. Not only may we regard the objects of \( A \) as “structures” over the objects of \( C \), but we may also regard the objects of \( C \) as “co-structures” over the objects of \( A \). By “co-structure” here, we mean the structure of coalgebras with respect to the comonad \( L \).

For the case of the set-monoid adjunction this means that sets are obtainable by adding “co-structure” to monoids. Let us see how. Call a monoid \( A = \langle A, \cdot_A, e_A \rangle \) **atomic** if there is a subset \( K \subseteq A \) such that every \( x \in A \) can be written uniquely as a product \( x = k_1 \cdots k_n \) of elements in \( K \). (The elements of \( K \) are the **atoms** of \( A \).) A morphism of atomic monoids \( h : \langle A, K \rangle \to \langle B, L \rangle \) is a morphism of monoids such that, whenever \( x = k_1 \cdots k_n \) is an atomic factorization in \( A \), \( h(x) = h(k_1) \cdots h(k_n) \) is an atomic factorization in \( B \). It is clear that atomic monoids \( \langle A, K \rangle \) are essentially free monoids generated by \( K \) and that atomic monoid morphisms are essentially functions between the sets of atoms. The category of atomic monoids is equivalent to \( \textbf{Set} \) which is obtained by adding “co-structure” to monoids.

Now, consider the comonad \( L = FG : \textbf{Mon} \to \textbf{Mon} \). It assigns to each monoid \( A \) the monoid \( \text{list} A \). The counit \( \epsilon_A : \text{list} A \to A \) is just multiplication \( \langle x_1, \ldots, x_n \rangle \mapsto x_1 \cdots x_n \). The comultiplication \( \delta_A : \text{list} A \to \text{list list} A \) maps a list \( \langle x_1, \ldots, x_n \rangle \) to the list of singletons \( \langle \langle x_1 \rangle, \ldots, \langle x_n \rangle \rangle \). Coalgebras of this comonad are pairs \( \langle A, \alpha : A \to \text{list} A \rangle \) of monoids \( A \) and structure maps \( \alpha \). It may be verified that they are precisely atomic monoids with \( \alpha \) mapping each element \( x \in A \) to the lists of its atomic factors.

The situation we found here is rather special. We started with an adjunction \( \textbf{Set} \underbrace{\xrightarrow{F} \textbf{Mon}}_{G} \) and found an induced monad \( T = GF \) on \( \textbf{Set} \) and an induced comonad \( L = FG \) on \( \textbf{Mon} \).

• From \( T \), we can form a category of algebras \( \textbf{Set}^T \) which turned out to be equivalent to \( \textbf{Mon} \). In this situation, we say that the adjunction is “monadic.”

• From \( L \), we can form a category of coalgebras \( \textbf{Mon}^L \) which turned out to be equivalent to \( \textbf{Set} \). In this situation, we say that the adjunction is “comonadic.”

If an adjunction is both monadic and comonadic, we say it is “bimonadic.”

The adjunction between sets and complete semilattices

\[
\textbf{Set} \underbrace{\xrightarrow{F} \textbf{CSLat}}_{G}
\]

is also bimonadic. The induced monad on \( \textbf{Set} \) is the powerset monad \( \mathcal{P} \) (Example 5). The induced comonad on \( \textbf{CSLat} \) is the powerlattice comonad (Example 12). The algebras of the powerset monad are precisely complete semilattices, and the coalgebras of the powerlattice comonad on \( \textbf{CSLat} \) are atomic semilattices which form a category equivalent to \( \textbf{Set} \).

The co-Kleisli categories of such induced comonads are extremely interesting, and found throughout programming language theory. For example, the co-Kleisli category of the powerlattice comonad has objects complete lattices and morphisms additive functions of the form \( f : \mathcal{P}A \to B \). Since additive functions preserve unions in \( \mathcal{P}A \), \( f \) is uniquely determined by its action on singletons. The action on singletons can be described by a function \( \hat{f} : A \to B \).
So, the co-Kleisli category is the category of complete lattices with all functions as arrows. Similarly, the co-Kleisli category of the list comonad on \textbf{Mon} is the category of monoids with all functions as arrows.

The following example gives several applications of this feature in semantics.

\textbf{Example 14} There is an adjunction

\[
\begin{array}{ccc}
\text{Cpo} & \xrightarrow{F} & \text{Cpo}_\bot \\
\xleftarrow{G} & & \end{array}
\]

between the category of cpo’s and continuous functions and that of pointed cpo’s and strict continuous functions. (A \textit{pointed} cpo is a cpo with a least element \(\bot\).) If \(X = (X, \bigvee_X)\) is a cpo, let \(X_\bot\) denote the lifted cpo with an additional least element \(\bot\). The functor \(F : \text{Cpo} \to \text{Cpo}_\bot\) assigns to each cpo \(X\) the pointed cpo \((X_\bot, \bot)\) obtained by lifting. To each continuous function \(f : X \to Y\) it assigns the evident strict extension \(f_\bot : (X_\bot, \bot) \to (Y_\bot, \bot)\). The functor \(G\) gives the underlying cpo of a pointed cpo.

The monad induced by the adjunction is \textit{lifting} \((-)_\bot : \text{Cpo} \to \text{Cpo}_\bot\) with unit and multiplication given by \(\ldots\). Evidently, the algebras of the monad are precisely pointed cpo’s with strict continuous functions as morphisms.

The comonad induced by the adjunction is also lifting \((-)_\bot : \text{Cpo}_\bot \to \text{Cpo}_\bot\) which sends a pointed cpo \((A, \bot_A)\) to \((A_\bot, \bot)\). The counit and the comultiplication are given by \(\ldots\). A coalgebra of this comonad is a pointed cpo \((A, \bot_A)\) equipped with a structure map \(\alpha : (A, \bot_A) \to (A_\bot, \bot)\) which just “picks out” the bottom element \(\bot_A\). A morphism of coalgebras preserves the operation of “picking out” the bottom elements, i.e., it is a \(\bot\)-reflecting function. Such morphisms are essentially continuous functions between cpos \(A \setminus \{\bot_A\} \to B \setminus \{\bot_B\}\). So, the adjunction between cpos and pointed cpos is “bimonadic.”

The co-Kleisli category of the comonad has maps, strict continuous functions \(A_\bot \to B\) between pointed cpo’s. Since the least element of \(A_\bot\) is always mapped to \(\bot_B\), this is determined by a continuous function \(A \to B\). Thus, the co-Kleisli category has pointed cpo’s for objects but all continuous functions for morphisms. We often use the symbol ! for such a comonad when the co-Kleisli category is cartesian closed (with inspiration from linear logic).

Many other structures over \textbf{Cpo} have such ! comonads. The category of coherent cpos \textbf{CohCpo} has objects coherent cpos (i.e., cpo’s where pairwise compatible subsets have lubs). The morphisms are additive functions that preserve the lubs of pairwise compatible subsets. The adjunction between \textbf{Cpo} and \textbf{CohCpo} gives rise to a comonad ! on \textbf{CohCpo} whose co-Kleisli category has coherent cpo’s but all continuous functions. The category of bounded-complete cpos \textbf{BcCpo} similarly has a ! comonad whose co-Kleisli category has all continuous functions between bounded-complete cpo’s.
3 Monad morphisms

If $S$ and $T$ are monads on $C$, a morphism of monads $\sigma : S \to T$ is a natural transformation $S \to T$ that commutes with the units and multiplications in the evident fashion:

$$
\begin{align*}
SS & \xrightarrow{\sigma \sigma} TT \\
\mu^S & \downarrow \quad \downarrow \mu^T \\
S & \xrightarrow{\sigma} T \\
\eta^S & \downarrow \quad \downarrow \eta^T \\
I & = I
\end{align*}
$$

We call $\sigma$ a monomorphism (epimorphism) of monads if each component $\sigma_X$ is a mono (epi). The notions of submonads and quotient monads are defined correspondingly.

In set-theoretic terms, $S$ and $T$ generate sets of abstract terms. A morphism $\sigma : S \to T$ then maps $S$-terms to $T$-terms in such a way that variable terms and substitutions are preserved. For example, there is a monad morphism $F^+ \to F$ (in fact, a monomorphism) that injects nonempty finite sets $F^+X$ to finite sets $FX$. So, $F^+$ is a submonad of $F$. There is a monad morphism $\text{list} \to F$ (in fact, an epimorphism) that maps lists $\text{list}X$ to finite sets of their elements $FX$. So, $F$ is a quotient monad of $\text{list}$.

If $\sigma : S \to T$ is a morphism of monads, it induces a functor $C^\sigma : C^T \to C^S$ between the corresponding categories of algebras. (Note the reversal of direction.) $C^\sigma$ assigns to a $T$-algebra $\alpha : TX \to X$ the $S$-algebra $\sigma_X ; \alpha : SX \to TX \to X$. Its morphism part assigns to a morphism $f : \langle X, \alpha \rangle \to \langle Y, \beta \rangle$ of $T$-algebras, the same function $f$ regarded as a morphism of $S$-algebras. That $f$ is indeed a morphism of $S$-algebras is evidenced by

$$
\begin{align*}
SX & \xrightarrow{Sf} SY \\
\sigma_X & \downarrow \quad \downarrow \sigma_Y \\
TX & \xrightarrow{Tf} TY \\
\alpha & \downarrow \quad \downarrow \beta \\
X & \xrightarrow{f} Y
\end{align*}
$$

where the top square is an instance of naturality and the bottom square commutes because $f$ is a morphism of $T$-algebras. At the level of categories, this situation is depicted by:

$$
\begin{align*}
C^T & \xrightarrow{C^\sigma} C^S \\
G^T & \downarrow \quad \downarrow G^S \\
C & = C
\end{align*}
$$

For example, if $\sigma : F^+ \to F$ is the injection then $C^\sigma : C^F \to C^{F^+}$ is the forgetful functor that forgets the bottom elements of semilattices. For $\sigma : \text{list} \to F$, the induced functor $C^\sigma : C^\text{list} \to C^\text{list}$ is the inclusion of semilattices in $\text{Mon}$.

More interestingly, functors of this form are all induced by monad morphisms. More precisely, if $G : C^T \to C^S$ is a functor such that

$$
\begin{align*}
C^T & \xrightarrow{G} C^S \\
G^T & \downarrow \quad \downarrow G^S \\
C & = C
\end{align*}
$$
then $G = C^\sigma$ for a monad morphism $\sigma : S \to T$. To recover $\sigma$ from $G$, consider the action of $G$ on the free algebra $\mu_X^T : TTX \to TX$. The image of $\mu_X^T$ is an $S$-algebra structure on $TX$ (since $G$ preserves the underlying objects). Denote it by $\bar{\mu}_X^T : STX \to TX$.

Since $G$ also preserves the underlying arrows of morphisms, we have

$$G : \begin{pmatrix} TTX \\ \mu_X^T \\ TX \end{pmatrix} \xrightarrow{Tf} \begin{pmatrix} TTY \\ \mu_Y^T \\ TY \end{pmatrix} \xrightarrow{\bar{\mu}_X^T} \begin{pmatrix} STX \\ \bar{\mu}_X^T \\ TX \end{pmatrix} \xrightarrow{Tf} \begin{pmatrix} STY \\ \bar{\mu}_Y^T \\ TY \end{pmatrix}$$

for all morphisms $h$. Now, for any arrow $f : X \to Y$ in $C$, $Tf : TX \to TY$ is a morphism of $T$-algebras. So, by setting $h = Tf$, we find the commutative square:

$$\begin{array}{ccc} STX & \xrightarrow{STf} & STY \\ \downarrow \bar{\mu}_X^T & & \downarrow \bar{\mu}_Y^T \\ TX & \xrightarrow{Tf} & TY \end{array}$$

which just says that $\bar{\mu}_X^T : ST \to T$ is a natural transformation.

Define $\sigma = S\eta^T; \bar{\mu}_X^T : S \to ST \to T$. A little diagram chasing shows that it is a monad morphism. Now, $C^\sigma \langle X, \alpha \rangle = \langle X, \sigma_X^X; \alpha \rangle$. . .

**Example 15** Let $A$ and $B$ monoids, with induced free action monads $- \times A, - \times B : \text{Set} \to \text{Set}$. Monad morphisms $- \times A \to - \times B$ are of the form $- \times h : - \times A \to - \times B$ where $h : A \to B$ is a morphism of monoids. Any such morphism induces a functor from $B$-actions to $A$-actions, viz., $h - \text{Act} = \text{Set} - \times h : B - \text{Act} \to A - \text{Act}$ defined by $h - \text{Act}(\beta : X \times B \to B) = (\alpha : X \times A \to A)$ where $\alpha(x, a) = \beta(x, h(a))$.

Thinking of $\beta$ is an automaton with instruction sequences $B$, $h - \text{Act}(\beta)$ simulates an $A$-automaton using $\beta$ by translating every $A$-instruction sequence $a$ to $h(a)$.

**Example 16** The monomorphism $\sigma : \mathbb{F}^+ \to \mathbb{F}$ induces a functor $\text{Set}^\sigma : \text{SLat}_* \to \text{SLat}$ which just forgets the least elements of pointed semilattices. Similarly, $\mathbb{F}$ is a submonad of $\mathbb{P}$, $\check{\mathbb{D}}$ is a submonad of $\check{\mathbb{P}}$ etc. The induced functors are forgetful. More interestingly, the list monad and the free abelian group monad $\mathbb{Z}(\_)$ are both submonads of the free ring monad. The induced functors forget part of the structure of a ring.

The epimorphism $\sigma : \text{list} \to \text{Fin}$ induces a functor $\text{Set}^\sigma : \text{SLat}_* \to \text{Mon}$ which regards a pointed semilattice as a monoid (there by ignoring the fact that $\lor$ is commutative and idempotent).
4 Distributive laws and composite monads

Suppose we are given two monads \( S, T : C \to C \). Does their composite \( TS : C \to C \) form a monad? Treating this entirely formally, we need a unit \( \eta : I \to TS \) and a multiplication \( \mu : TSTS \to TS \). The unit can be defined as the composite \( \eta = \eta^T \eta^S : I \to TS \). However, there is no immediate way to define multiplication. The composition \( \mu^T \mu^S : TTSS \to TS \) has the source \( TTSS \) which does not match the required \( TSTS \). But, all is not lost. If we have a natural transformation \( \lambda : ST \to TS \), we can complete the definition of \( \mu \) as \( T\lambda S \). The natural transformation \( \lambda \) (with a suitable axiomatization) is called a “distributive law” for \( S \) over \( T \). Conversely, if \( TS \) is indeed a monad with a multiplication \( \mu : TSTS \to TS \), we can produce a distributive law \( \lambda : ST \to TS \) by the composition

\[
ST = ISTI \xrightarrow{\eta^TST\eta^S} TSTS \xrightarrow{\mu} TS
\]

To get some intuition for this notion, treat \( S \) and \( T \) as assigning to a set \( X \) two collections of terms \( SX \) and \( TX \). The set \( TSX \) then contains terms formed from \( T \) with subterms formed from \( S \). Regard \( T \)-terms as formal “summations” \( \sum_i x_i \) and \( S \)-terms as formal “products” \( \prod_j y_j \). A \( T \)-term is then a sum-of-products \( \sum_i \prod_j x_{ij} \). A “distributive law” transforms a product-of-sums term to a sum-of-products term as in

\[
\prod_{i \in I} \sum_{j \in J(i)} x_{ij} \xrightarrow{\lambda} \sum_{f \in IJ} \prod_{i \in I} x_{1f(i)}
\]

where \( IJ \) stands for the set of “dependent” functions mapping indices \( i \in I \) to indices \( j \in J(i) \).

The “distributive law” (better called a “distribution operator”) distributes multiplication over addition. For example, the term \( (x_1 + x_2) \cdot (y_1 + y_2) \) gets mapped to the term \( x_1y_1 + x_1y_2 + x_2y_1 + x_2y_2 \). The laws associated with the distribution operator will ensure that the two terms are evaluated in the same way.

To motivate these laws, we use the term notation of Section 1 and assume a distribution operator \( \lambda \) that converts an \( ST \)-term of the form \( s[t] \) to a \( TS \)-term of the form \( t'[s'] \). We write this as \( \lambda : s[t] \mapsto t'[s'] \). (There does not seem to be a way to represent this operator implicitly in the syntax.) We need to specify what happens if \( s \) or \( t \) is a variable term or a substitution term:

\[
\begin{align*}
\lambda : s[\bullet] & \mapsto t[\bullet] \\
\lambda : s[t_1] & \mapsto t'_1[s'] \\
\lambda : s[t_1(t_2)] & \mapsto t'_1(t'_2)[s'']
\end{align*}
\]

The first two rules say that the action of \( \lambda \) on variable terms is null. The next two rules say that distribution commutes with substitution.

The following is a categorical statement of these ideas:

**Definition 17** If \( S = \langle S, \eta^S, \mu^S \rangle \) and \( T = \langle T, \eta^T, \mu^T \rangle \) are monads on a category \( C \), a distributive law of \( S \) over \( T \) is a natural transformation \( \lambda : ST \to TS \) such that

\[
\begin{align*}
STT \xrightarrow{\lambda T} TST & \xrightarrow{T\lambda} TTS \\
S\mu^T \xrightarrow{\lambda} TS & \xrightarrow{\mu^T S} TS
\end{align*}
\]

\[
\begin{align*}
SST \xrightarrow{S\lambda} STS & \xrightarrow{\lambda S} TSS \\
\mu^ST \xrightarrow{\lambda} TS & \xrightarrow{T\mu^S} TS
\end{align*}
\]
Lemma 18 If $S$ and $T$ are monads on $C$ with a distributive law $\lambda : ST \Rightarrow TS$, then the functor $TS : C \to C$ extends to a composite monad $(TS)_\lambda$ with unit and multiplication given by

\[
\eta = \eta^T \eta^S : I = II \to TS \\
\mu = T\lambda S; \mu^T \mu^S : TSTS \to TTSS \to TS
\]

The proof is a straightforward diagram chasing. Note that $\eta^T \eta^S$ refers to one of the two equal composites:

Likewise for $\mu^T \mu^S$.

The natural transformations $\eta^T S : S \Rightarrow TS$ and $T\eta^S : T \Rightarrow TS$ above are monad morphisms. The relevant diagrams are:

This fact will prove important when we consider algebras of composite monads.

Proposition 19 Suppose $S$, $T$ and $TS$ are monads on $C$ such that

1. $\eta = \eta^T \eta^S$,

2. $\eta^TS : S \Rightarrow TS$ and $T\eta^S : T \Rightarrow TS$ are monad morphisms, and

3. the “middle unitary law”

\[
TS \xrightarrow{T\eta^S \eta^T S} TSTS \xleftarrow{\mu} TS
\]

holds,

then the composite

\[
\lambda = ST \xrightarrow{\eta^T S \eta^T S} TSTS \xrightarrow{\mu} TS
\]

is a distributive law.

Example 20 Consider the monads $\mathbb{P}^1, \text{list}^+ : \text{Set} \to \text{Set}$. In this example, we write $\mathbb{P}^1 X$ as $X_\epsilon = \{\epsilon\} + X$. The composite functor $\mathbb{P}^1 \text{list}^+ : \text{Set} \to \text{Set}$ is isomorphic to $\text{list} : \text{Set} \to \text{Set}$.
which extends to a monad. So, one would expect to find a distributive law \( \lambda_X : \text{list}^+\mathcal{P}^1X \to \mathcal{P}^1\text{list}^+X \). It is given by

\[
\begin{align*}
\langle \epsilon \rangle & \mapsto \epsilon \\
\langle x_1, \ldots, x_n \rangle & \mapsto \langle x_{i_1}, \ldots, x_{i_k} \rangle
\end{align*}
\]

where \( x_{i_1}, \ldots, x_{i_k} \) is the sequence of non-\( \epsilon \) elements of \( \langle x_1, \ldots, x_n \rangle \).

Put another way, think of \( \text{list}^+X \) as the infinite coproduct \( X + X^2 + X^3 + \cdots \). Then \( \text{list}^+ (\mathcal{P}^1X) \) is

\[
(1 + X) + (1 + X)^2 + (1 + X)^3 + \cdots
\]

\[
= 1 + X + 1 + X + X + X^2 + 1 + X + X + X^2 + X^2 + X^2 + X^3 + \cdots
\]

while \( \mathcal{P}^1 (\text{list}^+ (X)) \) is \( 1 + X + X^2 + X^3 + \cdots \). The distributive law \( \lambda_X \) sends each summand of the source space to the same summand in the target space.

The induced monad morphisms \( \mathcal{P}^1 \to \text{list} \) and \( \text{list}^+ \to \text{list} \) are evident monics. The corresponding functors \( \text{Mon} \to \text{Set}_\perp \) and \( \text{Mon} \to \text{SGrp} \) are forgetful.

**Example 21** Consider the monads \( \mathcal{F}^1 \) and \( \mathcal{F}^+ \) on \( \text{Set} \), whose algebras are pointed sets and semilattices respectively. The functor composition \( \mathcal{F}^1 \mathcal{F}^+ : \text{Set} \to \text{Set} \) is isomorphic to \( \mathcal{F} : \text{Set} \to \text{Set} \) which extends to a monad. The distributive law \( \lambda_X : \mathcal{F}^+ \mathcal{F}^1X \to \mathcal{F}^1 \mathcal{F}^+X \) is given by

\[
\begin{align*}
\{ \varnothing \} & \mapsto \varnothing \\
\{ \varnothing \} & \mapsto \{ \{ x : \{ x \} \in a \} \}
\end{align*}
\]

The same arguments apply to the composition \( \mathcal{F}^1 \mathcal{F}^+ : \text{Set} \to \text{Set} \) which forms a monad isomorphic to the powerset monad \( \mathcal{P} \) on \( \text{Set} \).
Suppose \( \alpha : TSX \to X \) is a \( TS \)-algebra. Since we have monad morphisms \( S \to TS \) and \( T \to TS \), \( X \) has both a \( T \)-algebra structure and an \( S \)-algebra structure (which we denote by \( \alpha^T \) and \( \alpha^S \) respectively):

\[
\begin{array}{ccc}
SX & \xrightarrow{\eta_S X} & TSX \\
\downarrow{\alpha^S} & & \downarrow{\alpha_X} \\
X & & TX \\
\end{array}
\]

(5)

In other words, from the valuation map for \( TS \)-terms, we can extract a valuation map \( \alpha^S \) for \( S \)-terms and another \( \alpha^T \) for \( T \)-terms.

One would expect that these two evaluation maps compose to give back the original map:

\[
\begin{array}{ccc}
TX & \xrightarrow{T \alpha^S} & TSX \\
\downarrow{\alpha} & & \downarrow{\lambda_X} \\
TX & & TX \\
\end{array}
\]

(6)

This is indeed the case as may be verified by diagram chasing. That is, given a \( TS \)-term (a \( T \)-term with \( S \)-subterms), its evaluation by \( \alpha \) can be decomposed into the evaluation of \( S \)-subterms followed by the evaluation of the \( T \)-term.

Moreover, \( \alpha^S \) is “\( \lambda \)-distributive” over \( \alpha^T \):

\[
\begin{array}{ccc}
STX & \xrightarrow{\lambda_X} & TSX \\
\downarrow{S \alpha^T} & & \downarrow{T \alpha^S} \\
SX & & TX \\
\end{array}
\]

(7)

For example, if \( \lambda \) is the distribution operator (4) of products over sums, this says that the innermost evaluations of \( \prod_{i \in I} \sum_{j \in J(i)} x_{ij} \) and \( \sum_{f \in \Pi J} \prod_{i \in I} x_{i f(i)} \) must be equal. This corresponds to the usual distributivity law:

\[(x_1 + x_2) \cdot (y_1 + y_2) = x_1y_1 + x_1y_2 + x_2y_1 + x_2y_2\]

of multiplication over addition in rings. Another way to think about this identity is that multiplication is a homomorphism over addition (in both positions; so it is a “bimorphism”).

**Proposition 22** Let \( T \) be \( S \) monads on \( C \) with a distributive law \( \lambda : ST \to TS \). If \( TSX \xrightarrow{\alpha} X \) is an algebra for a composite monad \( TS \), then \( SX \xrightarrow{\alpha^S} X \) and \( TX \xrightarrow{\alpha^T} X \) defined by (5) are algebra structures on \( X \) that satisfy (6) and (7). Conversely, if \( SX \xrightarrow{\alpha^S} X \) and \( TX \xrightarrow{\alpha^T} X \) are algebra structures on \( X \) satisfying (7), then \( TSX \xrightarrow{\alpha} X \) defined by (6) is an algebra of the composite monad.

The algebras of composite monads may thus be called “distributive algebras.” We give several examples of such algebras below.

**Example 23** Consider the monads \( \hat{F}, \hat{D} \) on \textbf{Poset}. The composition of the functors \( \hat{D} \hat{F} : \textbf{Poset} \to \textbf{Poset} \) is isomorphic to \( \hat{P} : \textbf{Poset} \to \textbf{Poset} \). The isomorphism \( \sigma_X : \hat{P}X \sim \hat{D} \hat{F}X \) is:

\[\{a \mapsto \downarrow\{\downarrow a' : a' \leq_{\text{fin}} a\}\}\]
with the inverse being directed union. This isomorphism corresponds to the fact a cpo is a complete lattice if and only if it has finite joins.

There is a distributive law \( \lambda_X : \mathcal{F}\mathcal{D}X \to \mathcal{D}\mathcal{F}X \) given by

\[
\downarrow\{\downarrow d_1, \ldots, \downarrow d_n\} \mapsto \downarrow\{\downarrow\{x_1, \ldots, x_n\} : x_1 \in d_1, \ldots, x_n \in d_n\}
\]

\[
\ldots
\]

**Example 24** Consider the monads \( \hat{\mathcal{F}} \), \( \hat{\mathcal{F}} \) on \( \text{Poset} \), whose algebras are (pointed) join-semilattices and (topped) meet-semilattices respectively. We have a distributive law \( \lambda_X : \hat{\mathcal{F}}\hat{\mathcal{F}}X \to \hat{\mathcal{F}}\hat{\mathcal{F}}X \) given by

\[
\hat{\uparrow}\{\hat{\uparrow} a_1, \ldots, \hat{\uparrow} a_n\} \mapsto \hat{\uparrow}\{\hat{\uparrow}\{x_1, \ldots, x_n\} : x_1 \in a_1, \ldots, x_n \in a_n\}
\]

The upper set of \( \{\downarrow a_1, \ldots, \downarrow a_n\} \) is closure under supersets (denoted by \( \hat{\uparrow}\)). On the right, \( \hat{\mathcal{F}}X \) is ordered by reverse inclusion. So, a lower set by this order is also closure under supersets. This gives rise to a composite monad \( \hat{\mathcal{F}}\hat{\mathcal{F}} \). The unit \( \eta_X : X \to \hat{\mathcal{F}}\hat{\mathcal{F}}X \) is \( \hat{\uparrow}\{\hat{\downarrow}\{a\}\} \).

We claim that the algebras for the composite monad \( \hat{\mathcal{F}}\hat{\mathcal{F}} \) are *distributive lattices* (with units). If \( \alpha : \hat{\mathcal{F}}\hat{\mathcal{F}}X \to X \) is an algebra, the structure map can be decomposed into \( \alpha^\hat{\mathcal{F}} : \hat{\mathcal{F}}X \to X \) and \( \alpha^\hat{\mathcal{F}} : \hat{\mathcal{F}}X \to X \), which correspond to finite joins and finite meets respectively (Examples 7 and 13). We can then recover \( \alpha \) as \( \alpha^\hat{\mathcal{F}} \circ \hat{\mathcal{F}} \alpha \) whose action may be written as

\[
\hat{\uparrow}\{\hat{\uparrow} a_1, \ldots, \hat{\uparrow} a_n\} \mapsto \hat{\downarrow}\{\hat{\downarrow} a_1, \ldots, \hat{\downarrow} a_n\} \mapsto \bigvee\{\bigwedge a_1, \ldots, \bigwedge a_n\}
\]

The distributive law gives the identity:

\[
(\bigvee a_1) \land \cdots \land (\bigvee a_n) = \bigvee\{x_1 \land \cdots \land x_n : x_1 \in a_1, \ldots, x_n \in a_n\}
\]

Since the operations are finitary, we can reduce it to each of the following equivlanent forms:

\[
\begin{align*}
z \land (\bigvee a) &= \bigvee\{z \land x : x \in a\} \\
z \land (x \lor y) &= (z \land x) \lor (z \land y)
\end{align*}
\]

To see that \( \alpha \) is a lattice, we must also check the absorption laws: \( x \land (x \lor y) = x \) and its dual. Note that \( x \land (x \lor y) \) is the valuation of the “term” \( \hat{\uparrow}\{\hat{\downarrow}\{x\}, \hat{\downarrow}\{x, y\}\} \). Since \( \hat{\downarrow}\{x\} \subseteq \hat{\downarrow}\{x, y\} \), this term is equal to \( \hat{\uparrow}\{\hat{\downarrow}\{x\}\} \), whose valuation is just \( x \). The other absorption law is similar.

Other similar examples are the following:

- By symmetry, there is a distributive law \( \lambda'_X : \hat{\mathcal{F}}\hat{\mathcal{F}}X \to \hat{\mathcal{F}}\hat{\mathcal{F}}X \) which gives a composite monad \( \hat{\mathcal{F}}\hat{\mathcal{F}} \). This monad is isomorphic to the above composite monad. The distributive laws \( \lambda \) and \( \lambda' \) are inverses.

- The distributive law \( \lambda_X : \hat{\mathcal{F}}+\hat{\mathcal{F}}+X \to \hat{\mathcal{F}}+\hat{\mathcal{F}}+X \) gives a composite monad \( \hat{\mathcal{F}}+\hat{\mathcal{F}}+ \). Its algebras are distributive lattices (with possibly no units).

- The distributive law \( \lambda_X : \hat{\mathcal{F}}+\hat{\mathcal{F}}X \to \hat{\mathcal{F}}\hat{\mathcal{F}}+ \) gives a composite monad \( \hat{\mathcal{F}}\hat{\mathcal{F}}+ \). Its algebras are complete lattices with an infinite distributive law:

\[
z \land (\bigvee a) = \bigvee\{z \land x : x \in a\}
\]

Such structures are referred to as *frames* or *complete Heyting algebras*. The frame morphisms preserve sups and binary meets. As we have already noted the monad isomorphism \( \hat{\mathcal{F}} \cong \hat{\mathcal{D}}\hat{\mathcal{F}} \), the free frame monad is a composite of three monads \( \hat{\mathcal{D}}\hat{\mathcal{F}}\hat{\mathcal{F}}+ \).
• The monads $\mathcal{P}^\con_-$ and $\mathcal{F}^\con_+$ on Poset are defined as above but restrict to consistent sets or finite sets. The distributive law $\lambda_X : \mathcal{F}^\con_+ \mathcal{P}^\con X \to \mathcal{F}^\con_+ \mathcal{P}^\con X$ gives a composite monad $\mathcal{P}^\con_+ \mathcal{F}^\con_+$. Its algebras are distributive consistently-complete cpo’s. They satisfy the infinite distributive law:

$$z \land (\bigvee^\con a) = \bigvee^\con \{ z \land x : x \in a \} \quad \text{if } \{ z \} \cup a \text{ is consistent}$$

This is again a composite of three monads $\mathcal{DP}^\con_+ \mathcal{F}^\con_+$. One can also use pairwise compatible sets in place of consistent sets above. Bounded sets can also be used, except that

**Example 25** If $X$ is a poset, let $\mathcal{P}_{pc}X$ denote the poset of pairwise compatible subsets (finite subsets) of $X$, ordered by inclusion. We have the usual adaptions. $\downarrow \mathcal{P}_{pc}X$ is the poset of lower sets of elements of $\mathcal{P}_{pc}X$. $\uparrow \mathcal{P}_{pc}X$ is the poset of upper sets of nonempty finite elements of $\mathcal{P}_{pc}X$, ordered by reverse inclusion. We have monads $\downarrow \mathcal{P}_{pc}$ and $\uparrow \mathcal{P}_{pc}$ on Poset.

Using the distributive law $\lambda_X : \uparrow \mathcal{P}_{pc} \downarrow \mathcal{P}_{pc}X \to \downarrow \mathcal{P}_{pc} \uparrow \mathcal{P}_{pc}X$, we obtain a composite monad $\downarrow \mathcal{P}_{pc} \uparrow \mathcal{P}_{pc}$ on Poset.

The functor $\downarrow \mathcal{P}_{pc} : \text{Poset} \to \text{Poset}$ assigns to a poset $X$, the poset of lower sets of $\mathcal{P}_{pc}X$, ordered by inclusion. The functor $\uparrow \mathcal{P}_{pc} : \text{Poset} \to \text{Poset}$ assigns the poset of upper sets of finite sets in $\mathcal{P}_{pc}X$, ordered by reverse...
TS-algebras and morphisms form a category $C^{TS}$. There are evident forgetful functors from $C^{TS}$ to $C^T$ and $C^S$. Thus, we have the following commutative square of forgetful functors:

\[
\begin{array}{ccc}
C^{TS} & \longrightarrow & C^T \\
\downarrow & & \downarrow \\
C^S & \longrightarrow & C^G \\
\end{array}
\]

The functors leading to $C$ have left adjoints $F_S$ and $F_T$. We may ask if the functors going out of $C^{TS}$ have left adjoints too. The results of Beck [Bec69] show that the functor $C^{TS} \rightarrow C^S$ always has a left adjoint while the functor $C^{TS} \rightarrow C^T$ may not always have one, but if it does it will be a certain coequalizer. We examine these functors below:

1. Note that an $S$-algebra is a map $\sigma : SX \rightarrow X$ and such algebras form a category $C^S$. We obtain a monad $\tilde{T}$ on $C^S$ by first defining a functor $\tilde{T} : C^S \rightarrow C^S$:

\[
\langle X, \sigma : SX \rightarrow X \rangle \mapsto \langle TX, \lambda_X ; T\sigma : STX \rightarrow TSX \rightarrow TX \rangle
\]

\[
f : \langle X, \sigma \rangle \rightarrow \langle Y, \psi \rangle \mapsto Tf : \langle TX, \lambda_X ; T\sigma \rangle \rightarrow \langle TY, \lambda_Y ; T\psi \rangle
\]

One verifies that $Tf$ is a morphism of $S$-algebras. Likewise, the natural transformations $\eta^T_X : X \rightarrow TX$ and $\mu^T_X : TTX \rightarrow TX$ are also morphisms of $S$-algebras. Thus we obtain monad $\tilde{T}$, which is called the *lifted monad* of $T$ on $C^S$.

Now, a $\tilde{T}$-algebra in $C^S$ is a map $\tau : \tilde{T}\langle X, \sigma \rangle \rightarrow \langle X, \sigma \rangle$ which must be a morphism of $S$-algebras, i.e.,

\[
\begin{array}{ccc}
STX & \xrightarrow{S\tau} & SX \\
\downarrow \lambda_X ; T\sigma & & \downarrow \sigma \\
TX & \xrightarrow{\tau} & X
\end{array}
\]

If $\alpha : TSX \rightarrow X$ is a $TS$-algebra, then by taking $\sigma = \alpha^S$ and $\tau = \alpha^T$, the above square is nothing but (7). Thus, every $TS$-algebra on $C$ can be viewed as a $\tilde{T}$-algebra on $C^S$ and we have an embedding

\[
C^{TS} \xrightarrow{\Phi} (C^S)^\tilde{T}
\]

It turns out that this is in fact an isomorphism. Thus, the adjunction $\tilde{F}^T \dashv \tilde{G}^T : (C^S)^\tilde{T} \rightarrow C^S$ applies to $C^{TS} \rightarrow C^S$ (upto isomorphism).

2. Coequalizer . . .

**Example 26** Consider the monads $\text{list, Z}(-) : \text{Set} \rightarrow \text{Set}$ whose algebras are monoids and abelian groups respectively. With the distributive law $\lambda_X : \text{listZ}(X) \rightarrow \text{Z(listX)}$

\[
\lambda_X : \Pi_i \Sigma_x (\Pi_j k_{ij} x_j) \mapsto \Sigma_x (\Pi_i (k_{ij}) x_j)
\]

we obtain the composite monad $\text{Z(list-)}$. The algebras of this monad are rings. Evidently, rings are abelian groups in the category of monoids.

**Example 27** Consider the monads $\mathbb{P}^1, \mathbb{P}^+ : \text{Set} \rightarrow \text{Set}$ whose algebras are pointed sets and affine complete semilattices (structures with nonempty sups) respectively. We can define a distributive law $\lambda_X : \mathbb{P}^+ \mathbb{P}^1 X \rightarrow \mathbb{P}^1 \mathbb{P}^+ X$ by:

\[
\begin{array}{ccc}
\emptyset & \mapsto & \emptyset \\
a \neq \emptyset & \mapsto & \{ x : \{ x \} \in a \}
\end{array}
\]

24
Thus we obtain a composite monad $\mathbb{P}^1\mathbb{P}^+$ which is isomorphic to the powerset monad. The isomorphism $\sigma_X : \mathbb{P}^1\mathbb{P}^+X \to \mathbb{P}X$ is given by

\[
\begin{align*}
\emptyset & \mapsto \emptyset \\
\{a\} & \mapsto a
\end{align*}
\]

Thus, one can view complete semilattices as pointed affine complete semilattices.

There is also a distributive law in the reverse direction $\kappa_X : \mathbb{P}^1\mathbb{P}^+X \to \mathbb{P}^+\mathbb{P}^1X$ given by

\[
\begin{align*}
\emptyset & \mapsto \{\emptyset\} \\
\{a\} & \mapsto \{\{x\} : x \in a\}
\end{align*}
\]
References


