Math. Struct. in Comp. Science (1998), *vol.* 8, *pp.* 231–276. Printed in the United Kingdom
© 1998 Cambridge University Press

Implementing collection classes with monads

ERNIE G. MANES[†]

Department of Mathematics and Statistics, University of Massachussetts, Research Center Tower, Amherst, Massachussetts, U.S.A.

Received 15 October 1995; revised 15 November 1997

In object-oriented programming, there are many notions of 'collection with members in X'. This paper offers an axiomatic theory of collections based on monads in the category of sets and total functions. Heuristically, the axioms defining a *collection monad* state that each collection has a finite set of members of X, that pure 1-element collections exist and that a collection of collections flattens to a single collection whose members are the union of the members of the constituent collections. The relationship between monads and universal algebra leads to a formal definition of collection implementation in terms of tree-processing. Ideas from elementary category theory underly the classification of collections. For example, collections can be zipped if and only if the monad's endofunctor preserves pullbacks. Or, a collection can be uniquely specified by its shape and list of data if the morphisms of the Kleisli category of the monad are all deterministic, and the converse holds for commutative monads. Again, a collection monad is *ordered* if the monad's functor preserves equalizers of monomorphisms (so, in particular, if collections can be zipped the monad is ordered). Every implementable monad is ordered. It is shown using the well-ordering principle that a collection monad is ordered if and only if its functor admits an appropriated list-valued natural transformation that lists the members of each collection.

1. Introduction

Filinski (1994, page 446) states

> Over the past few years, monads have gained considerable acceptance in the lazy functional programming world.... It is somewhat remarkable that monads have had no comparable input on impure functional programming....

This paper uses monads to implement collection classes (including sets, bags, lists and trees) in languages that are not functional. We particularly target mainstream object-oriented languages such as C++ and Delphi, although specific language features play no role below. We do tangentially note, however, that imperative object languages have the potential to support a functional programming style. The hallmark operators of functional programming can be implemented as instance methods (as is in fact done in Smalltalk). Functions can be passed as arguments because objects can be, and objects include single-function-method classes whose instances 'are' functions. While these mainstream languages

[†] This work was partially supported by the National Science Foundation under grant CCR-9406793.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

will never be truly functional like ML and Haskell, so much programming is done with them that it is worthwhile developing mathematical principles that govern the design of the collection class hierarchy and its internal implementation in these settings.

What is a collection class? Intuitively, they should satisfy the following axioms with respect to a universal set X:

Each collection has a finite set of members of X.	(1.1)
Each element of X induces a 'pure' one-element collection (for example, singleton set, one-element multiplicity-one bag, one-element list, a single leaf tree).	(1.2)
A collection of collections should 'flatten' to a single collection whose members are the union of the individual sets of members.	(1.3)

These axioms are formalized in Section 4 with the concept of a *collection monad* in the category of sets and total functions; this category will henceforth be denoted as $\mathbb{S}$. Thus we fully adopt the idea in Moggi (1991) of using monads to implement functional operations. What is new is the axiomatization of the root class appropriate for collections. We develop three primary subclasses; the 'ordered' collection monads (whose members form a list, not just a set), the 'zipped' collection monads (for which it is possible to 'zip together' two collections with the same 'shape', and the 'shapely' collection monads, which admit a list-valued monad map. We formalize how collection classes are implemented with tree-processing and show that for shapely monads there is a plenitude of canonical implementations. Shapely monads are characterized by the ability to specify a collection with two independent elements of structure: a shape and a list of data, an idea emphasized by Jay (1995).

Trinder (1991) introduced a class of monads called 'ringads' to model 'bulk types' in database programming with specific emphasis on the 'comprehensions' of Section 3 below. His motivation is very similar to ours and he speaks explicitly about collections. We discuss the ringad axioms after Lemma 2.6. Ringads are not developed beyond the definition in that paper.

Another database paper emphasizing the monad structure of collections is Buneman *et al.* (1995). The examples in that paper are restricted primarily to three types of collection (sets, bags and lists), but it is noted in an appendix that monads are a generalization. Fundamental in that paper is the principle that finite collections are built up by finitely many unions given singletons where 'union' and 'singleton' change with the notion of collection. For general collection monads it is more complicated. For example, probability distributions are combined by arbitrary probability weights so there is a whole class of 'union' operations. As another example, repetition-free lists are constructed by adding a new element only if it does not already occur, so that unions may not get larger. Such structural recursions are worth investigating, but that is the subject for another paper.

Overall, the collection-monad point of view is not compatible with the type-theoretic aspect prevalent in papers on functional programming and database programming. For example, the fundamental cartesian-product type used to form records of collections is *never* of collection type (Proposition 5.4 below).

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Monads in categories have a long history beginning with Godement (1958, Appendix) and Eilenberg and Moore (1965). See Barr and Wells (1985) for a textbook treatment. We will work exclusively in the category $\mathbb{S}$ because our collections are ordinary finite collections. The theoretical advantages of this specialization have been previously noted by the author (Manes (1982) – see Goodman and Nguyen (1985) for a textbook treatment) and are further pursued below. Other papers developing monads in functional programming include Moggi (1995), Steele (1994) and Wadler (1992).

Some would attempt to present the theory of this paper for more general categories. At this point we see little motivation to do so, particularly since it would make the paper inaccessible to many mainstream programmers. We do feel that the limited amount of elementary category theory used below is a very good advertisement for its power – it has allowed us an elegant language to express fundamental design issues in an implementation-independent form.

Heuristics for the definition of 'monad' follow by directly attempting to formalize Axioms (1,2,3) above. A specific monad $\mathbf{T}$ should consist of T, $\eta, (-)^{\#}$ as follows. TX is the set of all '$\mathbf{T}$-collections with members in X'. The second axiom provides functions $\eta_X : X \longrightarrow TX$. Let us interpret 'collections of collections' as in Axiom (3) as follows. Start with a collection $\tau \in TX$. Then, for some set Y, 'replace' each member x of τ with a collection $\alpha(x) \in TY$. The net result is a new collection $\nu \in TY$ formed from the initial collection of collections in TTY by 'flattening'. Since it is not clear at this stage how to isolate the subset of X that will comprise the members of τ, we will play it safe and define $\alpha(x)$ for every $x \in X$. This amounts to a construction $(-)^{\#}$ assigning to each $\alpha : X \longrightarrow TY$ a function $\alpha^{\#} : TX \longrightarrow TY$. Thus ν above is $\alpha^{\#}(\tau)$. Axiom (3) still cannot be stated in full until the set $mem\,\tau$ of members of τ is defined. To this end, notice that for a function $f : X \longrightarrow Y$ there is an induced function $Tf : TX \longrightarrow TY$ defined by $Tf = (\eta_Y f)^{\#}$. It will turn out that for $\tau \in TX, mem\,\tau$ will be the unique smallest subset $P \subset X$ such that τ is in the image of $T\,inc_P$ with $inc_P : P \longrightarrow X$ the inclusion function.

The quintessential implementable collection class is Σ-trees. Here, Σ is a (single-sorted) *ranked set*, that is, a (not necessarily disjoint) sequence $(\Sigma_n : n = 0, 1, 2, \ldots)$ of sets, any of which may be empty. The set $T_\Sigma X$ of Σ-*trees with leaves in* X is defined in the usual way by

$$\begin{aligned} &X \subset T_\Sigma X \\ &\text{Given } \tau \in \Sigma_n, \tau_1, \ldots \tau_n \in T_\Sigma X, \tau(\tau_1, \ldots, \tau_n) \in T_\Sigma X. \end{aligned} \tag{1.4}$$

If $\tau \in T_\Sigma X$, its members are its leaves, *i.e.*

$$\begin{aligned} &mem\, x = \{x\} \\ &mem(\tau(\tau_1, \ldots, \tau_n)) = \bigcup (mem\, \tau_i : i = 1, \ldots, n). \end{aligned} \tag{1.5}$$

For trees, members form a list, not just a set. For example, if $\Sigma_2 = \{\omega\}$ (we adopt henceforth the convention that unmentioned Σ_n are empty), then the set of members of the tree $\tau = \omega(\omega(x, y), \omega(z, \omega(y, x)))$ is $\{x, y, z\}$ but the list of members, read in left-to-right order, is $xyzyx$. The *shape* of τ is $\omega(\omega(\bullet, \bullet), \omega(\bullet, \omega(\bullet, \bullet)))$. The original tree τ is uniquely specified by its shape and its member list (of the correct length). This exemplifies Jay's concept

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

of separating shape and data (data meaning the member list). The formalization of these ideas for more arbitrary collection classes will be given in the later sections of this paper.

It is an empirical fact that standard collection classes arise by imposing equations on trees. This amounts to equational presentations as arise in the study of universal algebra. We pause to review some of the definitions.

If $V = a, b, c, \ldots$ is a countably-infinite set of standard variables, a Σ-*equation* is a pair (τ, v), written $\tau = v$, with τ, $v \in T_\Sigma X$. As we have motivated already, for any function $\alpha : V \longrightarrow T_\Sigma X$, there is an induced function $\alpha^\# : T_\Sigma V \longrightarrow T_\Sigma X$ defined by

$$\begin{aligned} \alpha^\#(v) &= \alpha(v) \text{ (for } v \in V) \\ \alpha^\#(\tau(\tau_1, \ldots, \tau_n)) &= \tau(\alpha^\#(\tau_1), \ldots, \alpha^\#(\tau_n)). \end{aligned} \tag{1.6}$$

A *congruence* on $T_\Sigma X$ is an equivalence relation $\simeq$ on $T_\Sigma X$ with the additional property that if $\tau_1 \simeq v_1, \ldots, \tau_n \simeq v_n$, then for all $\omega \in \Sigma_n, \omega(\tau_1, \ldots, \tau_n) \simeq \omega(v_1, \ldots, v_n)$. For E a set of equations, the set $T_{\Sigma,E}X$ of all (Σ, E)-collections on X is defined as the set of all E-equivalence classes on $T_\Sigma X$; more precisely,

$$\begin{aligned} &T_{\Sigma,E}X = T_\Sigma X / E_X \text{ for } E_X \text{ the intersection of all congruences} \\ &\text{containing all pairs } (\alpha^\#(\tau), \alpha^\#(v)) \text{ for } \tau = v \in E,\ \alpha : V \longrightarrow T_\Sigma X. \end{aligned} \tag{1.7}$$

This is a usual construction for the free algebra generated by X for the equational presentation (Σ, E).

Let us look at some examples from the point of view of collection classes. The set of trees with members in X is just $T_\Sigma X$, so that here E is empty. Lists are well-known to be elements of a free monoid so we use an equational presentation for monoids: $\Sigma_0 = \{\Lambda\}$, $\Sigma_2 = \{\omega\}$ with equations

$$\begin{aligned} \omega(a, \omega(b, c)) &= \omega(\omega(a, b), c) \\ \omega(a, \Lambda) &= a \\ \omega(\Lambda, a) &= a. \end{aligned}$$

Each equivalence class in $T_{\Sigma,E}X$ has a canonical representative in $T_\Sigma X$, for example $\omega(x_1, \omega(x_2, \ldots, \omega(x_{n-1}, x_n)\ldots)$, which we abbreviate using the usual list notation as $x_1 \ldots x_n$. The collection class of repetition-free lists can be constructed from the above presentation for lists by adding the equations $ab_1 \ldots b_n a = ab_1 \ldots b_n (n \geqslant 0)$. Applying this equation reduces a list by keeping leftmost occurrences. Other strategies are possible, so, ultimately, the choice of equational presentation may have some effect on algorithms used for internal implementation. Continuing, if we add the commutative law $\omega(a, b) = \omega(b, a)$ to repetition-free lists, we get finite subsets. Unlike the previous three examples, the internal implementation will have to make choices, since any expression must list its n leaves in a particular order with $n! - 1$ other choices representing the same set. Of course the solution to this problem of possible ambiguity will be to make the particular choice inaccessible to the user; because collections must be output, the choice cannot be made invisible, however. Our fifth and final example is bags. Just notice that a bag with members in X is an element of the free abelian monoid generated by X.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

It is a theorem (see Section 4) that every collection monad arises from an equational presentation. Our concept of implementation (in Section 6) is based on choosing a presentation (Σ, E) and representing a collection τ by a Σ-tree in the E-equivalence class of τ. However, we will see that some examples of collection monad do not have an obvious equational presentation. In a language such as C++ or Delphi we expect to code collection classes from scratch. At the level of internal implementation, collections will be equivalence class of Σ-trees represented by a canonical form, but at the theoretical level all this information is condensed into a monad, an entirely different structure.

A rule-based language such as Mathematica is well suited for the direct implementation of examples such as those above. It is interesting to look briefly at how this is done, even though such code is not available in other mainstream languages. In Mathematica, 'everything is an expression'. In particular, any Σ-tree in X can be written just by replacing parentheses by square brackets. Lists are built in with canonical form *List* $[x_1 \dots x_n]$, synonymously written $\{x_1 \dots x_n\}$. Mathematica uses pattern-matching reminiscent of ML. The notation $p_$ means any expression, named p within the pattern definition. Two underscores specifies one or more expressions and three underscores specifies zero or more expressions. Thus sets may be defined by *set* $[x___] := \mathit{Union}\ [\{x\}]$. (Example: *set* $[a, c, a, b, a, b]$ is simplified to *set* $[a, b, c]$ – Mathematica chooses alphabetical ordering for internal implementation). The equation for repetition-free lists is implemented by the single rule

$$rfl[x___, a_, y___, a_, z___] := rfl[x, a, y, z].$$

Hence, $rfl[a, c, a, b, a, b]$ simplifies to $rfl[a, c, b]$. Finally, we use the notation $\{x, i\}$ to denote x with multiplicity i. Then bags may be implemented by

$$\begin{aligned} &\mathit{SetAttributes}[\mathit{bag}, \mathit{Orderless}] \\ &\mathit{bag}\ [x___, \{a_, i_\}, \{a_, j_\}] := \mathit{bag}[x, \{a, i + j\}]. \end{aligned}$$

The first line makes the $bag[\{x_1, i_1\} \dots \{x_n, i_n\}]$ independent of the order of the $\{x_j, i_j\}$. The rule in the second line tries to match any of the $(k + 2)!$ possible orderings (where k is the length of the sequence x), a very inefficient algorithm despite the concise notation.

Morphisms between monads and other less structured natural transformations will play an important role in our theory. Assigning the set of members of a collection is a monad map (called *asSet* in Smalltalk). By the well-known variety theorem of Birkhoff (1935) quotient monads are tantamount to 'adding additional equations'. Any shapely monad admits a monad map to lists that assigns its list of members. There are many submonads of interest; for example, the 'deterministic collections' in a collection monad form a shapely submonad.

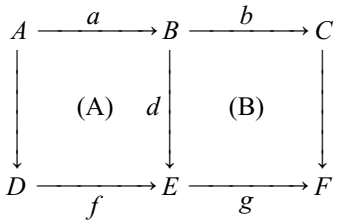
Now let us begin!

We assume a modest background in elementary category theory. A few salient definitions and facts for general categories will be provided now. Many texts such as Mac Lane (1971) exist to help the reader further. Given $B \xleftarrow{f} A \xrightarrow{g} C$, say that f, g are *jointly monic* if for all $t, u : D \longrightarrow A$, if $ft = fu$ and $gt = tu$, then $t = u$. For example, the two morphisms with common domain in a pullback square are jointly monic. A morphism f is $\{\mathit{split\ monic}\}, \{\mathit{split\ epic}\}$ if there exists a morphism g with $\{gf = id\}$, $\{fg = id\}$. In

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

the category $\mathbb{S}$ the split monics are the injective functions (except for the inclusion of the empty set in a non-empty set) and the split epics are the surjective functions. Of course we assume the axioms of choice here, as we shall at other key points below.

Lemma 1.8. (Pullback Lemma) Consider the following commutative diagram in an arbitrary category.

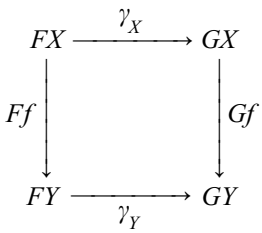


Then the following statements hold.

(1) If (A) and (B) are pullbacks, so is the perimeter (A, B).
(2) If (A, B) is a pullback and b, d are jointly monic, then (A) is a pullback.
(3) If b, d are jointly monic, f is split epic and the perimeter rectangle is a pullback, then (B) is a pullback.
(4) If $C = A$, $F = D$, $ba = id_A$, $gf = id_D$ and d is split monic, then (A) is a pullback.

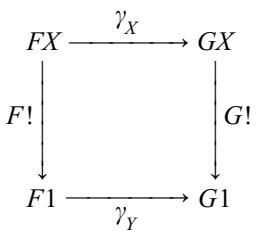
Proof. The proof is routine. □

A natural transformation $\gamma : F \longrightarrow G$ is *cartesian* if all naturality squares



are pullbacks.

Lemma 1.9. Let F, G be functors $C \longrightarrow D$ and let 1 be a terminal object of C. Then a natural transformation $\gamma : F \longrightarrow G$ is cartesian if and only if the squares



(where $! : X \longrightarrow 1$ is the unique morphism) are pullbacks.

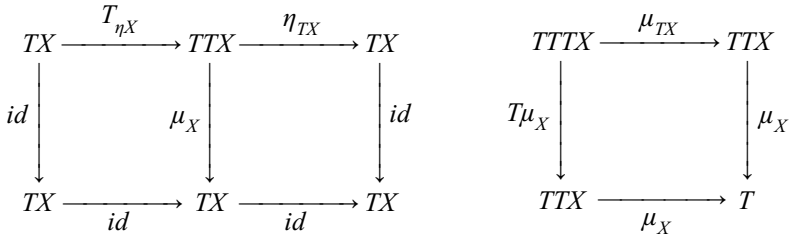
Proof. For $f : X \longrightarrow Y$, $!f = ! : X \longrightarrow 1$. Now use Part (2) of the Pullback Lemma. □

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

2. Finitary monads

The collection class of lists admits at least the following mathematical constructions. For each set X there is a set $TX = X^*$ of lists with members in X (including the empty list Λ). There is a 'class method' that creates, for each set X and element $x \in X$, the one-member list $\eta_X(x) = x \in TX$. There is a function of form $\mu_X : TTX \longrightarrow TX$ that 'flattens' a list of lists to a single list, for example $\mu_X((xyz)(\Lambda)(ab)) = xyzab$. It is a recent discovery that list-processing operations one expects to find in a programming language can, for the most part, be expressed in terms of T, η, μ and a few auxiliary parameters, as appropriate. The axiomatized theory is precisely the theory of monads. Definition 2.1 gives the formal definition.

Definition 2.1. A *monad* (in the category $\mathbb{S}$ of sets and functions) is a triple $\mathbf{T} = (T, \eta, \mu)$ with $T : \mathbb{S} \longrightarrow \mathbb{S}$ an endofunctor and with $\eta : id \longrightarrow T$, $\mu : TT \longrightarrow T$ natural transformations subject to the following additional three commutative diagrams that must hold for all sets X:



To verify that a candidate (T, η, μ) is a monad requires that seven separate squares commute and one of these iterates T twice. The following version requires only three axioms and none of these iterates T.

Proposition 2.2. A monad may equivalently be viewed as $(T, \eta, (-)^\#)$ where T is an object-construction $X \mapsto TX$, η assigns a function $\eta_X : X \longrightarrow TX$ to each set X and $(-)^\#$ is the extension operation assigning $\alpha^\# : TX \longrightarrow TY$ to $\alpha : X \longrightarrow TY$ subject to the three axioms:

$$\alpha^\# \eta_X = \alpha$$
$$\eta_X^\# = id_{TX}$$
$$\text{For } \alpha : X \longrightarrow TY, \quad \beta : Y \longrightarrow TZ, \quad (\beta^\# \alpha)^\# = \beta^\# \alpha^\#$$

Proof. Given Definition 2.1, define $\alpha^\# = \mu_Y T\alpha$. Conversely, for $f : X \longrightarrow Y$ define $Tf : TX \longrightarrow TY$ to be $(\eta_Y f)^\#$ and define $\mu_X = (id_{TX})^\#$. Though there are many things to check, the details are routine. □

In group theory it is usual to use the same symbol for multiplication, inverse and unit in all groups, but different symbols for the carrier sets. We adopt a similar convention for monads, usually using η, μ, and $(-)^\#$ for all monads, but different symbols for the functors. Monads will be in boldface non-italic such as **T** and **Bag**. Often we use the same symbol as the monad but non-bold italic for the functor, as in T, *Bag*.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Proposition 2.3. Given a monad $\mathbf{T} = (T, \eta, (-)^{\#})$ define

$$f^{\star} = X \xrightarrow{f} Y \xrightarrow{\eta_Y} TY$$

for $f : X \longrightarrow Y$ and define the *Kleisli composition* $\beta \circ \alpha : X \longrightarrow TZ$ given $\alpha : X \longrightarrow TY$ and $\beta : Y \longrightarrow TZ$ by

$$\beta \circ \alpha = X \xrightarrow{\alpha} TX \xrightarrow{\beta^{\#}} TY.$$

Then the following describes a category $\mathbb{S}_{\mathrm{T}}$: the objects are all sets; the morphisms are given by $\mathbb{S}_{\mathrm{T}}(X, Y) = \mathbb{S}(X, TY)$; the identity morphism of X is η_X; the composition is $\beta \circ \alpha$. Moreover, given $f : X \longrightarrow Y$, $\alpha : Y \longrightarrow TZ$, $\alpha \circ f^{\star} = \alpha f : X \longrightarrow TZ$.

Proof. The proof is easy. □

$\mathbb{S}_{\mathrm{T}}$ is called the *Kleisli category* of $\mathbf{T}$ since the construction originates with Kleisli (1965). (In more general discussions, if $\mathbf{T}$ is a monad defined over a category $\mathbb{K}$, the Kleisli category would be written $\mathbb{K}_{\mathrm{T}}$).

Definition 2.4. A *morphism of monads*, synonymously a *monad map* $\gamma : \mathbf{S} \longrightarrow \mathbf{T}$, is a natural transformation $\gamma : S \longrightarrow T$ such that (A) and (B) commute:

$$\begin{array}{ccc} id & \xrightarrow{\eta} & S \\ {\scriptstyle id}\downarrow & (\mathrm{A}) & \downarrow{\scriptstyle \gamma} \\ id & \xrightarrow[\eta]{} & T \end{array} \qquad \begin{array}{ccc} SS & \xrightarrow{\gamma\gamma} & TT \\ {\scriptstyle \mu}\downarrow & (\mathrm{B}) & \downarrow{\scriptstyle \mu} \\ S & \xrightarrow[\gamma]{} & T \end{array}$$

where $\gamma\gamma$ is the vertical composition $(\gamma\gamma)_X = SSX \xrightarrow{S_{\gamma X}} STX \xrightarrow{\gamma_{TX}} TTX = SSX \xrightarrow{\gamma_{SX}} TSX \xrightarrow{T_{\gamma X}} TTX$. This can be equivalently expressed using extension instead of μ by requiring (A) and (C) (or, alternatively, (A) and (D)) for all $\alpha : X \longrightarrow SY$:

$$\begin{array}{ccc} SX & \xrightarrow{\gamma_X} & TX \\ {\scriptstyle \alpha^{\#}}\downarrow & (\mathrm{C}) & \downarrow{\scriptstyle (\gamma_Y \alpha)^{\#}} \\ SY & \xrightarrow[\gamma_Y]{} & TY \end{array} \qquad \begin{array}{ccc} SSX & \xrightarrow{\gamma_{SX}} & TSX \\ {\scriptstyle \mu_X}\downarrow & (\mathrm{D}) & \downarrow{\scriptstyle \gamma_X^{\#}} \\ SX & \xrightarrow[\gamma_X]{} & TX \end{array}$$

Here, it is unnecessary to assume that γ is natural, since that follows from (A) and (C/D). The proof is a routine diagram paste based on the definitions. Notice that a monad map $\gamma : \mathbf{S} \longrightarrow \mathbf{T}$ induces a functor on the Kleisli categories $\bar{\gamma} : \mathbb{S}_{\mathrm{S}} \longrightarrow \mathbb{S}_{\mathrm{T}}$ by $\bar{\gamma}(X) = X$, $\bar{\gamma}(X \xrightarrow{\alpha} SY) = X \xrightarrow{\alpha} SY \xrightarrow{\gamma_Y} TY$.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Table 2.1. *Some list-processing operations for general monads.* $X + X$ *means disjoint union, the coproduct in* $\mathbb{S}$.

- *atomic* (x) is $\eta_X(x)$.
- $\lambda x \cdot$ *map* (f, x) is Tf.
- *flatten* (τ) is $\mu_X(\tau)$.
- *outer* (f, τ, v) for $f : X \times Y \longrightarrow Z$, $\tau \in TX$, $v \in TY$ is the element of TTZ given by $T(\lambda x.T(\lambda y.f(x, y))v)(\tau)$.
- *select* $(\tau, test, reduce) \in TX$ for $\tau \in TX$, $test : X \longrightarrow X + X$, $reduce : X + X \longrightarrow TX$ is $(reduce\ test)^{\#}\tau$.
- *join* $(\omega, \tau, v) \in TX$ for $\omega \in T\bar{2}(\bar{2} = \{1, 2\}), \tau, v \in TX$ is $(\bar{2} \xrightarrow{< \tau, v >} TX)^{\#}\omega$. The order $1 < 2$ in $\bar{2}$ is important.
- *append* $(\omega, \tau, x) \in TX$ for $\omega \in T2$, $\tau \in TX$, $x \in X$ is *join* $(\omega, \tau, atomic\ (x))$.

Definition 2.5. A monad map $\mathbf{S} \longrightarrow \mathbf{T}$ is a *submonad* if it is pointwise injective and is a *quotient monad* if it is pointwise surjective. We also say '**S** is a submonad of **T**' and '**T** is a quotient of **S**', respectively.

Monads and their morphisms constitute a category whose isomorphisms are the pointwise bijective maps. We will see many examples of submonads and quotient monads later, so let us pause to note that there is a simpler way to think of them. Consider a monad **T** and monomorphisms of form $\gamma_X : SX \longrightarrow TX$ (in practice, SX is defined as a subset of TX and then γ is just the inclusion). At this stage S is not even a functor but, even so, S either is or is not a submonad, that is, S can be the functor part of a submonad in at most one way. This will happen when $\eta_X : X \longrightarrow TX$ factors through γ_X (to define η for **S**) and in 2.4(C), when $(\gamma_Y \alpha)^{\#}\gamma_X$ factors through γ_Y (to define $\alpha^{\#}$ for **S**). Similarly, given a monad **S** and pointwise surjective $\gamma_X : SX \longrightarrow TX$ there is at most one monad structure on **T** rendering γ a quotient monad. For η is forced by 2.4(A) and there is at most one way to define $(\gamma_X \alpha)^{\#}$ in 2.4(C) since γ_X is onto and every $X \longrightarrow TY$ has form $\gamma_Y \alpha$ because γ_Y is onto.

We note that the pointwise image factorization of a monad map provides the category of monads and monad maps with a strong epi - mono factorization system.

The following lemma will be needed later.

Lemma 2.6. Let $\theta : \mathbf{S} \longrightarrow \mathbf{T}$ be a monad quotient, let **U** be a monad and let $\gamma_X : TX \longrightarrow UX$ be functions. Then if $\gamma\theta$ is a monad map, so too is γ.

Proof. This is routine. To establish 2.4(C), show that the diagram commutes preceded by θ_X and note that every $\alpha : X \longrightarrow TY$ has form $\theta_Y \beta$. □

Table 2.1 gives some of the promised 'list-processing operations' that apply to any monad. Monad comprehensions also apply to arbitrary monads, and these will be the subject of the next section.

Trinder (1991) introduced 'ringads', which are monads subject to an additional *join* operation subject to the following additional axioms (translated to our notations) for $\alpha : X \longrightarrow TY$, $\tau, v \in TX$:

$$T\varnothing \text{ has one element, call it } z.$$
$$join_X(z, \tau) = \tau = join_X(\tau, z)$$
$$\alpha^{\#}(join\ (\tau, v) = join\ (\alpha^{\#}\tau, \beta^{\#}\tau).$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

(By Lemma 2.8, below, $T\varnothing \longrightarrow TX$ is a monomorphism, by virtue of which $z \in TX$). The first axiom provides the Kleisli category with zero maps. This is very restrictive and excludes most of our examples. Our *join* in Table 2.1 always satisfies Trinder's third axiom as follows:

$$\begin{aligned} \alpha^{\#}(join\ (\omega, \tau, \nu) &= \alpha^{\#} < \tau, \nu >^{\#} \omega = (\alpha^{\#} < \tau, \nu >)^{\#}\omega \\ &= < \alpha^{\#}\tau, \alpha^{\#}\nu >^{\#} \omega = join\ (\omega, \alpha^{\#}\tau, \alpha^{\#}\nu). \end{aligned}$$

For the monad of lists, *flatten, outer, select, join* and *append* behave exactly as the operations with those names in Mathematica, providing that the extra parameters *reduce* and ω are set to the appropriate values (usually these are fixed for a given monad). The examples below illustrate this.

Example 2.7. Σ-trees forms a monad $\mathbf{T}_\Sigma = (T_\Sigma, \eta, (-)^{\#})$ with $\eta_X(x) = x$ and $\alpha^{\#}$ as in Equation 1.5. Tf preserves the tree shape and replaces each leaf x with $f(x)$. The *flatten* operation regards a tree of trees as a single tree in the obvious way. The *select* operation is most natural when there is some sort of nil symbol $\perp \in \Sigma_0$ to indicate 'non-selection', but any fixed tree could be used instead. Define *reduce* : $X + X \longrightarrow T_\Sigma X$ by mapping $(x, 1)$ to x and $(x, 2)$ to $\perp$. For $A \subset X$ let *test* map x to $(x, 1)$ if $x \in A$ and map x to $(x, 2)$ if $x \notin A$. Then *select* $(\tau, test, reduce)$ is obtained from τ by replacing all leaves not in A by $\perp$. The *join* and *append* operations are most natural when there is a specified 'join symbol' $\omega \in \Sigma_2$ producing the 'join tree' $\omega(1, 2) \in \mathbf{T}_\Sigma 2$, although any fixed tree with leaves $1, 2$ (possibly occurring more than once) can be used. Then the join of τ, ν is the tree $\omega(\tau, \nu)$.

The following notational conventions will find use later. Let us choose as a standard terminal object of $\mathbb{S}$ the set $1 = \{\bullet\}$. Thus a tree shape is an element of $T_\Sigma 1$. A tree $\tau \in T_\Sigma X$ is entirely determined by its shape t and its list of leaves w (listed in left-to-right order). We use the notation $\tau = t(w)$ to indicate this. Notice, by the way, that $t = (T_\Sigma !)\tau$ where $! : X \longrightarrow 1$ is the unique map.

Lemma 2.8. For any monad $\mathbf{T}$, T preserves monomorphisms.

Proof. Any functor preserves split monomorphisms, so we have only to prove that T maps the inclusion $i : \varnothing \longrightarrow X$ of the empty set to a monic when $T\varnothing \neq \varnothing$. To that end, let $\alpha : X \longrightarrow T\varnothing$ be any function. Then $\alpha i = \eta_\varnothing$ are both the empty function $\varnothing \longrightarrow T\varnothing$. It suffices to show that $\alpha^{\#} Ti$ is monic. Indeed, $\alpha^{\#} Ti = \alpha^{\#}(\eta_X i)^{\#} = (\alpha^{\#}\eta_X i)^{\#} = (\alpha i)^{\#} = \eta_\varnothing^{\#} = id_{T\varnothing}$. □

Definition 2.9. Let $\mathbf{T}$ be a monad and let $\tau \in TX$. If τ collects a set of members we expect this set to be a subset of X. We think of any subset of X containing the members of τ as a 'support' of τ. Formally, say that $P \subset X$ is a *support of* τ if $\tau \in TP$. This slight abuse of language makes sense, since $TP \subset TX$ via T of the inclusion $P \longrightarrow X$. Say that $\mathbf{T}$ is *finitary* if every τ has a finite support.

The monad $\mathbf{T}_\Sigma$ of Example 2.7 is finitary, since any superset of the set of leaves of a Σ-tree is a support. The structure of the set of supports can be quite complicated for infinitary monads. For example there is a monad $\mathbf{T}$ with TX the set of all ultrafilters on X for which the set of supports of $\mathscr{U} \in TX$ is $\mathscr{U}$ itself – any ultrafilter can be a set of supports!

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Theorem 2.10. For any equational presentation (Σ, E), the canonical projections $T_\Sigma X \longrightarrow T_{\Sigma,E}X$ form a monad quotient and $\mathbf{T}_{\Sigma,E}$ is finitary. Conversely, every finitary monad has form $\mathbf{T}_{\Sigma,E}$.

Proof. See Manes (1976, Theorems 4.15, 4.25) noting that 'monads' are called '(algebraic) theories' there. In this paper, a proof that collection monads have a special equational presentation is given in Theorem 4.14 and a proof that shapely monads have a special equational presentation is given in Theorem 8.5. □

Example 2.11. Lists forms a monad **List** $= ((-)^*, \eta, \mu)$ with X^* the usual set of all lists on X with η, μ as in the motivating discussion prior to Definition 2.1. The extension $\alpha^\#$ preserves the empty list and maps $x_1 \ldots x_n$ to the (flattened) concatenation $\alpha(x_1)\ldots\alpha(x_n)$. Thus $(Tf)(x_1 \ldots x_n) = f(x_1)\ldots f(x_n)$ is length-preserving. Menus and submenus form a possible scenario for Kleisli composition. For this monad it is natural to choose *reduce* : $X + X \longrightarrow X^*$ to be η on the true summand and constantly Λ on the false summand so that *select* produces the sublist passing the test, for example, *select* (1 10 3 7, $>$ 4?) is 10 7. Setting $\omega = 0\ 1 \in 2^*$ gives *join* as concatenation. *outer* ($\times$, 2 3, 1 5 6) with $\times$ ordinary multiplication produces a rectangular multiplication table as a list of lists: (2 10 12) (3 15 18).

Reversing a list produces a monad map *rev* : **List** $\longrightarrow$ **List**.

For any Σ, let $memList_X : T_\Sigma X \longrightarrow X^*$ assign each Σ-tree its list of members. This defines a cartesian monad map *memList* : $\mathbf{T}_\Sigma \longrightarrow$ **List**.

Example 2.12. An *ordered bag* on X is a list of form $x_1^{m_1} \ldots x_k^{m_k}$ where $x_1, \ldots, x_k$ are distinct elements of X and the m_i are integers > 0. A standard example is the prime factorization of an integer with the primes x_i arranged, as they often are, in ascending order. This notion of collection class gives rise to a monad **oBag** as follows. Of course, *oBag* X is the set of all ordered bags on X. The singleton collections are $\eta_X(x) = x^1$. It will be clearest to describe $(-)^\#$ by an example. Suppose that $\alpha(x) = a^2b^3$ and $\alpha(y) = d^2b^1a^4$. Then $\alpha^\#(x^m y^n)$ is obtained by the following process. First substitute the values of α:

$$(a^2b^3)^m(d^2b^1a^4)^n.$$

Then apply a law of exponents to distribute:

$$a^{2m}b^{3m}d^{2n}b^n a^{4n}.$$

Finally, apply leftmost first occurrence collection:

$$a^{2m+4n}b^{3m+n}d^{2n}.$$

The map operation is as follows: $(Tf)(x_1^{m_1} \ldots x_k^{m_k}) = f(x_1)^{m_1} \ldots f(x_k)^{m_k}$ followed by leftmost first occurrence collection. *outer* $(f, x^2y^3, a^1b^6c^2)$ will be an ordered bag of two ordered bags, namely

$$(f(x,a)^1 f(x,b)^6 f(x,c)^2)^2 (f(y,a)^1 f(y,b)^6 f(y,c)^2)^3.$$

For this monad, choose *reduce* : $X + X \longrightarrow oBags X$ to be η on the true summand and the empty bag on the false summand. Then if *test* is true for a, c and false for b, *select* $(2^3 3^5 7^2, \leqslant 3?)$ is $2^3 3^5$. The natural choice for the join parameter is $\omega = 0^1 1^1 \in T2$ since this gives the usual bag union (add the exponents).

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

A *bag* on X is a set of form $\{x_1^{m_1}, \ldots, x_k^{m_k}\}$ where $x_1, \ldots, x_k$ are distinct elements of X and each m_i is an integer > 0, so it is just like an ordered bag except that the order is thrown away. Bags also form a monad **Bag** using exactly the constructions of **oBag** except that all lists are replaced by sets, and similarly for the various derived operations. Thus leftmost first occurrence collection is replaced with ordinary collection of exponents.

Example 2.13. Finite-support probability distributions provide a non-standard collection class, which offers the intriguing possibility of writing programs that compute probabilities symbolically. A (finite-support) probability distribution on X is like a bag except that the integer weights are replaced by probabilities. Here it is more normal to write a formal convex combination $\sum \lambda_x x$ instead of $x_1^{\lambda_1} \ldots x_k^{\lambda_k}$ (where $\lambda_x = \lambda_i$ if $x = x_i$ and $\lambda_x = 0$ if x is not one of the x_i). Thus, define a monad **Prob** as follows. Let $Prob\, X$ be the set of finite-support probability distributions $\sum \lambda_x x$ with $0 \leqslant \lambda_x \leqslant 1$ and $\sum \lambda_x = 1$. Define $\eta_X(x) = x$ (that is, $1x$). Define $\alpha^{\#}(\sum \lambda_x x)$ by substituting $\alpha(x)$ for x and simplifying to a linear combination; it is well known that a convex combination of convex combinations is again convex. The operations in Table 2.1 are similar to those for **Bag**, but there is one new one that can be defined for any monad, the *marginal* operation:

$$T(X \times Y) \xrightarrow{marginal = [T pr_x, T pr_y]} TX \times TY.$$

For **Prob**,

$$marginal\ (\sum \lambda_{x,y}(x, y)) = (\sum \lambda_{x,y} x, \sum \lambda_{x,y} y)$$

assigns marginal probabilities in the usual sense, as is easily verified. (For the zipped monads defined later, it turns out that the marginal map is injective and that the inverse function on its image is the zip map – the reader may wish to try this for **List**).

An interesting example of a monad map involves **Prob**. For a fixed threshold $0 \leqslant c < 1$, define θ^c : **Prob** $\longrightarrow$ **Set** by

$$\theta^c_X(\sum \lambda_x x) = \{x \in X : \lambda_x > c\}.$$

It is not true, however, that 'equally-likely probability' defines a monad map in the other direction.

Example 2.14. Repetition-free lists form a monad **RFList**. Define $\alpha^{\#}$ as for lists but followed by leftmost-first-occurrence reduction to remove repetitions. This is the quotient monad of **List** that results from the additional equation $aba = ab$. The operations of Table 2.1 are as for lists with the exception that they are followed by reduction in each case.

Example 2.15. The monad **Set** of finite subsets (do not confuse with $\mathbb{S}$!) is the quotient monad of **RFList** obtained by removing order via the commutative law $ab = ba$. Setting $\omega = \{1, 2\} \in T(\{1, 2\})$ gives *join* $(\omega, A, B) = A \cup B$. Let X be the set $\{0, 1, 2, 3, 4, 5, 6, 7\}$ and define *test* : $X \longrightarrow X + X$ to be '$x \leqslant 4$', that is, $x \mapsto (x, 1)$ if $x \leqslant 4$ and $x \mapsto (x, 2)$ if $x > 4$. Then *select* $(\{0, 3, 6, 7\}, test, reduce)$ is $\{0, 3\}$. Here *reduce* : $X + X \longrightarrow TX$ maps $(x, 1)$ to x and maps $(x, 2)$ to $\varnothing$.

Example 2.16. There are two *trivial* monads. The first has $TX = 1$ for all X and the second has $TX = 1$ for all non-empty X but $T\varnothing = \varnothing$. In both cases there is only one way to define η and μ. The only point to defining these monads is so that we can avoid them as we generally shall ('collection monads' will be defined to be non-trivial). The following

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

result, due to Lawvere (1963) is proved in monad form in Manes (1976, pages 50–51): *If* (T, η, μ) *is a nontrivial monad, then* T *is a faithful functor and* η_X *is monic.* In fact, if $B = \{true, false\}$, then **T** is nontrivial if and only if η_B is monic, that is, if $true \neq false$; Lawvere called nontrivial monads 'consistent'.

Example 2.17. (Moggi 1971). Let Q be a fixed set of 'states'. We define the (finitary) *side-effects monad* $\mathbf{SE}_Q = (T, \eta, (-)^\#)$ as follows. TX is the set of all pairs (f, g) with $f : Q \longrightarrow X$, $g : Q \longrightarrow Q$, and $f(Q)$ a finite subset of X. Intuitively, Q is the store, f is the output map and g is the state-transition map. We will eventually see that what an (f, g) 'collects' is its set $f(Q)$ of outputs but that, even so, $\mathbf{SE}_Q$ is not a collection monad. Define the monad structure by $\eta_X(x) = (\hat{x}, id_Q)$, where $\hat{x}$ is constantly x. Given $\alpha : X \longrightarrow TY$ write $\alpha(x) = (t_x, u_x)$. Given $f : Q \longrightarrow X$, $g : Q \longrightarrow Q$ define $\alpha^\#(f, g) = (r, s)$ with $r : Q \longrightarrow Y$, $s : Q \longrightarrow Q$ defined by $r(q) = t_{fq}(gq)$, $s(q) = u_{fq}(gq)$. As each t_x has finite image and as there are only finitely-many different fq, r has finite image. The monad axioms are routinely verified. For $h : X \longrightarrow Y$, $(Th)(f, g) = (hf, g)$.

Example 2.18. An unstructured but not entirely useless notion of collection is atomic elements that cannot be further collected. The corresponding monad is $\mathbf{Atom} = (id, id, id)$, that is, (T, η, μ) with $TX = X$, $Tf = f$, $\eta_X = id_X$, $\mu_X = id_X$. Since $\eta : \mathbf{Atom} \longrightarrow \mathbf{T}$ is a monad map for every **T**, it follows that **Atom** is a submonad of every non-trivial monad.

A slightly more interesting example adds in a set E of 'exceptions' giving rise to the *exceptions monad* $\mathbf{Exception}_E = (T, \eta, (-)^\#)$ with $TX = X + E$, $\eta_X = in_X : X \longrightarrow X + E$ and $\alpha^\#(x) = \alpha(x)$, $\alpha^\#(e) = e$. When E has one element, the Kleisli category is isomorphic to the category of sets and partial functions.

Example 2.19. Let M be a monoid. Since we have 'time' in mind (in effect the positive real numbers under addition), we write the monoid operation as $+$ and the unit as 0, but commutativity need not be assumed. The monad $\mathbf{Timing}_M$ augments **Atom** with time tags from M. Formally, the monad is (T, η, μ), defined by $TX = M \times X$, $Tf = id_M \times f$, $\eta_X(x) = (x, 0)$, $\mu_X(t, (u, x)) = (t + u, x)$.
Notice that when a sequence of maps in the Kleisli category is composed, the 'times' are added together to give a total time.

In Beck (1967), Jon Beck made precise the question 'given monads **S** and **T**, when is the composition functor TS part of a monad' and solved it in terms of 'distributive laws' that are natural transformations of form $\delta : ST \longrightarrow TS$ subject to further axioms. The terminology stems from an interesting example in which the distributive law of ring theory is expressed as a distributive law in Beck's sense that produces the monad whose algebras are rings from those whose algebras are abelian groups and monoids. A careful look at how distributive laws interact with the concepts of this paper requires much analysis and is the proper subject for another paper, but we say a little more here to indicate the importance of the notion. First of all, it is easily stated how δ gives the monad structure. Define

$$X \xrightarrow{\eta_X} TSX = X \xrightarrow{\eta_X} SX \xrightarrow{\eta_{SX}} TSX$$

and define

$$TSTSX \xrightarrow{\mu_X} TSX = TSTSX \xrightarrow{T\delta_{SX}} TTSSX \xrightarrow{(\mu\mu)_X} TSX.$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

There are at least two distributive laws that interest us for general **T**. The first has $\mathbf{S} = \mathbf{Exception}_E$ with distributive law

$$\delta_X = < Tin_X, \eta_{in_E} > : TX + E \longrightarrow T(E + X).$$

The second has $\mathbf{S} = \mathbf{Timing}_M$ with $\delta_X : M \times TX \longrightarrow T(M \times X)$ (the *strength* of T (Kock 1970) given by $\delta_X(m, \omega) = (Ti_m)\omega$ where $i_m(x) = (m, x)$. The import here is that exception-adding and adding timing tags are polymorphic operations with arbitrary monad as parameter.

We conclude this section with some remarks on limits of monads. It is well known that the category of endofunctors on a complete category has all small limits and that these are constructed pointwise. Just as a limit of groups is built upon the limit at the level of sets, the category of monads and monad maps is complete with limits constructed at the level of endofunctors of sets. The limit property induces both η and μ uniquely. For example, the cartesian product $\mathbf{S} \times \mathbf{T}$ has an ordered pair of η, and extension given by

$$(X \xrightarrow{[\alpha, \beta]} SY \times TY)^{\#} = SX \times TX \xrightarrow{\alpha^{\#} \times \beta^{\#}} SY \times TY.$$

For another example, the equalizer of the monad maps *rev, id* : **List** $\longrightarrow$ **List** is the submonad of palindromes. The fact that monads behave so much like universal algebras can be explained. In an appropriate sense, finitary monads are isomorphic to algebraic theories in the sense of Lawvere (1963). Bloom and Ésik (1993) shows that algebraic theories are many-sorted universal algebras.

3. Monad Comprehensions

Given a monad **T**, each function $X_1 \times \ldots \times X_n \longrightarrow TY$ has canonical extensions of form $TX_1 \times \ldots \times TX_n \longrightarrow TY$. Certain of these were called *monad comprehensions* by Wadler (1992) since they generalized list comprehensions (Peyton Jones 1987) and we use the same term for all the related constructions defined here. Comprehensions provide an intrinsic **T**-propositional logic (see Definitions 3.11, 3.12 below). Propositions 7.2, 7.3 characterize shapelessness in terms of comprehensions (some collections such as Σ-trees have varying shapes whereas others such as probability distributions do not). The second of these results is equational: **T** is shapeless if and only if the Boolean equation $\omega \wedge false = false$ holds in the **T**-logic.

Proposition 3.1. Let **T** be a monad. For a function $\psi : TX \longrightarrow TY$ the following three statements are equivalent.

(1) The following square commutes.

$$\begin{array}{ccc} TTX & \xrightarrow{T\psi} & TTY \\ {\scriptstyle \mu_X}\downarrow & & \downarrow{\scriptstyle \mu_Y} \\ TX & \xrightarrow[\psi]{} & TY \end{array}$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

(2) $\psi = (\psi\eta_X)^\#$.
(3) $\psi = \alpha^\#$ for some α.

Proof. To see that (1) implies (2), recall the constructions of Proposition 2.2 and observe that $(\psi\eta_X)^\# = \mu_Y T\psi T_{\eta_X} = \psi\mu_X\eta_X = \psi$. For (3) implies (1), suppose $\psi = \alpha^\#$. We have $\mu_Y T\psi = (id_{TY})^\#(\eta_{TY}\alpha^\#)^\# = ((id_{TY})^\#(\eta_{TY}\alpha^\#)^\# = (\mu_Y\eta_{TY}\alpha^\#)^\# = \alpha^{\#\#} = (\alpha^\# id_{TX})^\# = \alpha^\#(id_{TX})^\# = \psi\mu_X$. □

A morphism ψ in the previous proposition is called **T**-*homomorphism*. This term comes from the theory of algebras over a monad (Eilenberg and Moore 1965).

Corollary 3.2. $\alpha^\# : TX \longrightarrow TY$ is the unique homomorphic extension of α.

Proof. Use Condition (2) of the previous proposition. □

Monad comprehensions are functions $\psi : TX_1 \times \ldots \times TX_n \longrightarrow TY$ that arise from $\alpha : X_1 \times \ldots \times X_n \longrightarrow TY$ by constructing homomorphic extensions one variable at a time in a prescribed order specified by $\sigma \in \{1,\ldots,n\}!$. We call such functions σ-*homomorphisms*. Related constructions for monads over more general categories require strength (Kock 1970, 1971). Here is the definition.

Definition 3.3. Let **T** be a monad and let $\sigma \in \{1,\ldots,n\}!$. A function of form $\psi : TX_1 \times \ldots \times TX_n \longrightarrow TY$ is a σ-*homomorphism* if the appropriate case holds:

— If $n = 0$: Any $\psi \in TY$ is a σ-homomorphism (for the unique empty σ).
— If $n = 1$: ψ is a **T**-homomorphism.
— If $n > 1$: For every $1 \leqslant i \leqslant n$, if we fix all other variables to be an arbitrary element of $TX_{\sigma j}$ if $1 \leqslant j \leqslant i$, or of $X_{\sigma j}$ (more precisely, in the image of $\eta_{X_{\sigma j}}$) if $i < j \leqslant n$, then the resulting slice of $\psi : TX_i \longrightarrow TY$ is a **T**-homomorphism.

Proposition 3.4. For every $\sigma \in \{1,\ldots,n\}!$ and $\alpha : X_1 \times \ldots \times X_n \longrightarrow TY$ there exists a unique σ-homomorphism $\alpha^\sigma : TX_1 \times \ldots \times TX_n \longrightarrow TY$ that extends α in the sense that $\alpha^\sigma(\eta_{X_1} \times \ldots \times \eta_{X_n}) = \alpha$.

Proof. When $n = 0$, $\eta_{X_1} \times \ldots \times \eta_{X_n}$ is the identity of the terminal object, so this case is clear. The case $n = 1$ is exactly Corollary 3.2. For $n > 1$ we show that α^σ can be constructed uniquely as follows. First some notation. For $1 \leqslant i, j \leqslant n$ define A_{ij} to be $TX_{\sigma j}$ if $j \leqslant i$ but to be $X_{\sigma j}$ when $i < j$. Then set $A_0 = X_1 \times \ldots \times X_n$ and $A_i = A_{i1} \times \ldots \times A_{in}$ for $1 \leqslant i \leqslant n$. We have an inclusion chain $A_0 \subset \ldots \subset A_n$ and $A_n = TX_1 \times \ldots \times TX_n$. We shall define a corresponding chain of extensions $\psi_i : A_i \longrightarrow TY$ inductively with ψ_n the desired α^σ. Start with $\psi_0 = \alpha$. For ψ_{k+1}, if we fix variables in $A_{k+1,j}$ for all $j \neq k+1$ the resulting slice of ψ_k has unique homomorphic extension to TX_{k+1}. □

Notation 3.5. Write $\tilde{\alpha}$ for α^σ with σ the identity permutation.

Definition 3.6. A monad is *commutative* if for all α, α^σ is independent of σ. In that case $\tilde{\alpha}$ is a **T**-homomorphism in each variable.

Wadler (1992) considered the case $\tilde{\alpha}$ for α with values in Y, that is of form $X_1 \times \ldots \times X_n \xrightarrow{f} Y \xrightarrow{\eta_Y} TY$. In his notation,

$$\tilde{\alpha}(\omega_1,\ldots,\omega_n) = [f(x_1,\ldots,x_n) | (x_1 \leftarrow \omega_1),\ldots,(x_n \leftarrow \omega_n)].$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

The construction of $\tilde{\alpha}$ for monads of sets was studied in Manes (1982). That paper was not concerned with programming languages and emphasized commutative monads.

Lemma 3.7. For any monad $\mathbf{T}$ and $\sigma \in \{1,\ldots,n\}!$, the composition $TY_1 \times \ldots \times TY_n \xrightarrow{Tf_1 \times \ldots \times Tf_n} TX_1 \times \ldots \times TX_n \xrightarrow{\psi} TY \xrightarrow{\beta^\#} TZ$ is a σ-homomorphism if ψ is.

Proof. If $n = 0$, there is nothing to prove. If $n = 1$, all three maps are $\mathbf{T}$-homomorphisms, and homomorphisms are closed under composition by Proposition 3.1(2). For $n > 1$, since η is a natural transformation each $\eta_{Y_i}(y_i)$ is mapped by $Tf_1 \times \ldots \times Tf_n$ into $\eta_{X_i}(f_i y_i)$. As such, each appropriate fixing of the variables in $TY_1 \times \ldots \times TY_n$ results in an appropriate one for $TX_1 \times \ldots \times TX_n$ to make the ψ-slice a homomorphism, so we are done. □

Definition 3.8. For a monad $\mathbf{T}$ and $\sigma \in \{1,\ldots,n\}!$, the *fundamental σ-homomorphism* is

$$TX_1 \times \ldots \times TX_n \xrightarrow{\Gamma^\sigma_{X_1 \ldots X_n}} T(X_1 \times \ldots \times X_n),$$

defined as $(\eta_{X_1 \times \ldots \times X_n})^\sigma$. When $\sigma = id$, we simply write $\Gamma_{X_1 \ldots X_n}$.

Proposition 3.9. Let $\mathbf{T}$ be any monad and let $\sigma \in \{1,\ldots,n\}!$. Then α^σ coincides with the composition $\alpha^\# \Gamma^\sigma_{X_1 \ldots X_n}$ for all $\alpha : X_1 \times \ldots \times X_n \longrightarrow TY$. Moreover, Γ^σ is a natural transformation, and each monad map commutes with it.

Proof. $\alpha^\# \Gamma^\sigma_{X_1 \ldots X_n}$ is a σ-homomorphism by Lemma 3.7 and, by the definition of Γ, it extends $\alpha^\# \eta_{X_1 \times \ldots \times X_n} = \alpha$. For the second statement, consider the following commutative diagram induced by $f_i : X_i \longrightarrow Y_i$:

$$\begin{array}{ccccc}
X_1 \times \ldots \times X_n & \xrightarrow{\eta_{X_1} \times \ldots \times \eta_{X_n}} & TX_1 \times \ldots \times TX_n & \xrightarrow{\Gamma^\sigma_{X_1 \ldots X_n}} & T(X_1 \times \ldots \times X_n) \\
\downarrow{\scriptstyle f_1 \times \ldots \times f_n} & (A) & \downarrow{\scriptstyle Tf_1 \times \ldots \times Tf_n} & (B) & \downarrow{\scriptstyle T(f_1 \times \ldots \times f_n)} \\
Y_1 \times \ldots \times Y_n & \xrightarrow[\eta_{Y_1} \times \ldots \times \eta_{Y_n}]{} & TY_1 \times \ldots \times TY_n & \xrightarrow[\Gamma^\sigma_{Y_1 \ldots Y_n}]{} & T(Y_1 \times \ldots \times Y_n)
\end{array}$$

Since η is natural, (A) and (A, B) commute. Thus (B) commutes restricted to $\eta_{X_1} \times \ldots \times \eta_{X_n}$. But both paths in (B) are σ-homomorphisms by Lemma 3.7 so, since they agree on generators, they are equal.

Now let $\gamma : \mathbf{S} \longrightarrow \mathbf{T}$ be a monad map. We must show that the following square commutes:

$$\begin{array}{ccc}
SX_1 \times \ldots \times SX_n & \xrightarrow{\gamma_{X_1} \times \ldots \times \gamma_{X_n}} & TX_1 \times \ldots \times TX_n \\
\downarrow{\scriptstyle \Gamma^\sigma_{X_1 \ldots X_n}} & & \downarrow{\scriptstyle \Gamma^\sigma_{X_1 \ldots X_n}} \\
S(X_1 \times \ldots \times X_n) & \xrightarrow[\gamma_{X_1 \times \ldots \times X_n}]{} & T(X_1 \times \ldots \times X_n)
\end{array}$$

We will give the proof for $n = 2$ and $\sigma = id$. The more general case is similar. If $Y = \varnothing$, both Γ map (τ, v) to v and the desired commutativity is clear. Otherwise, for fixed $y \in Y$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Table 3.1. *Examples of* Γ_{XY}. *The dagger indicates that the monad is commutative*

• **List**:	$(xy, abc) \mapsto (x,a)(y,a)(x,b)(y,b)(x,c)(y,c)$
• **Set**†:	$(A, B) \mapsto A \times B$
• **Bag**†:	$(\{x^m, y^n\}, \{a^t, b^u\}) \mapsto \{(x,a)^{mt}, (x,b)^{mu}, (y,a)^{nt}, (y,b)^{nu}\}$
• **Atom**†:	$(x, y) \mapsto (x, y)$
• **Prob**†:	$(\sum \lambda_x x, \sum \mu_y y) \mapsto \sum \lambda_x \mu_y (x, y)$
• **Timing**$_M$	$((x,t),(y,u)) \mapsto ((x,y), t+u)$
• **SE**$_Q$	$((f,g),(\bar{f},\bar{g})) \mapsto ([f\bar{g}, \bar{f}], g\bar{g})$

we have the following commutative square because γ is a monad map:

$$\begin{array}{ccc} SX & \xrightarrow{\delta_y} & S(X\times Y) \\ {\scriptstyle \gamma_X}\downarrow & & \downarrow{\scriptstyle \gamma_{X\times Y}} \\ TX & \xrightarrow[\epsilon_y]{} & T(X\times Y) \end{array}$$

where δ_y, ϵ_y are defined by $\delta_y = (X \xrightarrow{in_y} X \times Y \xrightarrow{\eta_{X\times Y}} S(X \times Y))^\#$, $\epsilon_y = (X \xrightarrow{\delta_y} X \times Y \xrightarrow{\gamma_{X\times Y}} S(X \times Y))^\#$ and $in_y(x) = (x, y)$. As $\gamma\eta = \eta$, $\epsilon_y = (\eta_{X\times Y} in_y)^\#$, so, by definition, $\Gamma_{XY} : SX \times SY \longrightarrow S(X \times Y)$ maps τ, τ_1 to $(y \mapsto \delta_y(\tau))^\# \tau_1$ and $\Gamma_{XY} : TX \times TY \longrightarrow T(X \times Y)$ maps v, v_1 to $(y \mapsto \delta_y(v))^\# v_1$. It then suffices to prove that the following square commutes for fixed $\tau \in SX$:

$$\begin{array}{ccc} SY & \xrightarrow{(y\mapsto\delta_y)^\#} & S(X\times Y) \\ {\scriptstyle \gamma_Y}\downarrow & & \downarrow{\scriptstyle \gamma_{X\times Y}} \\ TY & \xrightarrow[(y\mapsto\epsilon_y)^\#]{} & T(X\times Y) \end{array}$$

This is immediate from the fact that γ is a monad map, because $y \mapsto \gamma_{X\times Y}\delta_y(\tau) = \epsilon_y(\gamma_X\tau)$ is precisely the previous diagram. □

Table 3.1 gives some examples of Γ_{XY}. **Timing**$_M$ is commutative if and only if M is a commutative monoid. Notice that for **Prob**, Γ_{XY} identifies the joint distributions that are independent.

Observation 3.10. Γ is related to *outer* (see Table 2.1) by the following equation involving $f : X \times Y \longrightarrow Z$, $\tau \in TX$, $v \in TY$:

$$Tf(\Gamma_{XY}(\tau, v)) = \mu_Z(outer\ (f, \tau, v)).$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Definition 3.11. Let **T** be a monad. Let $B = \{true, false\}$. We regard TB as the set of **T**-*truth values*. Define the **T**-*equality map* $eq_X : TX \times TX \longrightarrow TB$ to be $(\eta_B \chi)^\sim$, where $\chi : X \times X \longrightarrow B$ is the characteristic function of $\{(x, x) : x \in X\}$.

When $\mathbf{T} = \mathbf{Prob}$, TB may be identified with the unit interval $[0, 1]$. If Y, Z are independent random variables with respective finite distributions $\sum \lambda_x x$, $\sum \mu_x x$ on a common outcome set X, then $eq(\sum \lambda_x x, \sum \mu_x x) = \sum \lambda_x \mu_x$ is the probability that $X = Y$.

If $\mathbf{T} = \mathbf{RFList}$, we have

$$\begin{aligned} eq_X(\omega, \tau) &= \textit{true} \text{ if } \omega, \tau \text{ are the same 1-element list} \\ &= \textit{false} \text{ if } \omega, \tau \text{ have no elements in common} \\ &= \textit{true false} \text{ if } \omega \neq \tau \text{ have the same first element} \\ &= \textit{false true} \text{ otherwise} \end{aligned}$$

Definition 3.12. For $B = \{True, False\}$, a morphism $X \longrightarrow TB$ is a **T**-*proposition* on X. Each $\omega \in TX$ is 'represented' by the proposition $\hat{\omega}(x) = eq_X(\omega, \eta_X(x))$, and this representation is faithful in many cases, as the reader may wish to explore. It is thus natural to view the set TB together with all $\tilde{\alpha} : TB^n \longrightarrow TB$ as α ranges over all $B^n \longrightarrow B$ to be the *propositional logic* of **T**, or **T**-*logic* for short. The equations of Boolean algebra may not hold when the terms are extended to TB. It is shown in Manes (1982, 6.10) that if **T** is commutative, any equation both of whose terms have the same set of variables and with no repetitions of variables will hold in TB if it holds in B.

For example, if $\mathbf{T} = \mathbf{Prob}$ so that $TB = [0, 1]$,

$$\begin{aligned} p' &= 1 - p \\ p \wedge q &= pq \\ p \vee q &= p + q - pq. \end{aligned}$$

For $\mathbf{T} = \mathbf{Set}$, TB has four elements namely *true, false, undefined* $(= \varnothing)$ and *maybe* $(= B)$. One checks, for example, that $\vee$ is commutative and so is defined by

$$\begin{aligned} \textit{undefined} \vee A &= \textit{undefined} \\ \textit{true} \vee A &= \textit{true} \\ \text{false} \vee A &= A \\ \textit{maybe} \vee \textit{maybe} &= \textit{maybe}. \end{aligned}$$

For $\mathbf{SE}_Q$, $eq_X((f, g), (\bar{f}, \bar{g})) = (\{q \in Q : f\bar{g}q = fq\}, g\bar{g})$.

4. Collection monads

Definition 4.1. A monad **T** is a *collection monad* if it is nontrivial and finitary, and the following two axioms are satisfied.

Axiom 1. members exist Every $\omega \in TX$ has a minimum support. Such minimum support is called the set of *members* of ω, written $mem\, \omega$.

Axiom 2. members are collected for $\alpha : X \longrightarrow TY$, $\omega \in TX$, $mem(\alpha^\#(\omega)) = \bigcup (mem(\alpha(x)) : x \in mem\, \omega)$.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Moggi (1995, Section 2) considers monads over $\mathbb{S}$ in connection with set-valued semantics for his evaluation logic EL_T, T being the monad. Elements of TX are thought of as programs rather than collections. Moggi's construction $\{x : \tau \Downarrow x\}$ produces $mem\,\tau$ in our examples.

Example 4.2. $\mathbf{T}_\Sigma$ is a collection monad. The members of ω are its leaves.

Example 4.3. Let $\Sigma_1 = \{u\}$ with E the single equation $ua = ub$. Then $T_{\Sigma,E}\varnothing = \varnothing$, whereas for $x \in X$ the E-equivalence class $[u(x)]$ has every singleton in X as minimal support. Hence $\mathbf{T}_{\Sigma,E}$ is not a collection monad, because members do not exist.

Example 4.4. Let $\mathbf{T}$ be the free group monad. Explicitly, TX is the set of all reduced words in $(X \bigcup X^{-1})^*$ (where X^{-1} means $\{x^{-1} : x \in X\}$, a disjoint copy of X) resulting from the reduction rules $xx^{-1} = x^{-1}x = \Lambda$, for Λ the empty word. It is well known that different reduction strategies produce the same result. Here members exist as the set of x appearing in the reduced word, but Axiom 4.1(2) fails: if $\omega = ab \in T\{a,b\}$ and if $\alpha(a) = x$, $\alpha(b) = x^{-1}$, $\bigcup(mem(\alpha(x)) : x \in mem\,\omega) = \{x\}$ but $mem(\alpha^\#(\omega)) = \varnothing$.

How well has the definition of collection monad captured the intent of Axioms 1.1, 1.2, 1.3? Axiom 1.3 is directly expressed by Axiom 4.1(2). We will confirm now that Axioms 1.1, 1.2 are valid. Their proof does not use Axiom 4.1(2).

Axiom 1.1 is formalized by the requirement that $\mathbf{T}$ is finitary, since then the minimum support $mem\,\omega$ of collection members is finite.

Axiom 1.2 is formalized by

$$mem(x^\star) = \{x\}. \tag{4.5}$$

(Write $x^\star$ for $\eta_X(x)$ as in Proposition 2.3, since $x \in X$ may be thought of as a function from a one-element set). We see that this is true as follows. That $\{x\}$ is a support of $x^\star$ is clear, but we must show that $x^\star \notin T\varnothing$. Suppose otherwise, and let $y \in X$ be arbitrary. Let $f : X \longrightarrow X$ be any function mapping x to y. If $e : \varnothing \longrightarrow X$ is the inclusion, there exists $\omega \in T\varnothing$ with $(Te)\omega = x^\star$ by hypothesis. As $fe = e$ and η is natural, we have

$$\begin{aligned} \eta_X(y) &= \eta_X(f(x)) = Tf(x^\star) = (TfTe)(\omega) \\ &= Te(\omega) = x^\star. \end{aligned}$$

As y is arbitrary, η_X is not monic, which is the desired contradiction.

Axiom 4.1(2) can be expressed in an interesting way as follows. Write $\mathbf{Set} = (\mathscr{F}, \eta, \mu)$ so that $\mathscr{F}X$ is the set of finite subsets of X. A typical collection of collections is $\omega \in TTX$. We flatten such ω as $\mu_X(\omega) \in TX$, and the set of members of the flattened collection is then given by evaluating

$$TTX \xrightarrow{\mu_X} TX \xrightarrow{mem_X} \mathscr{F}X \tag{4.6}$$

on ω, where mem_X maps a collection to its members. On the other hand, the function $T\,mem_X : TTX \longrightarrow T\mathscr{F}X$ maps ω to the collection of subcollection member-sets. Evaluating this under $mem_{\mathscr{F}X} : T\mathscr{F}X \longrightarrow \mathscr{F}\mathscr{F}X$ produces the set of subcollection member-sets whose union will be the set of all subcollection members. But for **Set**, μ is union. Hence the formalization of the third axiom is that the morphisms in (4.6) and (4.7)

$$TTX \xrightarrow{T\,mem_X} T\mathscr{F}X \xrightarrow{mem_{\mathscr{F}}X} \mathscr{F}\mathscr{F}X \xrightarrow{\mu_X} \mathscr{F}X \tag{4.7}$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

are equal. But now note that the equality of (4.6, 4.7) is just the commutativity of square (B) in the definition 2.4 of monad map. Square (A) there will commute because of (4.5). Hence Axioms 1.2, 1.3 on a collection monad are simultaneously captured by the requirement that

$$\mathbf{T} \xrightarrow{mem} \mathbf{Set} \text{ is a monad map.} \tag{4.8}$$

It is a mild exercise to formally show that 4.8 holds for a collection monad. In effect, the monad map *mem* implements Smalltalk's *asSet* coercion and is an example of a 'monad coercion'.

We shall now relate collection monads to concepts of elementary category theory. It is obvious that a finitary, nontrivial $\mathbf{T}$ satisfies Axiom 4.1(1) if and only if every $\omega \in TX$ has the property that if A, B are supports then so is $A \bigcap B$. From the definition of 'support' this condition is equivalently expressed: the functor T preserves binary intersections. Recall that a binary *intersection* in category theory is the pullback of two monomorphisms and a functor *preserves* an intersection if its functorial image, necessarily a commutative square, is also an intersection. We will also have use for *inverse images*, which are pullbacks of two morphisms at least one of which is a monomorphism.

Example 4.9. T_Σ preserves all pullbacks for every Σ.

What with monad implementations and monad coercions, we shall be considering many transformations between collection monads. Some of these are monad maps, some of these are natural transformations of various types. In passing back and forth with these transformations we shall, not surprisingly, be concerned with how members are preserved.

Definition 4.10. Let $F, G : \mathbb{K} \longrightarrow \mathbb{L}$ be functors. A $\mathbb{K}$-object-indexed collection of $\mathbb{L}$-morphisms $\gamma_X : FX \longrightarrow GX$ is a *mono-transformation* if γ is a natural transformation between the restrictions of F, G to the subcategory of monomorphisms of $\mathbb{K}$. Cartesian mono-transformations will be important in this paper, so we abbreviate them with the name *taut*. To recapitulate, while taut transformations γ are not required to have commutative squares $Gf\gamma_X = \gamma_Y Ff$ unless f is monic, such squares for monic f must be pullbacks. An example of a natural transformation that is taut but not cartesian is $\gamma : List \longrightarrow List$, $\gamma_X(w) = ww$. The inclusion from repetition-free lists to lists is a taut transformation that is not natural. The natural transformation $\gamma : List \longrightarrow List$, $\gamma_X(\Lambda) = \Lambda$, $\gamma_X(x_1, \ldots x_n) = x_1$ is not taut by Proposition 4.12 below.

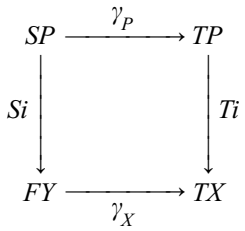
Example 4.11. Let $\Omega_n \subset \Sigma_n$ for all n. Let $j_X : T_\Omega X \longrightarrow T_\Sigma X$ be the obvious inclusion. Then $j : \mathbf{T}_\Omega \longrightarrow \mathbf{T}_\Sigma$ is a cartesian submonad. For example, the pullback property asserts that if when $f(x)$ is substituted for each member x of a Σ-expression τ the result is an Ω-expression, then τ was originally an Ω-expression. A formal inductive proof can be given by noting that $j_X(x) = x$, $j_X(\omega(\tau_1, \ldots, \tau_n)) = \omega((j_X\tau_1), \ldots, (j_X\tau_n))$.

The following will find frequent use.

Proposition 4.12. Let $\mathbf{S}$, $\mathbf{T}$ be collection monads and let $\gamma : S \longrightarrow T$ be a mono-transformation. Then $mem(\gamma_X\tau) \subset mem\,\tau$ for all $\tau \in TX$. Furthermore, $mem(\gamma_X\tau) = mem(\tau)$ for all $\tau \in TX$ if and only if γ is taut.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Proof. Given monic $i : P \longrightarrow X$, the commutativity of the square



asserts that for $\tau \in SX$, if P supports τ, then P supports $\gamma_X \tau$ so that $mem(\gamma_X \tau) \subset mem\, \tau$. Additionally, the square is a pullback if and only if whenever P supports $\gamma_X \tau$, then P supports τ, in which case $mem(\gamma_X \tau) = mem\, \tau$. □

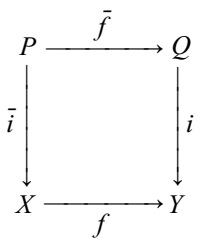
Definition 4.13. An equation in E is *balanced* if the same set of variables occurs in both terms of the equation. The presentation (Σ, E) is *balanced* if all equations in E are balanced.

Let $V = \{v_1, v_2, v_3, \ldots\}$ be the set of standard variables. Let V_n denote $\{v_1, \ldots, v_n\}$. The following result provides a practical way to identify and construct collection monads.

Theorem 4.14. A monad $\mathbf{T} = (T, \eta, \mu)$ is a collection monad if and only if it is a taut quotient of a $\mathbf{T}_\Sigma$, again, if and only if it is isomorphic to a $\mathbf{T}_{\Sigma,E}$ with (Σ, E) balanced. If $\mathbf{T}$ is a collection monad, T preserves inverse images.

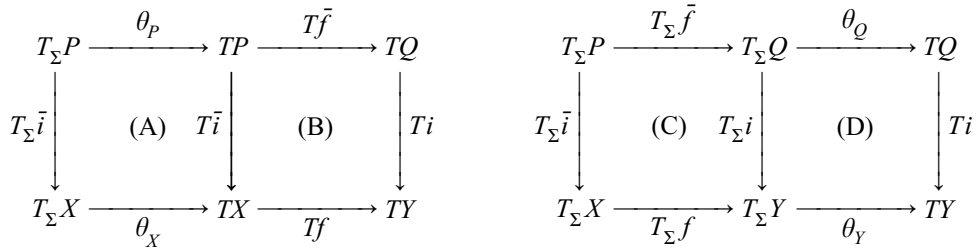
Proof. Consider an equational presentation $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}_{\Sigma,E}$. If θ is taut, then for $\sigma = \tau$ in E, $\theta_V(\sigma) = \theta_V(\tau)$, so $mem\, \sigma = mem(\theta_V(\sigma)) = mem(\theta_V(\tau)) = mem\, \tau$ and the equation is balanced. Conversely, if E is balanced, then θ preserves members (and so is taut) because according to the laws of equational logic (Taylor 1979), it is impossible to derive an unbalanced equation from balanced equations. This shows, so far, that taut quotients of $\mathbf{T}_\Sigma$ correspond to balanced presentations.

Now suppose that $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ is a taut, pointwise surjective monad map. That $\mathbf{T}$ is finitary is clear. To see that $\mathbf{T}$ is nontrivial, we use Proposition 4.12 to show that each η_X is injective as follows: for $x \in X$, $mem(\eta_X(x)) = mem(\theta_X \eta_{\Sigma,X}(x)) = mem(\eta_{\Sigma,X}(x)) = \{x\}$. Now we must show T preserves inverse images; since then T preserves intersections in particular, this would show that members exist. Let $i : Q \longrightarrow Y$ be monic, $f : X \longrightarrow Y$ arbitrary giving rise to the inverse image pullback square



https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Then the following rectangles commute and have equal perimeters because θ is a natural transformation:

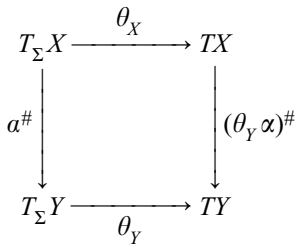


Then (C) is a pullback by Example 4.9 and (D) is a pullback because θ is taut. Thus (A, B) is a pullback. $T\bar{i}$ is monic because T preserves monomorphisms. θ_X is surjective and so split epic. It follows from the Pullback Lemma (Lemma 1.8(3)) that (B) is a pullback as desired. To complete the verification that $\mathbf{T}$ is a collection monad, it must be shown that $mem\,\mathbf{T} \longrightarrow \mathbf{Set}$ is a monad map. Now $mem_\Sigma : \mathbf{T}_\Sigma \longrightarrow \mathbf{Set}$ is a monad map by Example 4.2 and, for each X, $mem_X\,\theta_X = mem_{\Sigma,X}$ because θ_X preserves members, so use Lemma 2.6.

Conversely, let $\mathbf{T}$ be a collection monad. We will construct $\mathbf{T}$ as a taut quotient of a $\mathbf{T}_\Sigma$. Define $\Sigma_n = \{\omega \in TV_n : mem(\omega) = V_n\}$. Define $\theta_X : T_\Sigma X \longrightarrow TX$ inductively by

$$\begin{aligned} \theta_X(x) &= x \\ \theta_X(\omega(\tau_1,\ldots,\tau_n) &= (V_n \xrightarrow{<\theta_X\tau_i>} TX)^\#\omega, \end{aligned}$$

where $\langle\theta_X\tau_i\rangle$ is tuple notation for $i \mapsto \theta_X\tau_i$. That $\theta\eta = \eta$ is given. We will show that $\theta : \mathbf{T}_\Sigma :\longrightarrow \mathbf{T}$ is a taut monad quotient, which we call the *canonical presentation* of $\mathbf{T}$. For $\alpha : X \longrightarrow T_\Sigma Y$ we must show



We do this by induction. Both paths map $x \in X$ to $(\theta_Y\alpha)x$. For the inductive step,

$$\begin{aligned} (\theta_Y\alpha)^\#\omega(\tau_1,\ldots,\tau_n) &= (\theta_Y\alpha)^\# <\theta_X\tau_i>^\# \omega \\ &= ((\theta_Y\alpha)^\# <\theta_X\tau_i>)^\#\omega \\ &= <\theta_Y\alpha^\#\tau_i>^\# \omega \text{ (induction hypothesis)}, \end{aligned}$$

whereas, by the definition of $\alpha^\#$ in $\mathbf{T}_\Sigma$,

$$\begin{aligned} \theta_Y\alpha^\#(\omega(\tau_1,\ldots,\tau_n)) &= \theta_Y\omega((\alpha^\#\tau_1),\ldots,(\alpha^\#\tau_n))) \\ &= <\theta_Y\alpha^\#\tau_i>^\# \omega \end{aligned}$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

as well. Thus θ is a monad map. We next must show that θ_X is surjective. Let $\tau \in TX$. As $\mathbf{T}$ is a collection monad there exists monic $f : V_n \longrightarrow X$ and $\omega \in TV$ with $(Tf)\omega = \tau$ and $mem(\omega) = V_n$. Thus $\omega \in \Sigma_n$ and $\omega(v_1, \ldots, v_n) \in T_\Sigma V_n$. We have the naturality square

$$\begin{array}{ccc} T_\Sigma V_n & \xrightarrow{\theta_{V_n}} & TV_n \\ {\scriptstyle T_\Sigma f}\downarrow & & \downarrow{\scriptstyle Tf} \\ T_\Sigma X & \xrightarrow[\theta_X]{} & TX \end{array}$$

so $\tau = (Tf)\omega = (Tf)\theta_{V_n}(\omega(v_1, \ldots, v_n)) = (\theta_X T_\Sigma f)(\omega(v_1, \ldots, v_n))$ is in the image of θ_X. Finally, we must show θ is taut, that is, for $\tau \in T_\Sigma X$ that $mem(\theta_X(\tau)) = mem(\tau)$. Use induction. For $\tau = x$, both are $\{x\}$. For the inductive step, consider $\tau = \omega(\tau_1, \ldots, \tau_n)$. Then, because $mem : \mathbf{T} \longrightarrow \mathbf{Set}$ is a monad map,

$$\begin{aligned} mem(\theta_X \tau) &= mem_X((V_n \xrightarrow{< \theta_X \tau_i >} TX)^\# \omega) \\ &= (mem_X < \theta_X \tau_i >)^\# \, mem_{V_n} \omega \\ &= (mem_X < \theta_X \tau_i >)^\# V_n \text{ (definition of } \Sigma_n) \\ &= \bigcup (mem_X(\theta_X \tau_i) : i = 1, \ldots, n) \\ &= \bigcup (mem_X \, \tau_i : i = 1, \ldots, n) \text{ (induction hypothesis)} \\ &= mem_X \, \tau. \end{aligned}$$

□

In view of the preceding result, we call a taut monad quotient $\mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ a *taut presentation* of $\mathbf{T}$. The canonical presentation of the proof above is an example of a taut presentation; this presentation will be needed later.

We warn the reader that not every equational presentation of a collection monad need be balanced. It is always possible, for example, to add a new binary symbol ω and then make it redundant with the unbalanced equation $\omega(a, b) = a$.

The following simplification will be used in the proof of Proposition 5.3.

Lemma 4.15. Let $\mathbf{T}$ be a finitary monad. Then members exist (that is, Axiom 4.1(1) holds) if and only if whenever $\tau \in TX$ has two disjoint supports, $\varnothing$ is a support of τ.

Proof. We must show that T preserves binary intersections. The hypothesis is that T maps empty intersections to pullbacks, so it suffices to prove that T maps non-empty intersections to pullbacks. But this is true for all functors from $\mathbb{S}!$, since it is a routine, exercise to show that every non-empty intersection can be embedded as the square (A) in Lemma 1.8(4) and every functor preserves such a diagram. □

Lemma 4.16. For a collection monad (T, η, μ), Tf preserves members for every $f : X \longrightarrow Y$, that is, the members of $(Tf)\,\tau$ are the set of all $f(x)$ as x ranges over $mem\,\tau$.

Proof. The monad map $mem : T \longrightarrow Set$ is a natural transformation. □

We conclude this section with examples.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Definition 4.17. (Eilenberg 1974) A *positive semiring* is a semiring with unit with the properties that if $x + y = 0$, then $x = y = 0$, and if $xy = 0$, then $x = 0$ or $y = 0$.

Example 4.18. Let R be a positive semiring with $1 \neq 0$. The monad $\mathbf{oT}_R$ (as with **oBag** the 'o' is for 'ordered') of *R-lists* is the monad $\mathbf{T}_{\Sigma,E}$ corresponding to the equational presentation given by $\Sigma_2 = \{+, \cdot\}$ (with $\cdot$ written as concatenation xy), $\Sigma_1 = R - \{0\}$, and $\Sigma_0 = \{\Lambda\}$ and with equations

$$\begin{aligned}
(x + y) + z &= x + (y + z) \\
x + \Lambda &= x = \Lambda + x \\
1x &= x \\
\lambda(x + y) &= \lambda x + \lambda y \\
(\lambda\mu)x &= \lambda(\mu x) \\
\lambda x + y + \mu x &= (\lambda + \mu)x + y \text{ ('leftmost collection law').}
\end{aligned}$$

Conspicuously absent are $x + y = y + x$ and $0x = \Lambda$; $0x$ is not even defined and this presents no problem above since $\lambda + \mu \neq 0 \neq \lambda\mu$ when $\lambda \neq 0 \neq \mu$.

This equational presentation is not useful in describing a collection class unless we find a useful canonical representation for the instance collections. This amounts to solving the word problem. The simplest way is to guess, based on experience with modules over a ring, and then show that the resulting candidate for $oT_RX = T_{\Sigma,E}X$ is indeed the free ordered R-module (a good name for the (Σ, E)-algebras here, even if they are not ordered sets) generated by X. To this end, define oT_RX to be the set of all strings in $((R - \{0,1\}) \times X + X)^*$ in which no element of X repeats. For convenience we assume that $R \bigcap X = \varnothing$ and sometimes write $x \in X$ as $(1, x)$ so that a general non-empty list may be written as $(\lambda_1, x_1) \ldots (\lambda_n, x_n)$ with $\lambda_i \in R$ and $x_i \in X$. This is a Σ-algebra as follows. The empty string interprets the constant Λ. For $\lambda \in \Sigma_1$, define $\lambda(\lambda_1, x_1) \ldots (\lambda_n, x_n) = (\lambda\lambda_1, x_1) \ldots (\lambda\lambda_n, x_n)$. Define $+$ to be concatenation followed by leftmost-first-occurrence collection with the addition of coefficients, for example,

$$(\lambda, x)(\mu, y)(\phi, x)(\tau, z)(\theta, y) \mapsto (\lambda + \phi, x)(\mu + \theta, y)(\tau, z).$$

It is then a somewhat tedious but entirely routine calculation to show that this is the free ordered R-module generated by X with inclusion of the generators $\eta_X(x) = (1, x)$. The extension operation, specializing the construction used in proving freeness, obtains by substituting the ordered linear combination $\alpha(x)$ for each variable x and simplifying using leftmost-first-occurrence collection.

The monad $\mathbf{T}_R$ of *R-sets* is then obtained as the taut quotient of $\mathbf{oT}_R$ by imposing the further balanced equation $x + y = y + x$. With this new equation set F, the (Σ, F)-algebras are the usual R-modules with T_RX the usual set of (unordered) R-linear combinations of elements of X.

We recapture earlier examples by specializing R. If $R = \{0, 1\}$ is the Boolean semiring with $1 + 1 = 0$, then $\mathbf{oT}_R$ is **RFList**, and $\mathbf{T}_R$ is **Set**. If $R = \mathbb{N}$ is the semiring of natural numbers 0, then $\mathbf{oT}_R$ is **oBag** and $\mathbf{T}_R$ is **Bag**.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Example 4.19. For $\mathbf{SE}_Q$, given $h : P \longrightarrow X$ and $(f,g) \in TP$, $(Th)(f,g) = (hf,g)$. Now consider the case that h is the inclusion of the subset P of X to see that P supports (f,g) if and only if f factors through P. It follows that $mem(f,g)$ exists and is the image of f. But this monad is not a collection monad. Let $\alpha : X \longrightarrow TY$ and write $\alpha(x) = (t_x : Q \longrightarrow X, u_x : Q \longrightarrow Q)$. For $(f,g) \in TX$, $\alpha^{\#}(f,g)$ has members $\{t_{fq}(gq) : q \in Q\}$, which, in general, is a smaller set then the union of all $mem(\alpha(x))$ as x ranges over $mem(f,g)$, which is $\bigcup t_{fq}(Q)$.

5. Submonads

Submonads are an important way to create new monads from old and we explore their properties and examples in this section. The first result is somewhat surprising.

Theorem 5.1. Every submonad of a collection monad is taut.

Proof. Let $\gamma : \mathbf{S} \longrightarrow \mathbf{T}$ be a submonad with $\mathbf{T}$ finitary. Let $\tau \in SX$, $P = mem\,\tau$ and let $\sigma \in X!$ be a bijection of X. The image factorization of the restriction of σ to P gives rise to the commutative squares of monomorphisms

$$\begin{array}{ccc} P & \xrightarrow{e} & \sigma(P) \\ {\scriptstyle i}\downarrow & & \downarrow{\scriptstyle j} \\ X & \xrightarrow[\sigma]{} & X \end{array} \qquad \begin{array}{ccc} SP & \xrightarrow{Se} & S(\sigma(P)) \\ {\scriptstyle Si}\downarrow & & \downarrow{\scriptstyle Sj} \\ SX & \xrightarrow[S\sigma]{} & SX \end{array}$$

and this shows that $\sigma(P)$ is a support of $(S\sigma)\tau$. If any proper subset $Q \subset \sigma(P)$ supported $(S\sigma)\tau$, then, using σ^{-1} similarly, $\sigma^{-1}(Q)$ would be a proper subset of P supporting τ, which is impossible. We conclude

$$For\ every\ \sigma \in X!,\ mem((S\sigma)\tau) = \sigma(mem\,\tau). \tag{5.2}$$

This aside, we must show that $\gamma_X : SX \longrightarrow TX$ preserves members. Let $\tau \in SX$, $P = mem\,\tau$, $Q = mem(\gamma_X\tau)$. As γ is a mono-transformation, $Q \subset P$. Now suppose there exists $x \in P$ but $x \notin Q$, and seek a contradiction. We may assume without loss that P is a proper subset of X, since if $X \subset Y$ with inclusion k, $mem\,\tau = mem((Tk)\tau)$. Let $\sigma \in X!$ fix the elements of Q and map x outside of P. We obtain a commutative diagram

$$\begin{array}{ccccc} SX & \xrightarrow{\gamma_X} & TX & \xrightarrow{Tj} & TQ \\ {\scriptstyle S\sigma}\downarrow & & {\scriptstyle T\sigma}\downarrow & & \downarrow{\scriptstyle id} \\ TQ & \xrightarrow[\gamma_X]{} & TX & \xrightarrow[Tj]{} & TQ \end{array}$$

As $mem((S\sigma)\tau) = \sigma(P) \neq P$, $(S\sigma)\tau \neq \tau$. But $\gamma_X(S\sigma)\tau = (T\sigma)\gamma_X\tau = \gamma_X\tau$ so γ_X is not monic. □

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Proposition 5.3. The class of collection monads is closed under submonads and taut monad quotients.

Proof. Given a taut monad quotient $\gamma : \mathbf{S} \longrightarrow \mathbf{T}$ with $\mathbf{S}$ a collection monad, there exists a taut monad quotient $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{S}$ so that the composition $\gamma\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ is a taut monad quotient and $\mathbf{T}$ is a collection monad.

Now let $\gamma : \mathbf{S} \longrightarrow \mathbf{T}$ be a submonad with $\mathbf{T}$ a collection monad. Let $\tau \in SX$ and let $P = mem(\gamma_X(\tau))$ with inclusion function $i : P \longrightarrow X$. By Theorem 5.1, γ is taut, so the naturality square

$$\begin{array}{ccc} SP & \xrightarrow{\gamma_P} & TP \\ {\scriptstyle Si}\downarrow & & \downarrow{\scriptstyle Ti} \\ SX & \xrightarrow[\gamma_X]{} & TX \end{array}$$

is a pullback. This shows that P is a finite support of τ. As $\gamma_X\eta_{S,X} = \eta_{T,X}$ is monic, so is $\eta_{S,X}$, so $\mathbf{S}$ is non-trivial. To see that members exist, we use Lemma 4.15. Let P, Q be disjoint supports of $\tau \in \mathbf{S}X$. As γ is natural, P, Q are disjoint supports of $\gamma_X(\tau)$, so $\gamma_X(\tau) \in T\varnothing$. Setting $P = \varnothing$ in the pullback square above then shows that $\varnothing$ is a support of τ. Since the proof of Proposition 4.12 does not use Axiom 4.1(2), we now have that γ preserves members. This gives that the restriction of the monad map $mem : \mathbf{T} \longrightarrow \mathbf{Set}$ to $\mathbf{S}$, necessarily a monad map, is $mem : \mathbf{S} \longrightarrow \mathbf{Set}$. □

Example 4.3 is a finitary, non-trivial monad that is not a collection monad and yet is a quotient of a collection monad of form $\mathbf{T}_\Sigma$. This shows that collection monads are not closed under arbitrary quotients.

Cartesian product monads could be used to create records of collections. But it is not true that records of collections are collections as the following proposition shows.

Proposition 5.4. For every collection monad $\mathbf{T}$, the product monad $\mathbf{T} \times \mathbf{T}$ is not a collection monad.

Proof. As $\mathbf{Atom} \times \mathbf{Atom}$ is a submonad of $\mathbf{T} \times \mathbf{T}$ via $\eta \times \eta$ for every collection monad $\mathbf{T}$, it suffices to show that $\mathbf{Atom} \times \mathbf{Atom} = (S, \eta, \mu)$ does not satisfy Axiom 4.1(2). Now $SX = X \times X$, $\eta_X(x) = (x, x)$ and, if α, $\beta : X \longrightarrow Y$ so that $[\alpha, \beta] : X \longrightarrow Y \times Y$ is a typical function, $[\alpha, \beta]^\#(x_1, x_2) = (a, d)$ if $\alpha(x_1) = (a, b)$ and $\beta(x_2) = (c, d)$. For $B \subset Y$ with inclusion $i : B \longrightarrow Y$, $Ti : TB \longrightarrow TY$ is the inclusion $B \times B \longrightarrow Y \times Y$. Thus it is clear that members exist and $mem(a, b) = \{a, b\}$. But then if $\alpha : \{1, 2\} \longrightarrow \{a, b, c, d\}$ maps 1 to (a, b) and 2 to (c, d), then $mem(\alpha^\#((1, 2)) = mem(a, d) = \{a, d\}$, which takes on a different value from $\bigcup(mem(\alpha(x)) : x = 1, 2) = \{a, b\} \bigcup \{c, d\} = \{a, b, c, d\}$. □

There is no known characterization of submonads in terms of equational presentation. The following result provides a little information.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Lemma 5.5. Presentation Lemma for Submonads Let $\mathbf{T}$ be a collection monad and let $\gamma : \hat{\mathbf{T}} \longrightarrow \mathbf{T}$ be a submonad. Then there exists a commutative diagram of monad maps

$$\begin{array}{ccc} \mathbf{T}_{\hat{\Sigma}} & \xrightarrow{\;j\;} & \mathbf{T}_{\Sigma} \\ \downarrow{\scriptstyle \hat{\theta}} & & \downarrow{\scriptstyle \theta} \\ \hat{\mathbf{T}} & \xrightarrow[\;\gamma\;]{} & \mathbf{T} \end{array}$$

with $\theta, \hat{\theta}$ the respective canonical presentations, and j a cartesian submonad.

Proof. For convenience, assume $\hat{T}X \subset TX$. Then, by the definition of the canonical presentations in the proof of Theorem 4.14 and because of Theorem 5.1, $\hat{\Sigma}_n \subset \Sigma_n$ for all n. Let j be the cartesian submonad of Example 4.11. Both paths of the diagram map $x \in X$ to x. Proceeding inductively, let $\omega\tau_1 \ldots \tau_n \in T_{\hat{\Sigma}}X$, so that $\omega \in \hat{T}\bar{n}$ with $mem\,\omega = \bar{n}$ and $\tau_1, \ldots, \tau_n \in T_{\hat{\Sigma}}X$. As γ is a submonad, we have the commutative square

$$\begin{array}{ccc} FX & \xrightarrow{\;\gamma_X\;} & GY \\ \downarrow{\scriptstyle (\bar{n} \xrightarrow{\langle \hat{\theta}\tau_i \rangle} \hat{T}X)^{\#}} & & \downarrow{\scriptstyle (\bar{n} \xrightarrow{\langle \hat{\theta}_X\tau_i \rangle} \hat{T}X \xrightarrow{\gamma_X} TX)^{\#}} \\ FY & \xrightarrow[\;\gamma_Y\;]{} & GY \end{array}$$

Therefore,

$$\begin{aligned} \gamma_X\hat{\theta}_X(\omega(\tau_1, \ldots, \tau_n)) &= \gamma_X < \hat{\theta}_X\tau_i >^{\#} \omega \\ &= (\gamma_X < \hat{\theta}_X\tau_i >)^{\#}\gamma_{\bar{n}}\omega \\ &= < \theta_X j_X\tau_i >^{\#} \gamma_{\bar{n}}\omega \text{ (induction hypothesis)} \\ &= \theta_X((\gamma_{\bar{n}}\omega)((j_X\tau_1), \ldots, (j_X t_n)) \\ &= \theta_X j_X(\omega(\tau_1, \ldots, \tau_n). \end{aligned}$$

□

In Manes (1992, Definition 2.13) a morphism $f : X \longrightarrow Y$ in any category is defined to be *deterministic* if for every coproduct $Q \longrightarrow Y \longleftarrow Q'$ there exists a commutative diagram

$$\begin{array}{ccccc} P & \longrightarrow & X & \longrightarrow & P' \\ \downarrow & & \downarrow{\scriptstyle f} & & \downarrow \\ Q & \longrightarrow & Y & \longrightarrow & Q' \end{array}$$

where the top row is a coproduct. The definition may be paraphrased by 'no matter how Y is divided into disjoint alternatives, each element of f maps wholly into one

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

alternative or the other', that is, a deterministic morphism has single-valued outcomes. For example, in the Kleisli category of **Set**, the category of sets and multivalued functions, the deterministic morphisms are the partial functions. Based on the identity $\alpha \circ f^\star = \alpha f$ of Proposition 2.3, for any monad $\mathbf{T}$ the inclusion $(-)^\star : \mathbb{S} \longrightarrow \mathbb{S}_\mathrm{T}$ preserves coproducts. Thus the intuition above applies to any Kleisli category. We say the monad $\mathbf{T}$ is *deterministic* if every morphism in $\mathbb{S}_\mathrm{T}$ is deterministic. The next proposition shows that the deterministic morphisms always form a submonad.

Proposition 5.6. Let $\mathbf{T}$ be a collection monad. Then

$$T_d X = \{\tau \in TX : mem\, \tau \text{ has at most one element}\}$$

is a submonad $\mathbf{T}_d$ of $\mathbf{T}$ whose Kleisli category $\mathbb{S}_{\mathrm{T}_d}$ is isomorphic to the subcategory of deterministic morphisms of $\mathbb{S}_\mathrm{T}$.

Proof. First consider deterministic $\alpha : X \longrightarrow TY$. For $x \in X$ we must show that $\alpha(x)$ has at most one member. If there are no members we are done; otherwise, let $q \in mem(\alpha(x))$. Writing $(-)'$ for set-theoretic complement, there is a commutative diagram

$$\begin{array}{ccccc} P & \xrightarrow{i} & X & \xrightarrow{i'} & P' \\ \downarrow & & \downarrow{\scriptstyle \alpha} & & \downarrow \\ T\{q\} & \xrightarrow[Tj]{} & TY & \xrightarrow[Tj']{} & T\{q\}' \end{array}$$

whose rows are coproducts. If $x \in P'$, then $\{q\}'$ supports $\alpha(x)$, so $q \notin mem(\alpha(x))$. Thus $x \in P$ and $\{q\}$ supports $\alpha(x)$, so $mem(\alpha(x)) = \{q\}$. Next, assume that $\alpha(x)$ has at most one member for all x and show that α is deterministic. To this end, let $Q \subset Y$. It suffices to show that there is a commutative diagram

$$\begin{array}{ccccc} P & \xrightarrow{i} & X & \xrightarrow{i'} & P' \\ \downarrow & & \downarrow{\scriptstyle \alpha} & & \downarrow \\ TQ & \xrightarrow[Tj]{} & TY & \xrightarrow[Tj']{} & TQ' \end{array}$$

where $P = \{x \in X : mem(\alpha(x)) \subset Q\}$. If $x \in P$, then Q supports $\alpha(x)$, so αi factors through Tj. If $x \notin P$, then some member of $\alpha(x)$ is in Q'; but at most one member exists, so all are and $\alpha i'$ factors through Tj'.

Finally, it must be shown that $\mathbf{T}_d$ is a submonad. To this end, let $M = T1$. Identifying M with the set of functions $1 \longrightarrow T1$, M is a monoid namely the endomorphism monoid of 1 in $\mathbb{S}_\mathrm{T}$. Using no additional axioms on the monad $\mathbf{T}$, it is routine to verify that $\gamma : \mathbf{Timing}_M \longrightarrow \mathbf{T}$ defined by $\gamma_X(m, x) = T(1 \xrightarrow{x} X)(m)$ is a monad map. By Proposition 4.12, $mem\, \gamma_X(m, x) \subset \{x\}$, so $\gamma_X(m, x) \in T_d X$.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Conversely, if $\tau \in T_d X$, there exists $x \in X$ with $mem\,\tau \subset \{x\}$. As $Tx : T1 \longrightarrow T\{x\}$ is bijective, there exists $m \in T1$ with $\gamma_X(m, x) = (Tx)m$. Thus T_d is the image of the monad map γ and thus must be a submonad. □

Example 5.7. The deterministic submonad of **Set** is $\mathbf{Exception}_E$ with E a one-element set. As noted earlier, the Kleisli category is the category of sets and partial functions.

In Manes (1992, Definition 6.1) a morphism $f : X \longrightarrow Y$ in any category with initial object 0 is defined to be *null* if it factors through 0 and to be *total* if gf null $\Rightarrow g$ null. In $\mathbb{S}_{\mathbf{Set}}$ the empty relation is the unique null morphism $X \longrightarrow Y$ and the total morphisms are the multivalued functions f with $f(x) \neq \varnothing$ for all x. We say a monad $\mathbf{T}$ is *total* if each morphism $\mathbb{S}_{\mathrm{T}}$ is total.

Proposition 5.8. Let $\mathbf{T}$ be a collection monad. Then

$$T_t X = \{\tau \in TX : mem\,\tau \text{ has at least one element}\}$$

is a submonad $\mathbf{T}_t$ of $\mathbf{T}$ whose Kleisli category $\mathbb{S}_{\mathrm{T}_t}$ is isomorphic to the subcategory of total morphisms of $\mathbb{S}_{\mathrm{T}}$.

Proof. For collection monads, $mem(\eta_X(x)) = \{x\}$ and $mem((\beta \circ \alpha)(x)) = mem(\beta^{\#}(\alpha(x))) = \bigcup(mem(\beta(y)) : y \in mem(\alpha(x)))$, so it is clear that $\mathbf{T}_t$ is a submonad.

First suppose $\alpha : X \longrightarrow TY$ factors through $T_t Y$. To show α is total it suffices to show that for all $\omega \in TX$ with $\alpha^{\#}(\omega) \in T\varnothing$ that $mem\,\omega = \varnothing$. But this is immediate from

$$\varnothing = mem(\alpha^{\#}(\omega) = \bigcup(mem(\alpha(x)) : x \in mem(\omega)),$$

since each $\alpha(x)$ has a nonempty set of members.

Conversely, let $\alpha : X \longrightarrow TY$ be total. We assume there exists $x \in X$ with $\alpha(x) \in T\varnothing$ and seek a contradiction. Consider x as a morphism $x : 1 \longrightarrow X$ with 1 a 1-element set. It is routine to check, in general, that monomorphisms are total and that if $\beta\alpha$ is total, then so is α. Further, by Lemma 2.7, if $f : X \longrightarrow Y$ is injective, then $f^{\star}$ in monic in $\mathbb{S}_{\mathrm{T}}$. Thus the Kleisli composition $\alpha \circ x^{\star}$ is total. By the hypothesis on x, it is also null, so there exists $y : 1 \longrightarrow T\varnothing$ with $T_i y = \alpha x$, where $i : \varnothing \longrightarrow Y$. As y is total and $y \circ \eta_1$ is null, η_1 is null, so there exists $\psi : 1 \longrightarrow T\varnothing$ with $T_j = \eta_1$, where $j : \varnothing \longrightarrow 1$. Thus $j^{\star} \circ \psi = T_j \psi$ is the identity morphism of $\varnothing$ in the Kleisli category. As $\varnothing$ is initial, $\psi \circ j^{\star}$ is the identity as well. Thus 1 is initial in $\mathbb{S}_{\mathrm{T}}$. In particular, each TX has at most one element, so $\mathbf{T}$ is trivial, the desired contradiction. □

6. Implementation and order

In this section we define implementations of collection monads by canonical trees. Every implementable monad is ordered, that is, its members are a list rather than just a set.

Definition 6.1. Let $\mathbf{T}$ be a collection monad. An *implementation* of $\mathbf{T}$ is $(\Sigma, I, \theta, \hat{\mathbf{T}})$, where $\hat{\mathbf{T}}$ is a submonad of $\mathbf{T}_\Sigma$, $I : T \longrightarrow \hat{T}$ is a mono-transformation and $\theta : \hat{\mathbf{T}} \longrightarrow \mathbf{T}$ is a taut monad quotient subject to the equations

$$\begin{aligned} \hat{\eta} &= id \xrightarrow{\eta} T \xrightarrow{I} \hat{T} \\ id_T &= T \xrightarrow{I} \hat{T} \xrightarrow{\theta} T. \end{aligned}$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

The implementation is *full* if $\hat{\mathbf{T}} = \mathbf{T}_\Sigma$. We say $\mathbf{T}$ is *implementable* if it has an implementation. The general motivation behind the definition is as follows. As examples have shown, collection monads $\mathbf{T}$ arise naturally by imposing balanced equations on expressions giving rise to a taut monad quotient $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}$. To implement $\mathbf{T}$, pick representatives of the θ-equivalence classes to get $I : T \longrightarrow T_\Sigma$ with $\theta I = id_T$. Such I_X should be reasonably 'abstract', so they should satisfy some minimal properties. We should, for example, require a commutative square

$$\begin{array}{ccc} TX & \xrightarrow{I_X} & T_\Sigma X \\ {\scriptstyle Tf}\downarrow & & \downarrow{\scriptstyle T_\Sigma f} \\ TY & \xrightarrow[I_Y]{} & T_\Sigma Y \end{array}$$

whenever $f : X \longrightarrow Y$ is bijective, since the method of implementation should not change if members are relabelled. The diagram should also commute when f is a subset inclusion, since a collection on X should not be implemented differently when it is viewed as a collection on Y. Putting these together, the diagram should commute when f is a monomorphism and this is just exactly what we have required — that I be a mono-transformation. The final requirement that atomic collections be implemented as atomic expressions seems reasonable and does not exclude any useful examples known to us.

So far this motivates full implementations. The more general definition is justified by the following theory and examples, and can be loosely motivated by the observation that since, as a rule, only specific expressions will implement $\mathbf{T}$-collections, there is no need to have θ defined on all of $\mathbf{T}_\Sigma$ if a submonad $\hat{\mathbf{T}}$ contains all expressions needed. In general, $I_X\theta_X$ provides a canonical form for implemented expressions and the normal form algorithms used may need to constrain their inputs. Of course, being a submonad of $\mathbf{T}_\Sigma$, the operations of $\hat{\mathbf{T}}$ are still aptly described as 'tree-processing'.

Example 6.2. For any positive semiring R, the monad $\mathbf{oT}_R$ of R-Lists has a full implementation as follows. Let $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{oT}_R$ be the equational presentation of Example 4.18. Define $I_X : oT_X \longrightarrow T_\Sigma X$ by

$$\begin{aligned} I_X(\Lambda) &= \Lambda \\ I_X((\lambda_1, x_1)\ldots(\lambda_n, x_n)) &= \lambda_1 x_1 + (\ldots + (\lambda_{n-1}x_{n-1} + \lambda_n x_n)\ldots) \end{aligned}$$

except that when $\lambda_i = 1$ write x_n instead of $1x_n$. The details are easily checked. While I is a natural transformation, it is not a cartesian one, since if $f : X \longrightarrow Y$ is constantly u then

$$\theta_X(x + (x + y)) = (1 + 1, x) + y = \theta_X(x + (y + x)),$$

whereas

$$(T_\Sigma f)(x + (x + y)) = u + (u + u) = (T_\Sigma f)(x + (y + x)),$$

showing that θ_X, $T_\Sigma f$ are not jointly-monic. But the next result shows that I must be taut.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Proposition 6.3. If $(\Sigma, I, \theta, \hat{\mathbf{T}})$ implements $\mathbf{T}$, then $I : T \longrightarrow \hat{T}$ is taut.

Proof. Let $\tau \in TX$. Then $mem(I_X\tau) \subset mem\ \tau$, as I is a mono-transformation. Conversely, as θ is a natural transformation, $mem\ \tau = mem(\theta_X I_X \tau) \subset mem(I_X\tau)$. □

The next result establishes that, for implemented monads, the monad operations (and hence all operations derived from them such as those of Table 2.1) can be performed by coding (with I), tree-processing, and decoding (with θ). Since I and θ preserve η, the only issue is the extension operation.

Theorem 6.4. (Simulation Theorem) Let $(\Sigma, I, \theta, \hat{\mathbf{T}})$ implement $\mathbf{T}$. Then for $\alpha : X \longrightarrow TY$.

$$\alpha^{\#} = TX \xrightarrow{I_X} \hat{T}X \xrightarrow{(I_Y\alpha)^{\#}} \hat{T}Y \xrightarrow{\theta_Y} TY.$$

Proof. As θ is a monad map and $\theta_Y I_Y \alpha = \alpha$, we have

$$\begin{array}{ccc} \hat{T}X & \xrightarrow{(I_Y\alpha)^{\#}} & \hat{T}Y \\ {\scriptstyle\theta_X}\downarrow & & \downarrow{\scriptstyle\theta_Y} \\ TX & \xrightarrow[\alpha^{\#}]{} & TY \end{array}$$

so that for $\tau \in TX$, $\alpha^{\#}\tau = \alpha^{\#}\theta_X I_X \tau = \theta_Y (I_Y\alpha)^{\#} I_X \tau$. □

Proposition 6.5. A submonad of an implementable monad is implementable. If the old implementation I is cartesian then the new implementation I_0 may be chosen cartesian as well.

Proof. Let $i : \mathbf{S} \longrightarrow \mathbf{T}$ be a submonad and let $\mathbf{T}$ have implementation $(\Sigma, I, \theta, \hat{\mathbf{T}})$. Recalling that the category of monads and monad maps has pointwise limits, pull back $\mathbf{S}$ under θ to define the submonad j as shown in (B). Define I_0 by

$$\begin{array}{ccccc} \mathbf{S} & \xrightarrow{I_0} & \hat{\mathbf{S}} & \xrightarrow{\theta_0} & \mathbf{S} \\ {\scriptstyle i}\downarrow & (A) & {\scriptstyle j}\downarrow & (B) & \downarrow{\scriptstyle i} \\ \mathbf{T} & \xrightarrow[I]{} & \hat{\mathbf{T}} & \xrightarrow[\theta]{} & \mathbf{T} \end{array}$$

the pullback property as the unique mono-transformation with $\theta_0 I_0 = id_s$ and $j\ I_0 = I\ i$: this is possible because $\theta I\ i = i = i\ id_s$. Thus $(\Sigma, I_0, \theta_0, \hat{\mathbf{S}})$ is an implementation of $\mathbf{S}$. Moreover, (A) is a pullback, because (A, B) and (B) are. Since j is cartesian by Proposition 4.12 and cartesian is stable under pullback, I_0 is cartesian if I is. □

To implement, say, **Set** requires $I_X\{x_1, \ldots x_n\}$ to be a tree and, by Propositions 4.12 and 6.3 the set of leaves of this tree is $\{x_1, \ldots x_n\}$. These leaves must be listed in a definite order. But intuitively it is unlikely that one can assign a canonical list on the members of a given set because there is no way to assign a canonical order to an unordered collection.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

We now wish to formalize the concept of an 'ordered monad.' We shall see that every implementable monad is ordered. Other monads such as **Set**, are not ordered, and we shall deal with this situation later.

Definition 6.6. A collection monad **T** is *ordered* if there exists a taut transformation $\gamma : T \longrightarrow List$.
The definition evidently depends only on the functor part of the monad.

Example 6.7. $\mathbf{T}_\Sigma$ is ordered. Even more is true since (recall Example 2.11) $\mathbf{T}_\Sigma$ admits a cartesian monad map to **List**.

Proposition 6.8. Every implementable monad is ordered.

Proof. By Theorem 5.1, every submonad of $\mathbf{T}_\Sigma$ admits a taut transformation to **List**. Now use Proposition 6.3. □

Proposition 6.9. Let **S**, **T** be collection-monads and let $\gamma_X : S \longrightarrow T$ be taut. Then **S** is ordered if **T** is. □

The next result provides useful characterizations of ordered monads. We thank the referee for suggesting the third condition.

Theorem 6.10. The following three statements are equivalent for a collection monad $\mathbf{T} = (T, \eta, \mu)$.

(1) **T** is ordered ('members can be listed').
(2) Given $\tau \in TX$ and $\sigma \in X!$ ($X!$ is the set of bijections $X \longrightarrow X$), if $(T\sigma)\tau = \tau$, then $\sigma(x) = x$ for all $x \in mem\ \tau$ ('permuting members changes the collection').
(3) T preserves equalizers of monomorphisms.

Proof. (1) ⇒ (2) Let $\gamma : T \longrightarrow List$ be taut, $\sigma \in X!$, $\tau \in TX$ with $(T\sigma)\tau = \tau$. As σ is monic, we have $\gamma_X\tau = \gamma_X(T\sigma\tau) = \sigma^*\gamma_X\tau$. Here $\sigma^* = List(\sigma)$ maps $x_1 \dots x_n$ to $\sigma(x_1)\dots\sigma(x_n)$, so $\sigma(x) = x$ for all members of the list $\gamma_X\tau$. But $mem(\gamma_X\tau) = mem\ \tau$ by Proposition 4.12.
(2) ⇒ (3) Let $f, g : X \longrightarrow Y$ be monic and let $E = \{x \in X : fx = gx\}$ be the equalizer of f, g with inclusion $i : E \longrightarrow X$. To show that TE is the equalizer of Tf, Tg it is necessary to show that if $\tau \in TX$ with $(Tf)\tau = v = (Tg)\tau$, then $mem\ \tau \subset E$. By Lemma 4.16 $f(mem\ \tau) = mem\ v = g(mem\ \tau)$, so, by restriction, there are isomorphisms

$$mem\ \tau \xrightarrow{f} mem\ v \xrightarrow{g^{-1}} mem\ \tau.$$

As $g^{-1}f \in (mem\ \tau)!$ and $T(g^{-1}f)\tau = \tau$ as elements of $T(mem\ \tau)g^{-1}fx = x$ for all $x \in mem\ \tau$, that is, $fx = gx$ for all $x \in mem\ \tau$.
(3) ⇒ (1) Let $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ be a taut presentation. A *representation* of $\tau \in TX$ is (t, w, f), where t is a Σ-tree shape, $w \in V^*$ with $mem\ w = V_n$ and $f : V_n \longrightarrow X$ monic such that $(Tf)[t(w)] = \tau$. (Recall that V_n is the set of the first n standard variables and the $t(w)$ notation of Example 2.7; [–] refers to the θ-equivalence class of a Σ-tree). We know that each τ has representations by the proof that θ_X is surjective in Theorem 4.14. Also note that if (t, w, f) is a representation of τ, then, as θ is a natural transformation, $mem\ \tau = f(V_n)$.

Now let the set of all tree shapes be well ordered. V^* is well ordered by alphabetical ordering based on $v_1 < v_2 < v_3 \dots$. Define an ordering $\leqslant$ on the set of all representations

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

of $\tau \in TX$ by

$$(t, w, f) \leqslant (\bar{t}, \bar{w}, \bar{f}) \text{ if } t < \bar{t} \text{ or } t = \bar{t} \text{ and } w \leqslant \bar{w}.$$

We claim that this is a well ordering. This is obvious providing we show that if (t, w, f) and (t, w, g) represent τ, then $f = g$. Let E be the equalizer of f, g. As $(Tf)[t(w)] = \tau = (Tg)[t(w)]$ and TE is the equalizer of Tf, Tg by hypothesis, E is a support of τ. As θ is taut, the set V_n of members of w is a subset of E, that is, $f = g$.

We use this well ordering to define the desired taut transformation $\gamma : T \longrightarrow List$. Define $\gamma_X : TX \longrightarrow X^*$ by $\gamma_X \tau = f_\tau^* w_\tau$, where (t_τ, w_τ, f_τ) is the least representation of τ. If $P \subset X$ with inclusion $i : P \longrightarrow X$, the square

$$\begin{array}{ccc} TP & \xrightarrow{\gamma_P} & P^* \\ {\scriptstyle Ti}\downarrow & & \downarrow{\scriptstyle i^*} \\ TX & \xrightarrow[\gamma_X]{} & X^* \end{array}$$

commutes, because the set of representations of $\tau \in TP$ is essentially unchanged when τ is considered an element of TX (noting that $mem\, \tau \subset P$). To complete the proof we must show that the square

$$\begin{array}{ccc} TX & \xrightarrow{\gamma_X} & X^* \\ {\scriptstyle T\sigma}\downarrow & & \downarrow{\scriptstyle \sigma^*} \\ Tx & \xrightarrow[\gamma_X]{} & X^* \end{array}$$

commutes for every $\sigma \in X!$. For then γ is a mono-transformation and, since γ clearly preserves members, γ is taut.

Let $\tau \in TX$ have least representative (t, v, f) and let $(T\sigma)\tau$ have least representative (u, w, g). As $(t, v, \sigma f)$ represents $(T\sigma)\tau$, $(u, w, g) \leqslant (t, v, \sigma f)$. As $(u, v, \sigma^{-1} f)$ represents $(T\sigma^{-1})(T\sigma)\tau = \tau$, $(t, v, f) \leqslant (u, v, \sigma^{-1} f)$. It follows that $t = u$, $v = w$ and $g = \sigma f$. Thus $\sigma^*(\gamma_X \tau) = \sigma^* f^\star w = (\sigma f)^\star w = g^* w = \gamma_X((T\sigma)\tau)$. □

Example 6.11. RFList is ordered. The inclusion $RFList \longrightarrow List$, though not a natural transformation, is taut.

Example 6.12. OT$_R$ is ordered via $(\lambda_1, x_1) \ldots (\lambda_n, x_n) \mapsto x_1 \ldots x_n$. This is taut, though not a natural transformation.

Bag, Set and **Prob** are not ordered. This follows easily from the second condition in Theorem 6.10.

Our strategy to represent non-ordered collection classes will be to implement an 'ordered version' of them, which is not directly accessible to the class user. More precisely, we seek to represent a given collection monad with an *implementable cover*, which has C the class of implementable monads in the following definition.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Definition 6.13. If C is a class of monads, a *$\mathscr{C}$-cover* of a monad $\mathbf{T}$ is a taut monad quotient $\theta : \mathbf{T}_C \longrightarrow \mathbf{T}$ with $\mathbf{T}_C$ in $\mathscr{C}$ and with the property that given τ, $v \in TX$ with $\theta_X \tau = \theta_X v$ there exists $\sigma \in X!$ with $(T\sigma)\tau = v$. If $\mathscr{C}$ is all monads, we just say *'cover'*.

The meaning of a cover should be clear: to produce $\mathbf{T}$ from $\mathbf{T}_C$ add only equations that reorder the members.

Example 6.14. θ : **oBag** $\longrightarrow$ **Bag** defined by 'forgetting the order' is an implementable cover. If $\theta_X(x_1^{m_1} \dots x_k^{m_k}) = \theta_X(y_1^{n_1} \dots y_t^{n_t})$, then $k = t$ and the desired $\sigma \in X!$ is any such that $x_{\sigma i} = y_i$. Necessarily, $m_{\sigma i} = n_i$.

Example 6.15. What is an implementable cover for **Set?** The obvious *mem* : **List** $\longrightarrow$ **Set** that gives the set of elements in a list is a taut monad quotient. But *mem* is not a cover, because $mem(xx) = mem(x)$ whereas $\sigma^* : X^* \longrightarrow X^*$ is length preserving for every $\sigma \in X!$. **List** is the wrong monad. What is needed is an ordered version of sets that has no other refinements. The desired cover is the taut quotient **RFList** $\longrightarrow$ **Set** obtained by adding the balanced equation $ab = ba$. We leave it to the reader to find an implementation of **RFList**.

Example 6.16. Let R be the semiring of nonnegative reals. The subset of $oT_R X$ of all $(\lambda_1, x_1) \dots (\lambda_n, x_n)$ with $\sum \lambda_i = 1$ produces the submonad **oProb** of $\mathbf{oT}_R$ of *ordered probability distributions*. **oProb** is implementable by Proposition 6.5, and so provides an implementable cover for **Prob** in the obvious way. Finding a full implementation for **oProb** would involve discovering an equational presentation for it, which is not a trivial problem and is an extraneous problem as well, since the approach used corresponds to the usual viewpoint on probability distributions as linear combinations satisfying a constraint.

7. Shape and zipped monads

For lists, the zip of two lists of the same length is defined by $zip(x_1 \dots x_n, y_1 \dots y_n) = (x_1, y_1) \dots (x_n, y_n)$. In this section we study a more general class of monads for which collections having the same shape can be zipped. Our definition of shape is similar to that of Jay (1995, page 7).

Definition 7.1. Given a collection monad $\mathbf{T}$, the set $T1$ will be thought of as the set of $\mathbf{T}$-*shapes* and the function $T! : TX \longrightarrow T1$ will be called the *shape map of* X. (Here $1 = \{\bullet\}$ and $! : X \longrightarrow 1$ is the unique map). For intuition, think of $\mathbf{T}_\Sigma$ where the shape map substitutes $\bullet$ for all leaves of the tree, recovering our earlier definition of tree shape. The monad $\mathbf{T}$ is *shapeless* if there is only one shape, that is, $T1$ has only one element.

The next two propositions relate shapelessness to monad comprehensions.

Proposition 7.2. Let $\mathbf{T}$ be a monad with fundamental Γ as in Definition 3.8. Then $\mathbf{T}$ is shapeless if and only if $\Gamma_{11} : T1 \times T1 \longrightarrow T(1 \times 1)$ is injective. In that case, all Γ_{XY} are injective.

Proof. First suppose that Γ_{11} is injective so that, as $! : 1 \times 1 \longrightarrow 1$ is an isomorphism, $\psi = T!\Gamma_{11} : T1 \times T1 \longrightarrow T1$ is injective. Let $p, q \in T1$. As $\lambda x \cdot \psi(p, x)$ and $\lambda x \cdot \psi(x, q)$ are both the unique homomorphism $T1 \longrightarrow T1$, they are both id_{T1}. Thus $q = \psi(p, q) = p$ and $T1$ has at most one element. On the other hand, η_1 is a function, so $T1$ has exactly

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

one element. Conversely, we show $\Gamma_{XY} : TX \times TY \longrightarrow T(X \times Y)$ is injective if $T1$ has one element. Notice that if $\alpha : X \longrightarrow TY = X \xrightarrow{!} 1 \xrightarrow{\tau} TY$ is constant, then $\alpha^{\#}$ is also constant since its homomorphic extension is $\tau^{\#}T!$ and $T!$ is constant. Now consider

$$\varphi = TX \times Y \xrightarrow{\Gamma_0} T(X \times Y) \xrightarrow{Tpr_Y} TY,$$

where Γ_0 is the first stage of the construction of Γ_{XY} in Proposition 3.4. For fixed $y \in Y$, $\lambda\tau \cdot \varphi(\tau, y)$ is the homomorphic extension of the constant map $x \mapsto y$ and so is constantly y. Thus for fixed $\tau \in TX$, $\lambda v \cdot Tpr_Y \Gamma_{XY}(\tau, v) = (\eta_Y)^{\#} = id_{TY}$. In a similar vein, consider

$$\psi = TX \times Y \xrightarrow{\Gamma_0} T(X \times Y) \xrightarrow{Tpr_X} TY.$$

For fixed $y \in Y$, $\lambda\tau \cdot \psi(\tau, y)$ is the homomorphic extension of η_X and so is id_{TX}. Thus, for fixed $\tau \in TX$, $\lambda v \cdot Tpr_X \Gamma_{XY}(\tau, v)$ is the homomorphic extension of the map constantly τ and so is itself constantly τ. We have thus shown that

$$TX \times TY \xrightarrow{\Gamma_{XY}} T(X \times Y) \xrightarrow{[Tpr_X, Tpr_Y]} TX \times TY = id_{TX \times TY}$$

and thus that Γ_{XY} is injective. □

Proposition 7.3. Let $\mathbf{T}$ be a monad. Then $\mathbf{T}$ is shapeless if and only if the equation $\omega \wedge false = false$ holds in the Propositional logic of $\mathbf{T}$.

Proof. Let $f = B \xrightarrow{!} 1 \xrightarrow{false} B$, $\alpha = B \xrightarrow{f} B \xrightarrow{\eta_B} TB$. Then $\alpha^{\#} = Tf$ and $\alpha^{\#}(\omega) = \omega \wedge false$. If $\mathbf{T}$ is shapeless, then $Tf = (T\ false)(T!)$ factors through $T1 = 1$ and $\omega \wedge false$ is constant. As $false \wedge false = false$, $\omega \wedge false = false$. Conversely, let $\omega \wedge false =$ false. Then $\alpha^{\#}$ is constant. Define $h = 1 \xrightarrow{false} B \xrightarrow{\eta_B} TB$. The calculation

$$\begin{aligned} T\ false &= h^{\#} = (hid_1)^{\#} = (h\ !\ false)^{\#} \\ &= (\eta_B\ false!\ false)^{\#} = (\alpha\ false)^{\#} = (\alpha^{\#}\eta_B\ false)^{\#} \\ &= (\alpha^{\#}\eta_B\ false)^{\#} = (\alpha^{\#}h)^{\#} = \alpha^{\#}h^{\#} \end{aligned}$$

then shows that $T\ false$ is constant too. But $T\ false$ is also monic by Lemma 2.8 □

Proposition 7.4. (*cf.* Kock 1971) Every monad $\mathbf{T}$ has a largest shapeless submonad $\mathbf{T}_*$ and it is a cartesian submonad.

Proof. Define $i_X : T_*X \longrightarrow TX$ by the pullback

$$\begin{array}{ccc} T_*X & \xrightarrow{i_X} & TX \\ {\scriptstyle !}\downarrow & & \downarrow{\scriptstyle T!} \\ 1 & \xrightarrow[\eta_1]{} & T1 \end{array}$$

□

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Example 7.5. In the context of Example 6.16, **oProb** is the largest shapeless submonad of $\mathbf{oT}_R$. This provides a non-trivial example of an ordered shapeless monad. $\mathbf{RFList}_*$ is the submonad of non-empty lists.

We now turn toward the definition of collection monads with zip maps.

Definition 7.6. Let $\mathbf{T}$ be a collection monad. For sets X, Y the subset $sh_{XY} \subset TX \times TY$ of 'pairs of collections with the same shape' is defined by the pullback

$$\begin{array}{ccc} sh_{XY} & \xrightarrow{\pi_Y} & TY \\ {\scriptstyle \pi_Y}\downarrow & & \downarrow{\scriptstyle T!} \\ TX & \xrightarrow[T!]{} & 1 \end{array}$$

$\mathbf{T}$ is *zipped* if, for all X, Y, T preserves the pullback

$$\begin{array}{ccc} X \times Y & \xrightarrow{pr_Y} & Y \\ {\scriptstyle pr_X}\downarrow & & \downarrow{\scriptstyle !} \\ X & \xrightarrow[!]{} & 1 \end{array}$$

In that case there is a unique pullback-induced isomorphism

$$zip : sh_{XY} \longrightarrow T(X \times Y)$$

It is important to note that to recover the expected zip maps, we must use the 'standard' pullbacks. For example, sh_{XY} is $\{(\tau, v) \in TX \times TY : (T!)\tau = (T!)v\}$.

Example 7.7. $\mathbf{T}_\Sigma$ is zipped. If, for example, $\Sigma_3 = \{t\}$, $\Sigma_2 = \{m\}$, $\Sigma_0 = \{\Lambda\}$, then the shape of $t(x, m(y, z), \Lambda)$ is $t(\bullet, m(\bullet, \bullet), \Lambda)$. A second expression of the same shape is $t(a, m(b, c), \Lambda)$, and their zip is $t((x, a), m((y, b), (z, c)), \Lambda)$.

Example 7.8. List is zipped. Shape amounts to length and two lists of the same length are zipped in the usual way.

Example 7.9. For every positive semiring R, $\mathbf{oT}_R$ is not zipped. The shape of the R-list $(\lambda_1, x_1) \ldots (\lambda_n, x_n)$ is $\lambda_1 + \cdots + \lambda_n$ (identifying $T1$ with R). Consider $(1 + 1, x)y \in oT_R X$, $abc \in oT_R Y$. These have the same shape, namely $1 + 1 + 1$, but have ambiguous would-be zips $(x, a)(y, b)(x, c)$ and $(x, a)(x, b)(y, c)$ both of which project under Tpr_X to $(1 + 1, x)y$ and under Tpr_Y to abc.

We learned from Robin Cockett that any functor defined on a finitely complete category will preserve equalizers if it preserves pullbacks. This means there is a redundancy in the statement of the next proposition, which could also be stated, '$\mathbf{T}$ is zipped if and only if T preserves pullbacks'. We leave the statement as it is both for emphasis and because the proof must establish preservation of equalizers first.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Proposition 7.10. If **T** is a zipped monad, then T preserves equalizers and pullbacks.

Proof. We first show that T preserves equalizers. Let $i : E \longrightarrow X$ be the equalizer of $f, g : X \longrightarrow Y$. Equivalently, if $\Delta : X \longrightarrow X \times X$ is the diagonal map, the square

$$\begin{array}{ccc} E & \xrightarrow{fi=gi} & Y \\ {\scriptstyle i}\downarrow & & \downarrow{\scriptstyle \Delta} \\ X & \xrightarrow[[f,g]]{} & Y \times Y \end{array}$$

is an inverse image square. Thus (A) below is a pullback. To prove $Ti = eq(Tf, Tg)$, consider $k : Z \longrightarrow TX$ with $Tf\,k = Tg\,k$, and call the common

$$\begin{array}{ccc} TE & \xrightarrow{Th} & Y \\ {\scriptstyle Ti}\downarrow & (\mathrm{A}) & \downarrow{\scriptstyle T\Delta} \\ TX & \xrightarrow[T[f,g]]{} & T(Y \times Y) \end{array}$$

value t. We must find $\psi : Z \longrightarrow TE$ with $Ti\,\psi = k$, (uniqueness being clear since T preserves monomorphisms). Such ψ is induced by the pullback property because $T\Delta\, t = T[f,g]k$ as follows. It suffices to prove this equation holds followed by Tpr_1, $Tpr_2 : T(Y \times Y) \longrightarrow TY$, since these maps are jointly monic because **T** is zipped. Indeed, $Tpr_1 T\Delta Th = T(pr_1 \Delta h) = Th = T(fi) = T(pr_1[f,g]i) = Tpr_1 T[f,g]Ti$, and similarly for pr_2.

Finally, we show T preserves pullbacks. Let

$$\begin{array}{ccc} P & \xrightarrow{\bar{f}} & Y \\ {\scriptstyle \bar{g}}\downarrow & & \downarrow{\scriptstyle g} \\ X & \xrightarrow[f]{} & Y \end{array}$$

be a pullback. Since a morphism of form $Th : TA \longrightarrow TB$ preserves shape, $[T\bar{f}, T\bar{g}]$ factors through the inclusion $i : sh_{XY} \longrightarrow TX \times TY$. This gives the factorization

$$[T\bar{f}, T\bar{g}] = TP \xrightarrow{T[\bar{f},\bar{g}]} T(X \times Y) \xrightarrow{zip} sh_{XY} \xrightarrow{i} TX \times TY,$$

which is monic, so $T\bar{f}$, $T\bar{g}$ are jointly monic. This narrows the proof down to the existence property to show that $T\bar{f}$, $T\bar{g}$ is the pullback of Tf, Tg. Now, $[\bar{f}, \bar{g}] : P \longrightarrow X \times Y$ is the equalizer of $f\ pr_1, g\ pr_2 : X \times Y \longrightarrow Z$, so $T[\bar{f}, \bar{g}] = eq(Tf\ Tpr_1, Tg\ Tpr_2)$. Consider

$$TX \xleftarrow{\alpha} R \xrightarrow{\beta} TY$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

with $Tf\alpha = Tg\beta$. Since Tf, Tg preserve shape, $[\alpha, \beta]$ factors through sh_{XY} via the map γ shown below. Since $Tf\ Tpr_1\ zip\,\gamma = Tf\alpha = Tg\beta = Tg\ Tpr_2\ zip\,\gamma$

$$\begin{array}{ccc} R & \xrightarrow{\gamma} & sh_{XY} \\ {\scriptstyle\psi}\downarrow & & \downarrow{\scriptstyle zip} \\ TP & \xrightarrow[T[\bar{f},\bar{g}]]{} & T(X\times Y) \end{array}$$

there is a unique ψ completing the square above. We have $T\bar{g}\,\psi = T(pr_2[\bar{f},\bar{g}])\psi = Tpr_2\ zip\,\gamma = \alpha$ and $T\bar{f}\,\psi 1 = \beta$ similarly, so we are done. □

Corollary 7.11. A zipped monad is ordered.

Proof. The proof is immediate from Theorem 6.10. □

Example 7.12. $\mathbf{oT}_R$ is an example of an ordered collection monad that is not zipped (as was just seen in Example 7.9). Notice that such a monad can be commutative, since $\mathbf{oT}_R$ is a commutative monad when R is a commutative semiring.

Example 7.13. RFList is another example of an ordered collection monad that is not zipped. Define $f, g : \{a,b,c\} \longrightarrow \{a,b\}$ by $f(a) = a$, $f(b) = f(c) = b$, $g(a) = g(b) = a$, $g(c) = b$. Then if T is the functor part of **RFList**, $(Tf)(abc) = ab = (Tg)(abc)$ but $abc \notin T\{a,c\}$. This shows that T does not preserve the equalizer of f, g.

Proposition 7.14. Let $\mathbf{S}$, $\mathbf{T}$ be collection monads. If $\mathbf{T}$ is zipped and if there exists a cartesian natural transformation $S \longrightarrow T$, then $\mathbf{S}$ is zipped. In particular, zipped monads are closed under cartesian submonads.

Proof. The proof follows from elementary category theory. □

The next result establishes constraints for any equational presentation of a zipped monad.

Proposition 7.15. Let $\mathbf{T}$ be a zipped monad and let $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ be any taut presentation. For $\tau \in T_\Sigma X$ let $[\tau]$ denote $\theta_X\tau$. Then the following statements hold.

(1) There are no same-shape equations, that is, if t is a tree shape with n leaves ($n > 0$), then no non-trivial equation of form $t(x_1,\ldots,x_n) = t(y_1,\ldots,y_n)$ holds. Formally, if $[t(x_1,\ldots,x_n)] = [t(y_1,\ldots,y_n)]$, then $x_1 \ldots x_n = y_1 \ldots y_n$ in X^*.

(2) If t is a tree shape with n leaves ($n > 1$), then the equation $t(x^n) = x$ does not hold, that is, for all $x \in X$, $[t(x^n)] \neq x^*$ in TX.

Proof. For the first statement, let $f, g : V_n \longrightarrow X$ be defined by $f(v_i) = x_i$, $g(v_i) = y_i$. Let E be the equalizer of f, g. As $(Tf)[t(v_1,\ldots,v_n)] = [t(x_1,\ldots,x_n)] = [t(y_1,\ldots,y_n)] = (Tg)[t(y_1,\ldots,y_n)]$ and T preserves the equalizer, E is a support of $t(v_1,\ldots,v_n)$. Thus $V_n = mem(t(v_1,\ldots,v_n)) \subset E$ so $f = g$.

For the second statement, suppose that such an equation does hold. Write $u(x,y) = t(x,y^{n-1})$, let $X = \{a,b\}$, $Y = \{c,d\}$ and define τ, $v \in T_\Sigma(X \times Y)$ as follows:

$$\begin{aligned} \tau &= u((a,c),u((b,c),(b,d))) \\ v &= u(u((a,c),u((a,c),(a,d))),u((b,c),u((b,c),(b,d)))). \end{aligned}$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

As (a, d) is a member of v but not a member of τ and as θ is taut, $[\tau] \neq [v]$. But because of the equation, it is routinely checked that both τ, v are mapped by TPr_X to $u(a, b)$ and by TPr_Y to $u(c, u(c, d))$. This is the desired contradiction, since TPr_X, TPr_Y are jointly monic. □

oProb is a non-trivial example of an ordered shapeless collection monad. The following corollary slows that this cannot happen for zipped monads.

Corollary 7.16. The only zipped, shapeless monad is **Atom**.

Proof. Let **T** be zipped and shapeless and let $\mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ be a taut presentation. Suppose $\Sigma_0 \neq \varnothing$, $\omega \in \Sigma_0$. As $T1 = 1$, we must have the equation $\omega = x$, which is impossible since a collection monad must be non-trivial. If for some $n > 1$, $\Sigma_n \neq \varnothing$, $\omega \in \Sigma_n$, we have, as before, the equation $\omega(x^n) = x$, which contradicts the second statement of the previous theorem. If $\omega \in \Sigma_1$, then $\omega(x) = x$, so ω is redundant. Thus **T** can be presented using empty Σ and so is **Atom**. □

Example 7.17. There exists a non-trivial finitary monad $\mathbf{T} = (T, \eta, \mu)$ such that T is zipped but **T** is not a collection monad. $\mathbf{SE}_Q$ provides such an example. Given $(f, g) \in TX$, $(\bar{f}, \bar{g}) \in TY$ of the same shape (so that $g = \bar{g}$), their zip is $[f, \bar{f}] : Q \longrightarrow X \times Y$, $g : Q \longrightarrow Q$. The details are left to the reader.

8. Separating shape and data

A Σ-tree $\tau \in T_\Sigma X$ is specified by two independent pieces of information: its tree shape $t \in T_\Sigma 1$ and its list of data $w \in X^*$ (with the length of w the number of leaves of t). In the notation of Example 2.7, $\tau = t(w)$. Jay (1995) developed generalizations to pullback-preserving functors in categories with finite limits and list objects. Our study in this section fits into this program, specializing to collection monads in $\mathbb{S}$.

In Jay's terminology, a cartesian natural transformation between pullback-preserving functors is *shapely*. Thus a functor is shapely over *List* if it admits a cartesian natural transformation to *List*. The next definition is the natural generalization to monads.

Definition 8.1. A collection monad **T** is *shapely* if there exists a cartesian monad map $\gamma : \mathbf{T} \longrightarrow \mathbf{List}$.

Example 8.2. $\mathbf{T}_\Sigma$, **List**, $\mathbf{Timing}_M$ and $\mathbf{Exception}_E$ are shapely monads. The monad map $memList : \mathbf{T} \longrightarrow \mathbf{List}$ of Example 2.11 is cartesian. There are at least two cartesian monad maps from **List** to itself, the identity and list reversal. The projection $\gamma_X : M \times X \longrightarrow X^*$, $(m, x) \mapsto x$ is a cartesian monad map from $\mathbf{Timing}_M$ to **List**. A cartesian monad map $\mathbf{Exception}_E \longrightarrow \mathbf{List}$ is described by $x \mapsto x$, $e \mapsto \Lambda$.

Proposition 8.3. A deterministic collection monad is shapely. A commutative, shapely collection monad is deterministic.

Proof. Let **T** be a deterministic collection monad. As $\mathbf{Pfn} = \mathbf{Exception}_{\{\perp\}}$ is shapely, it suffices to construct a cartesian monad map $\mathbf{T} \longrightarrow \mathbf{Pfn}$. Now $mem : \mathbf{T} \longrightarrow \mathbf{Set}$ factors through **Pfn** because each $\tau \in TX$ has at most one member; this gives the monad map

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

$mem : \mathbf{T} \longrightarrow \mathbf{Pfn}$. We must show that the square

$$\begin{array}{ccc} TX & \xrightarrow{mem_X} & X+\{\perp\} \\ {\scriptstyle T!}\downarrow & & \downarrow{\scriptstyle !+id_\perp} \\ T1 & \xrightarrow[mem_1]{} & 1+\{\perp\} \end{array}$$

is a pullback. But this is clear since, by Lemma 4.15, $mem_1^{-1}(\perp) = T\varnothing$.

Now let $\gamma : \mathbf{T} \longrightarrow \mathbf{List}$ be a cartesian monad map with $\mathbf{T}$ a commutative collection monad. By Proposition 3.9, the following square commutes for both σ:

$$\begin{array}{ccc} TX\times TX & \xrightarrow{\Gamma_{XX}} & T(X\times X) \\ {\scriptstyle \gamma_X\times\gamma_X}\downarrow & & \downarrow{\scriptstyle \gamma_{X\times X}} \\ X^*\times X^* & \xrightarrow[\Gamma^\sigma_{XX}]{} & (X\times X)^* \end{array}$$

Thus, if $\gamma_X(\tau) = w \in X^*$, $\Gamma^\sigma_{XX}(w,w)$ is independent of σ. If $w = abc$, the two values of $\Gamma^\sigma_{XX}(w,w)$ are

$$(a,a)(a,b)(a,c)(b,a)(b,b)(b,c)(c,a)(c,b)(c,c)$$

and

$$(a,a)(b,a)(c,a)(a,b)(b,b)(c,b)(a,c)(b,c)(c,c).$$

If these are equal, $a = b = c$. In general, $\gamma_X(\tau)$ has the form x^n, $n \geqslant 0$. Since γ preserves members, each τ has at most one member. □

We now turn to characterizing shapely monads in terms of equational presentations.

Definition 8.4. Let $\mathbf{T}$ be a collection monad. An equational presentation $\mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ of $\mathbf{T}$ is *shapely* if the following two properties hold:

(1) Each $\tau \in TX$ has a unique list of members. More precisely, if $t, u \in T_\Sigma X$, $w, v \in X^*$ with $[t(w)] = \tau = [u(v)]$, then $u = v$. The unique list $u = v$ is denoted $memList_X(\tau)$.
(2) Each element of TX is determined by its shape and list of members. More precisely, if $\tau, v \in TX$ satisfy $(T!)\tau = (T!)v$ and $memList_X(\tau) = memList_X(v)$ then $\tau = v$.

Theorem 8.5. Let $\mathbf{T}$ be a collection monad. Then $\mathbf{T}$ is shapely if and only if $\mathbf{T}$ has a shapely presentation.

Proof. First suppose that $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ is a shapely presentation. Consider the diagram

$$\begin{array}{ccccc} T_\Sigma X & \xrightarrow{\theta_X} & TX & \xrightarrow{memList_X} & X^* \\ {\scriptstyle T_\Sigma f}\downarrow & (A) & {\scriptstyle Tf}\downarrow & (B) & \downarrow{\scriptstyle f^*} \\ T_\Sigma Y & \xrightarrow[\theta_Y]{} & TY & \xrightarrow[memList_Y]{} & Y^* \end{array}$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

The square (B) commutes by Definition 8.4(1) because $(Tf)[t(w)] = [t(f^\star(w)]$. To show (B) is a pullback, it suffices to let $Y = 1$. $T!$ and $memList_X$ are jointly monic by Definition 8.4(2). As the composition $T_\Sigma X \longrightarrow X^*$ in the top row is the original $memList_X$ for trees, the perimeter rectangle is a pullback. Now use the Pullback Lemma (Lemma 1.8(3)). And $memList : \mathbf{T} \longrightarrow \mathbf{T}_\Sigma$ is a monad map by Lemma 2.6.

For the converse, let $\gamma : \mathbf{T} \longrightarrow \mathbf{List}$ be a cartesian monad map. Recall that $V_n = \{v_1, \ldots v_n\}$ is the set of the first n standard variables. Define

$$\Sigma_n = \{\tau \in TV_n : \gamma_{V_n}(\tau) = v_1, \ldots v_n\}.$$

As γ is taut in particular, for $\tau \in \Sigma_n$ we must have $mem\ \tau = V_n$. Thus if $\hat{\theta} : \mathbf{T}_{\hat{\Sigma}} \longrightarrow \mathbf{T}$ is the canonical presentation of the proof of Theorem 4.14, $\Sigma_n \subset \hat{\Sigma}_n$. Noting that the inclusion $i : \mathbf{T}_\Sigma \longrightarrow T_{\hat{\Sigma}}$ is a cartesian monad map,

$$\theta = \mathbf{T}_\Sigma \xrightarrow{\ i\ } \mathbf{T}_{\hat{\Sigma}} \xrightarrow{\ \hat{\theta}\ } \mathbf{T}$$

is a taut monad map. We will show that θ is the desired presentation.

We begin by showing that θ is a presentation, that is, that θ_X is surjective. Let $\tau \in TX$ and write $\gamma_X(\tau) = x_1 \ldots x_n$. Define $f : V_n \longrightarrow X$ by $fv_i = x_i$. Consider the diagram

$$\begin{array}{ccccc}
T_\Sigma V_n & \xrightarrow{\theta_{V_n}} & TV_n & \xrightarrow{\gamma_{V_n}} & V_n^* \\
\downarrow{\scriptstyle T_\Sigma f} & & \downarrow{\scriptstyle Tf} & (\mathrm{C}) & \downarrow{\scriptstyle f^*} \\
T_\Sigma X & \xrightarrow[\theta_X]{} & TX & \xrightarrow[\gamma_X]{} & X^*
\end{array}$$

As γ is cartesian, (C) is a pullback, so there exists unique $v \in TV_n$ with $(Tf)v = \tau$, $\gamma_{V_n}(v) = v_1 \ldots v_n$. Thus $v \in \Sigma_n$ and $v(v_1, \ldots, v_n) \in T_\Sigma V_n$. Using the inductive definition of $\hat{\theta}$ in the proof of Theorem 4.14, we have

$$\begin{aligned}
\theta_{V_n}(v(v_1, \ldots, v_n)) &= (V_n \xrightarrow{< \theta_{V_n} \eta_{\Sigma, V_n}(v_i) >} TV_n)^\#(v) \\
&= (V_n \xrightarrow{\eta_{V_n}} TV_n)^\#(v) = v.
\end{aligned}$$

Thus

$$\begin{aligned}
\theta_X(v(v_1, \ldots, v_n)) &= \theta_X T_\Sigma f(v(v_1, \ldots, v_n)) = Tf\theta_{V_n}(v(v_1, \ldots, v_n)) \\
&= (Tf)v = \tau,
\end{aligned}$$

as desired

We next establish Axiom 8.4(1). Here is suffices to show that $\gamma_X \theta_X(v) = memList_X(v)$ for all $v \in T_\Sigma X$. Use induction. For variables qua trees, $\gamma_X \theta_X \eta_{\Sigma,X}(x) = x^\star$ because $\gamma\theta$ is a monad map. For the inductive step, let $\tau \in \Sigma_n$, $\tau_1, \ldots \tau_n \in T_\Sigma X$. Then

$$\begin{aligned}
\gamma_X \theta_X(\tau(\tau_1, \ldots \tau_n)) &= \gamma_X(V_n \xrightarrow{< \theta_{V_n} \tau_i >} TX)^\#(v))z \quad \text{(inductive definition of } \hat{\theta}) \\
&= (V_n \xrightarrow{< \gamma_X \theta_X \tau_i >} X^*)^\#(\gamma_{V_n}\tau) \quad (\gamma \text{ is a monad map})
\end{aligned}$$

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

$$
\begin{aligned}
&= (V_n \xrightarrow{< memList_X \tau_i >} X^*)^{\#}(\gamma_{V_n}\tau) \quad \text{(induction hypothesis)}\\
&= (V_n \xrightarrow{< memList_X \tau_i >} X^*)^{\#}(v_1 \dots v_n) \quad \text{(definition of } \Sigma_n)\\
&= (memList_X\, \tau_1) \dots (memList_X\, \tau_n)\\
&= memList_X(\tau(\tau_1, \dots \tau_n))
\end{aligned}
$$

Finally, we show Axiom 8.4(2). Let $\tau, v \in TX$ with $(T!)\tau = (T!)v$, $memList_X(\tau) = memList_X(v)$. As the square

$$
\begin{array}{ccc}
TX & \xrightarrow{\gamma_X} & X^* \\
{\scriptstyle T!}\downarrow & & \downarrow{\scriptstyle !} \\
T1 & \xrightarrow[\gamma_1]{} & 1
\end{array}
$$

is a pullback we need only show that $\gamma_X\tau = \gamma_X v$ to show that $\tau = v$. But $\gamma_X = memList_X$ by the proof just given. □

We now introduce another class of monads that separate shape and data. The notion turns out to be weaker than 'shapely'.

Definition 8.6. Let $\mathbf{T}$ be a collection monad. An equational presentation $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ of $\mathbf{T}$ is *shaped* if it is a cartesian natural transformation. Equivalently (by Lemma 1.9) the square

$$
\begin{array}{ccc}
T_\Sigma X & \xrightarrow{\theta_x} & TX \\
{\scriptstyle T_\Sigma !}\downarrow & & \downarrow{\scriptstyle T!} \\
T_\Sigma 1 & \xrightarrow[\theta_1]{} & T1
\end{array}
\tag{8.7}
$$

is a pullback for all X. This diagram says that given $\tau \in TX$ and a tree shape $t \in T_\Sigma 1$ with $[t] \in T1$ the shape of τ, there is exactly one way to arrange the members of τ into a list $w \in X^*$ so that $[t(w)] = \tau$. Say that $\mathbf{T}$ is *shaped* if it has a shaped presentation.

We note at once that we cannot expect every taut presentation of a shaped monad to be shaped. For example, if (Σ, E) is a taut presentation and if $\psi \in \Sigma_2$, we can add a unary φ and impose the balanced equation $\varphi(x) = \psi(x, x)$ (which makes φ redundant) to get a new taut presentation θ of the same monad. But $\theta_1(\varphi(\bullet)) = (T!)[\psi(x, y)]$ and there is no way to fill in $\varphi(\bullet)$ with x and y to recover $[\psi(y)]$ under θ_X because there is only one leaf.

Shaped presentations have full implementations. It is only necessary to choose a tree-shape representative for each $\mathbf{T}$-shape to get one. The formal statement is as follows.

Theorem 8.8. Let $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ be a shaped presentation. Then there exists $I : \mathbf{T} \longrightarrow \mathbf{T}_\Sigma$ such that $(\Sigma, I, \theta, T_\Sigma)$ is a full implementation with I a cartesian natural transformation. In fact, there is a bijective correspondence between such full cartesian implementations I and left inverses I_1 to θ_1 that satisfy $I_1\eta_1 = \eta_1$.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Proof. Let $I_1 : T1 \longrightarrow T_\Sigma 1$ with $\theta_1 I_1 = id_{T1}$ and $I_1 \eta_1 = \eta_1$ (at least one such I_1 exists because θ_1 is surjective). Using the pullback (B) shown below, there exists unique $I_X : TX \longrightarrow T_\Sigma X$ with $\theta_X I_X = id_{TX}$ and $T_X ! I_X = I_1 T !$ as shown.

$$\begin{array}{ccccc} TX & \xrightarrow{I_x} & T_\Sigma X & \xrightarrow{\theta_x} & TX \\ {\scriptstyle T!}\downarrow & (A) & {\scriptstyle T_\Sigma !}\downarrow & (B) & \downarrow{\scriptstyle T!} \\ T1 & \xrightarrow[I_1]{} & T_\Sigma 1 & \xrightarrow[\theta_1]{} & T1 \end{array}$$

It is clear that when $X = 1$ the new I_1 agrees with the old. Moreover, the left square defining I_X is also a pullback because the perimeter trivially is.

Next we check that I preserves η using the fact that θ_X, $T_\Sigma !$ are jointly monic. $\theta_X(I_X \eta_X) = \eta_X = \theta_X \eta_X$ (θ is a monad map) and $T_\Sigma !(I_X \eta_X) = I_1 T ! \eta_X = I_1 \eta_1 ! = \eta_1 !$ (axiom on I_1) $= T_\Sigma ! \eta_1$. To complete the verification that $(\mathbf{\Sigma}, I, \theta, \mathbf{T}_\Sigma)$ is a full cartesian implementation, it must be shown that $I : T \longrightarrow T_\Sigma$ is a natural transformation. To this end, let $f : X \longrightarrow Y$ and consult the diagram

$$\begin{array}{ccccc} TX & \xrightarrow{Tf} & TY & \xrightarrow{T!} & T1 \\ {\scriptstyle I_X}\downarrow & (C) & {\scriptstyle I_Y}\downarrow & (D) & \downarrow{\scriptstyle I_1} \\ T_\Sigma X & \xrightarrow{T_\Sigma f} & T_\Sigma Y & \xrightarrow{T_\Sigma !} & T_\Sigma 1 \\ {\scriptstyle \theta_X}\downarrow & (E) & {\scriptstyle \theta_Y}\downarrow & & \\ TX & \xrightarrow[Tf]{} & TY & & \end{array}$$

(D) and (C, D) are the defining pullbacks for I. (E) commutes because θ is natural, so (C) commutes since it does followed by $T_\Sigma !$ and θ_Y. But then (C) is a pullback.

Every cartesian implementation must arise this way since (A) is a pullback if I is cartesian. And, trivially, different I_1 give different I. This establishes the desired bijection. □

Our next task is to establish some facts about the relationship between shapely, shaped and zipped monads.

Proposition 8.9. Every shapely presentation is shaped; in particular, a shapely monad is shaped. Every shaped monad is zipped. If $\mathbf{T}$ is zipped, each natural transformation $\gamma : T \longrightarrow List$ satisfies

$$\gamma_{X \times Y}(zip(\tau, \nu)) = zip(\gamma_X \tau, \gamma_Y \nu)$$

Proof. In the proof of Theorem 8.5 it was seen that if $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ is a shapely presentation, there exists a cartesian monad map $\gamma : \mathbf{T} \longrightarrow \mathbf{List}$ such that $\gamma\theta = memList : \mathbf{T}_\Sigma \longrightarrow \mathbf{List}$. Since $\gamma\theta$ is cartesian, it follows from the Pullback Lemma 1.8(2) that θ is cartesian.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

For the second statement, if $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ is a shaped presentation, it follows from Theorem 8.8 that there exists a cartesian natural transformation $I : \mathbf{T} \longrightarrow \mathbf{T}_\Sigma$. Proposition 7.14 then gives that $\mathbf{T}$ is zipped.

For the last statement, if $\mathbf{T}$ is zipped and $\gamma : T \longrightarrow List$ is a natural transformation, we have the commutative diagram

$$\begin{array}{ccccc}
TX & \xrightarrow{Tpr_X} & T(X\times Y) & \xrightarrow{Tpr_Y} & TY \\
\downarrow{\scriptstyle \gamma_X} & & \downarrow{\scriptstyle \gamma_{X\times Y}} & & \downarrow{\scriptstyle \gamma_Y} \\
X^* & \xrightarrow[pr_X^*]{} & (X\times Y)^* & \xrightarrow[pr_Y^*]{} & Y^*
\end{array}$$

If τ, v have the same shape, $zip(\tau, v)$ is the unique $\omega \in T(X \times Y)$ with $(Tpr_X)\omega = \tau$, $(Tpr_Y)\omega = v$. From the diagram, $zip(\gamma_X\tau, \gamma_X v) = \gamma_{X\times Y}(zip(\tau, v))$. □

It is not clear that if $\mathbf{T}$ is zipped, there must exist a natural transformation $T \longrightarrow List$. If $\mathbf{T}$ is shaped, however, there even exists a cartesian natural transformation $T \longrightarrow List$, namely any cartesian implementation $I : \mathbf{T} \longrightarrow \mathbf{T}_\Sigma$ followed by $memList$.

Proposition 8.10. Let $\mathbf{T} = (T, \eta, \mu)$ be a shaped monad. Then η, μ are cartesian. Moreover, for every shaped presentation $\theta : \mathbf{T}_\Sigma \longrightarrow \mathbf{T}$ of $\mathbf{T}$, if $\theta_X(\tau) = \theta_X(v)$, then τ, v have the same number of leaves.

Proof. That η is cartesian is true even for zipped monads by Lemma 1.9 and Proposition 7.15(2). To see that μ is cartesian, consider the diagrams

$$\begin{array}{ccccc}
T_\Sigma T_\Sigma X & \xrightarrow{\mu_X} & T_\Sigma X & \xrightarrow{\theta_X} & TX \\
\downarrow{\scriptstyle T_\Sigma T_\Sigma !} & (A) & \downarrow{\scriptstyle T_\Sigma !} & (B) & \downarrow{\scriptstyle T!} \\
T_\Sigma T_\Sigma 1 & \xrightarrow[\mu_1]{} & T_\Sigma 1 & \xrightarrow[\theta_1]{} & T1
\end{array}$$

$$\begin{array}{ccccccc}
T_\Sigma T_\Sigma X & \xrightarrow{T_\Sigma\theta_X} & T_\Sigma TX & \xrightarrow{\theta_{TX}} & TTX & \xrightarrow{\mu_X} & TX \\
\downarrow{\scriptstyle T_\Sigma T_\Sigma !} & (C) & \downarrow{\scriptstyle T_\Sigma T!} & (D) & \downarrow{\scriptstyle TT!} & (E) & \downarrow{\scriptstyle T!} \\
T_\Sigma T_\Sigma 1 & \xrightarrow[T_\Sigma\theta_1]{} & T_\Sigma T1 & \xrightarrow[\theta_{T1}]{} & TT1 & \xrightarrow[\mu_1]{} & T1
\end{array}$$

The perimeter rectangles (A, B) and (C, D, E) agree since θ is a monad map. That (A) is a pullback is routinely verified. Then, as (B) is a pullback, so is (C, D, E). As $\mathbf{T}_\Sigma$ is zipped, (C), (D) and thus (C, D) are pullbacks. Every functor preserves split epimorphisms, so θ_{T1} and $T_\Sigma\theta_1$ are split epic. By Lemma 1.8(3), (E) is a pullback, as desired.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

For the last statement it is enough to show that if $s, t \in T_\Sigma 1$ with $[s(\bullet^m)] = [t(\bullet^n)]$, then $m = n$. Consider the pullback (B) where X has two elements. For each $y_1 \dots y_n \in X^*$ there exists unique $x_1 \dots x_m \in X^*$ with $[s(x_1 \dots x_m)] = [t(y_1 \dots y_n)]$. As $2^m = 2^n$, $m = n$. □

Example 8.11. There exists a shaped monad that is not shapely. By Manes (1982, 5.7), if $\Sigma_2 = \{+, \oplus\}$, then $\mathbf{T}_{\Sigma,E}$ is a commutative monad if and only if the *middle-four interchange* law

$$(a + b) \oplus (c + d) = (a \oplus c) + (b \oplus d)$$

can be derived from E. Let $\mathbf{T} = \mathbf{T}_{\Sigma,E}$, where E is precisely the middle-four interchange. Since $[x \oplus y]$ has two members, $\mathbf{T}$ is not deterministic, so, by Proposition 8.3, $\mathbf{T}$ is not shapely. Proving that (Σ, E) is a shaped presentation is routine

We have already seen that 'collection' and 'ordered' are closed under arbitrary submonads, and it is natural to ask if this holds for 'zipped', 'shaped' or 'shapely'. In fact this is not so in all three cases.

Example 8.12. List has a submonad that is not zipped. Let $TX = \{w \in X^* : w \in X$ or w has form $x\ u\ x\ v$ with $x \in X, u, v \in X^*\}$. To see that T is a submonad, $x \in TX$ for $x \in X$ by definition. Now let $\alpha : X \longrightarrow TY$. If $\alpha(x) = y$, then $\alpha^\#(x\ u\ x\ v) = y\ \alpha^\#(u) y\ \alpha^\#(u) \in TY$. If $\alpha(x) = y\ p\ y\ q$, then $\alpha^\#(x\ u\ x\ v) = y\ p\ y\ q \alpha^\#(u) y\ p\ y\ q \alpha^\#(u) \in TY$. The only possible zip of *aba* with *xxy* is $(a, x)(b, x)(a, y)$, but this is not in $T(\{a, b\} \times \{x, y\})$.

Example 8.13. Lists of odd length forms a cartesian submonad of **List** so this monad is shapely. Palindromes of odd length forms a submonad of **List** that is not cartesian but this monad is shapely anyway via the cartesian monad map $waw \mapsto wa$.

9. Conclusions

This paper has identified theoretical issues that would be used by a program designer creating a library of collection classes based on monads. The following steps underly class design for a particular monad $\mathbf{T}$.

Step 1. Verify theoretically that $\mathbf{T}$ satisfies the collection monad axioms.

Step 2. Find a cover $\bar{\theta} : \bar{\mathbf{T}} \longrightarrow \mathbf{T}$ and an implementation $(\Sigma, I, \theta, \hat{\mathbf{T}})$ of $\bar{\mathbf{T}}$. Functional programming operations such as *map* and *outer* are inherited by $\bar{\mathbf{T}}$ from the base class *collection* in terms of the virtual methods I, θ that code and decode the desired operations in terms of tree-processing with canonical-form trees. The end user sees $\mathbf{T}$ as $\bar{\mathbf{T}}$ but cannot access $\bar{\mathbf{T}}$-specific structure.

Step 3. Provide semantics for virtual methods such as *test* and *reduce* so that other functional programming operations such as *select, join* and *append* can be inherited from the base class.

Step 4. Determine theoretically if $\mathbf{T}$ has other special structure such as 'zipped', 'shaped' or 'shapely'. If $\mathbf{T}$ is zipped, the zip map is not inherited, since two collections might have the same $\mathbf{T}$-shape without having a common tree shape to zip. If $\mathbf{T}$ is shaped or shapely, its implementation should be a shaped, respectively, shapely presentation.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press

Further aspects of collection classes await investigation. Polymorphism, including the Beck distributive laws mentioned at the end of Section 2, is one topic. Another concerns infinite collections (as would arise in lazy functional programming). A rich supply of candidate monads, here, are iteration theories (Bloom and Ésik 1993).

Among the issues considered in this paper, major open questions remain. Is every zipped monad shaped? For zipped monads, is a taut presentation given by balanced equations in which both sides have the same number of leaves necessarily a shaped presentation? Is a collection monad with η and μ cartesian necessarily shaped?

References

Barr, M. and Wells, C. (1985) *Toposes, Triples, and Theories*, Springer-Verlag.

Beck, J. (1967) Distributive laws. *Springer-Verlag Lecture Notes in Mathematics* **80** 119–140.

Birkhoff, G. (1935) On the structure of abstract algebras. *Proc. Camb. Phil. Soc.* **31** 433–454.

Bloom, S. and Ésik, Z. (1993) *Iteration Theories, the Equational Logic of Iterative Processes*, Springer-Verlag.

Buneman, P. Naqvi, S. Tannen, V. and Wong, L. (1995) Principles of programming with complex objects and collection types. *Theoretical Computer Science* **149** 3–48.

Eilenberg, S. and Moore, J. C. (1965) Adjoint functors and triples. *Illinois J. Math.* **9** 381–398.

Eilenberg, S. (1974) *Automata, Languages, and Machines, Vol. A*, Academic Press.

Filinski, A. (1994) *Representing monads*, Proceedings of POPL '94 Symposium on Principles of Programming Languages, ACM Press 446–457.

Godement, R. (1958) *Theorie des Faisceux*, Hermann.

Goodman, I. R. and Nguyen, H. T. (1985) *Uncertainty Models for Knowledge-Based Systems*, North-Holland.

Barry Jay, C. (1995) A semantics for shape. *Science of Computer Programming* **25** 251–283.

Kleisli, H. (1965) Every standard construction is induced by a pair of adjoint functors. *Proc. Amer. Math. Soc.* **16** 544–546.

Kock, A. (1970) Monads on symmetric monoidal closed categories. *Arch. Math.* **21**.

Kock, A. (1971) Bilinearity of cartesian closed monads. *Math. Scand.* **29** 161–174.

Lawvere, F. W. (1963) *Functorial Semantics of Algebraic Theories*, dissertation, Columbia University.

Mac Lane, S. (1971) *Categories for the Working Mathematician*, Springer-Verlag.

Manes, E. G. (1976) *Algebraic Theories*, Springer-Verlag.

Manes, E. G. (1982) A class of fuzzy theories. *J. Math. Anal. Appl.* **85** 409–451.

Manes, E. G. (1992) *Predicate Transformer Semantics*, Cambridge University Press.

Moggi E. (1991) Notions of computation and monads. *Information and Computation* **93** 55–92.

Moggi, E. (1995) A semantics for evaluation logic. *Fund Informaticae* **22** 117–152.

Peyton Jones, S. L. (ed.) (1987) *The Implementation of Functional Programming Languages*, Prentice-Hall.

Steele, G. L. (1994) *Building interpreters by composing monads*, Proceedings of POPL '94 Symposium on Principles of Programming Languages, ACM Press, 472–492.

Taylor, W. (1979) Equational Logic. *Houston J. Math.* **5**.

Trinder, P. (1991) Comprehensions, a query notation for DBPLs. In: Kanellakis P. and Schmidt, J. W. (eds) *Database Programming Languages: Bulk Types and Persistent Data, Third International Workshop, Aug. 27–30, 1991, Nafplion. Greece*, Morgan-Kaufmann Publishers, San Mateo, CA, USA 55–68.

Wadler, P. (1992) Comprehending monads. *Math. Struct. in Comp. Sci.* **2** 461–493.

https://doi.org/10.1017/S0960129598002515 Published online by Cambridge University Press