Monads

Luís S. Barbosa

Dep. Informática
Universidade do Minho (Portugal)
2003.12.09
lsb@di.uminho.pt

1. Monads. The use of monads to structure the denotational semantics of programming languages, has been proposed by Moggi [Mog89,Mog91]. Later the concept was introduced by Wadler [Wad92,Wad95] in programming practice, as rigorous style of combining purely functional programs that mimic impure (side-)effects.

The key idea is that monads permit to encode in abstract terms several kinds of computational effects, such as exceptions, state updating, non determinism or continuations. Such effects are represented by a type constructor (*i.e.*, an endofunctor in a suitable category) B so that computations producing values of type O are regarded as terms of type $\mathsf{B}O$. In this way, *values* and *computations* are explicitly distinguished and programs can be thought of as arrows $I \to \mathsf{B}O$ representing the computation of values of type O from values of type I, while producing some effect described by B. Or, putting it in a different way, output values arise encapsulated (or embedded) in the effect specified by B. Such arrows will be referred to in the sequel as B-computations or *monadic arrows*. Furthermore, B obeys some equational laws on the interaction of values and computations. Formally the notion of a monad is a category theoretical one, defined as follows:

2. Definition. A *monad* in a category C consists of an endofunctor $\mathsf{B} : \mathsf{C} \longrightarrow \mathsf{C}$ and a pair of natural transformations $\mu : \mathsf{B}^2 \Longrightarrow \mathsf{B}$ and $\eta : \mathsf{Id}_\mathsf{C} \Longrightarrow \mathsf{B}$ understood as an associative 'multiplication' and its 'unit', such that the diagrams below commute:

$$\begin{tikzcd}
\mathsf{B}^3 \arrow[r, "\mathsf{B}\mu"] \arrow[d, "\mu\mathsf{B}"'] & \mathsf{B}^2 \arrow[d, "\mu"] \\
\mathsf{B}^2 \arrow[r, "\mu"'] & \mathsf{B}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{Id}_\mathsf{C}\mathsf{B} \arrow[r, "\eta\mathsf{B}"] \arrow[dr, equal] & \mathsf{B}^2 \arrow[d, "\mu"] & \mathsf{B}\mathsf{Id}_\mathsf{C} \arrow[l, "\mathsf{B}\eta"'] \arrow[dl, equal] \\
& \mathsf{B} &
\end{tikzcd}$$

that is,

$$\mu \cdot \eta\mathsf{B} \;= \mu \cdot \mathsf{B}\eta \;=\; \mathsf{id} \tag{1}$$

$$\mu \cdot \mathsf{B}\mu \;= \mu \cdot \mu\mathsf{B} \tag{2}$$

In fact, a monad can be thought of as a monoid in C^C, the category of C-endofunctors. Notice that $\mathsf{B}^2 = \mathsf{B}\mathsf{B}$, and, thus, functor composition, rather than functor product, is used in this definition. Thinking of B as the encapsulation of a computational structure, its unit η represents the minimal such structure when a value $o \in O$ is embedded in $\mathsf{B}O$. Multiplication, on the other hand, flattens computations, providing a way to view a B-effect of a B-effect still as a B-effect.

3. Partiality and Nondeterminism. Two common computational effects are *partiality* and *nondeterminism*. The former models computations that either succeed, returning a value of the given type, or fail raising a simple exception of type $\mathbf{1}$. This is captured by the simplest version of the usual exception monad $\mathsf{B} = \mathsf{Id} + E$, instantiating E with $\mathbf{1}$. In functional programming it is known as the *maybe* monad. *Nondeterminism*, on the other hand, is introduced by the (finite) powerset monad $\mathsf{B} = \mathcal{P}$. Unit and multiplication are defined, in the first case, by $\eta^{-+\mathbf{1}} = \iota_1$ and $\mu^{-+\mathbf{1}} = [\mathsf{id}, \iota_1]$. For the powerset monad, one gets $\eta^{\mathcal{P}-} = \mathsf{sing} = \lambda\, x\,.\ \{x\}$ and $\mu^{\mathcal{P}-} = \bigcup$. A variant to the powerset monad is the *sequence* monad based on $\mathsf{B} = \mathsf{Id}^*$, having as unit the singleton list constructor and, as multiplication, the distributed concatenation of sequences. This entails an 'ordered' view of nondeterminism. A possible interpretation is 'all outputs are possible, but their probability decreases (or increases) along the sequence'.

4. Exponential. The *exponential* monad is based on the functor $\mathsf{B} = \mathsf{Id}^S$, with unit and multiplication given by

$$\eta_X = X \xrightarrow{\mathsf{sp}} (X \times S)^S \xrightarrow{\pi_1{}^S} X^S$$

$$\mu_X = X^{S^S} \xrightarrow{\mathsf{sp}} (X^{S^S} \times S)^S \xrightarrow{\langle \mathsf{ev}, \pi_2 \rangle^S} (X^S \times S)^S \xrightarrow{\mathsf{ev}^S} X^S$$

Going pointwise, the definitions above read:

$$\eta_X\ x = \lambda s\,.\ x \quad (\textit{i.e.},\quad \underline{x} \cdot !_S\ )$$

and

$$\mu_X\ m = \lambda s\,.\ (m\ s)\ s$$

In the context of functional programming this is also known as the *state reader* monad used to encode computations with reading access to a state variable (of type S). When this kind of monadic functions are composed (see §7), the same state is passed as argument to all of them. In fact, a popular use of monads in functional programming is to encapsulate context information, or 'state'. A more elaborated monad coping with state information is the *state transformer*, based on the functor $\mathsf{B} = (\mathsf{Id} \times S)^S$. In this case $\mu = \mathsf{ev}^S$ and η is simply sp.

5. Monoids. In a cartesian category, any monoid $\langle M, \theta, e \rangle$ determines a monad based on the functor $\mathsf{B} = M \times \mathsf{Id}$. Unit and multiplication arise as $\eta = \langle \underline{e} \cdot !, \mathsf{id} \rangle$ and $\mu = (\theta \times \mathsf{id}) \cdot \mathsf{a}^\circ$.

6. Adjunctions. We have seen some common examples of monads in the previous paragraphs. The notion of a monad is, however, extremely general. A basic observation to justify this claim is that any adjunction $\mathsf{T} \dashv \mathsf{S}$ gives rise to a monad based on the composite functor ST. Its unit is simply the unit of the adjunction, and the multiplication is given by $\mathsf{S}\epsilon\mathsf{T}$.

For example, the adjunction between product and exponential originates the state transformer monad mentioned above, which is described in detail in [Wad92]. The typical case of adjunctions between free and forgetful functors in categories of algebraic structures, gives rise to the *identity* monad. And so on. It is also the case that every monad determines an adjunction.

A related topic is the notion of an *algebra* for a monad. Basically this is defined as an algebra for the monad functor which, additionally, verifies some extra conditions which resemble the definition of *action* for a monoid. We will not pursue this here, but notice that the two categories of algebras that may be associated to a monad B — known as its *Kleisli* and *Eilenberg-Moore* categories — represent, respectively, initial and final solutions to the problem of finding adjunctions that generate B. It should also be referred, to stress the generality of the concept, that any algebra, in the usual sense of the word in Universal Algebra — *i.e.*, every set of operations obeying equational laws — can be regarded as an (algebra for a) monad.

The reader is referred to [Mac71] or [BW85] for standard results on monads and their relevance in algebra. Our concern, in this appendix, is more simply to present the technicalities needed for using monads in the thesis. We will 'visit' Kleisli categories, however, as, operationally, they can be regarded as 'spaces of computations'.

7. Kleisli Composition. Monadic arrows, described above, can be composed to build more complex computations from simpler ones. In fact, given $f : I \longrightarrow \mathsf{B}J$ and $g : J \longrightarrow \mathsf{B}O$, their composition $g \bullet f : I \longrightarrow \mathsf{B}O$ is defined by

$$g \bullet f = I \xrightarrow{f} \mathsf{B}J \xrightarrow{\mathsf{B}g} \mathsf{BB}O \xrightarrow{\mu} \mathsf{B}O$$

This composition law is associative, with η acting as the identity. Therefore, for each monad B, a new category is defined over the objects of C but with B-computations as arrows. This is known as the Kleisli category for B on C.

8. Kleisli Triples. The composite $\mu \cdot \mathsf{B}\ g$ is sometimes written as $g^* : \mathsf{B}\ J \longrightarrow \mathsf{B}\ O$ and called the *extension* of g. This gives rise to an alternative definition of a monad, known as a *Kleisli triple*. Formally, a *Kleisli triple* consists of an object mapping $\mathsf{B} : \mathrm{obj}(\mathsf{C}) \longrightarrow \mathrm{obj}(\mathsf{C})$, a $\mathrm{obj}(\mathsf{C})$-indexed mapping $\eta_I : I \longrightarrow \mathsf{B}\ I$ natural in I, and an operator $\_^*$ which maps every $f : I \longrightarrow \mathsf{B}\ O$ into $f^* : \mathsf{B}\ I \longrightarrow \mathsf{B}\ O$, satisfying the following axioms:

$$
\begin{aligned}
{\eta_I}^* &= \mathsf{id}_{\mathsf{B}\ I} &\quad (3)\\
f^* \cdot \eta_I &= f &\quad (4)\\
g^* \cdot f^* &= (g^* \cdot f)^* &\quad (5)
\end{aligned}
$$

It is straightforward to show the equivalence between monads and Kleisli triples.

Proof. Given a monad $\langle \mathsf{B}, \eta, \mu \rangle$ a Kleisli triple is obtained by restricting B to objects and defining $f^* = \mu \cdot \mathsf{B}\ f$. For the other direction, suppose given a triple $\langle \mathsf{B}, \eta, \_^* \rangle$. To obtain a monad, first extend B to an endofunctor by defining its action on morphisms as $\mathsf{B}\ (f : I \longrightarrow O) = (\eta_O \cdot f)^*$. Then, define μ_I as $\mathsf{id}^*_{\mathsf{B}\ I}$. The reader can easily verify that, in both cases, the axioms just stated and the ones in §2 are respected.

□

9. Strong Monads. A *strong monad* is simply a monad $\langle \mathsf{B}, \eta, \mu \rangle$ where B is a strong functor and both η and μ strong natural transformations [Koc72] (see §11 below). Therefore, the characterisation law for strong natural transformations (18), entails the following additional axioms:

$$
\begin{aligned}
\tau_r^{\mathsf{B}} \cdot (\eta \times \mathsf{id}) &= \eta &\quad (6)\\
\tau_r^{\mathsf{B}} \cdot (\mu \times \mathsf{id}) &= \mu \cdot \mathsf{B}\ \tau_r^{\mathsf{B}} \cdot \tau_r^{\mathsf{B}} &\quad (7)
\end{aligned}
$$

which express the commutativity of the following diagrams

$$
\begin{tikzcd}
\_ \times \_ \arrow[d, "\eta\times\mathsf{id}"'] \arrow[dr, "\eta"] & \\
\mathsf{B} \times \_ \arrow[r, "\tau_r^{\mathsf{B}}"'] & \mathsf{B}(\_ \times \_)
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{B} \times \_ \arrow[rr, "\tau_r^{\mathsf{B}}"] & & \mathsf{B}(\_ \times \_) \\
\mathsf{B}^2 \times \_ \arrow[u, "\mu\times\mathsf{id}"] \arrow[rr, "\tau_r^{\mathsf{B}^2}"] \arrow[dr, "\tau_r^{\mathsf{B}}"'] & & \mathsf{B}^2(\_ \times \_) \arrow[u, "\mu"'] \\
& \mathsf{B}(\mathsf{B} \times \_) \arrow[ur, "\mathsf{B}\ \tau_r^{\mathsf{B}}"'] &
\end{tikzcd}
$$

Both laws result from a direct application of (18) followed by an unfolding of the strength definition. In particular, recall that τ_r^{Id} is the identity and $\tau_r^{\mathsf{B}^2} = \mathsf{B}\ \tau_r^{\mathsf{B}} \cdot \tau_r^{\mathsf{B}}$.

10. Distributing over products. Applying τ_r and then τ_l, or the other way round, on a product $\mathsf{B}I \times \mathsf{B}J$ yields a result of type $\mathsf{BB}(I \times J)$, which can, then, be flattened, with μ, to $\mathsf{B}(I \times J)$. In most cases, however, the *order* of application is relevant for the outcome.

Formally, we may say that the Kleisli composition of the right with the left strength, gives rise to a natural transformation $\delta_r : \mathsf{B} \times \mathsf{B} \Longrightarrow \mathsf{B}(\mathsf{Id} \times \mathsf{Id})$ whose component on objects I and J is given by

$$
\delta r_{I,J} \;=\; \tau_{r_{I,J}} \bullet \tau_{l_{\mathsf{B}I,J}} \qquad (8)
$$

Dually, one may define $\delta l_{I,J} = \tau_{l_{I,J}} \bullet \tau_{r_{I,\mathsf{B}J}}$. Such transformations specify how the monad distributes over the cartesian product and, therefore, represent a sort of sequential composition of B-computations. They relate to strength by

$$\delta r_{I,J} \cdot (\mathsf{id}_{\mathsf{B}I} \times \eta_J) = \tau_{r_{I,J}} \tag{9}$$

$$\delta l_{I,J} \cdot (\eta_I \times \mathsf{id}_{\mathsf{B}J}) = \tau_{l_{I,J}} \tag{10}$$

Whenever δ_r and δ_l coincide, the monad is said to be *commutative* — which is indeed the case of the partiality, powerset and exponential monads (§§3 and 4), but not of the sequence one. It is also the case that a monad generated by a monoid (§5) is commutative if the base monoid is Abelian. In general, however, δ_r and δ_l are related by the following law

$$\delta_r \cdot \mathsf{s} = \mathsf{B}\,\mathsf{s} \cdot \delta_l \tag{11}$$

Proof.

$$
\begin{array}{cl}
 & \delta_r \cdot \mathsf{s} \\
= & \quad \{\ \delta_r \text{ definition } \} \\
 & \mu \cdot \mathsf{B}\,\tau_r \cdot \tau_l \cdot \mathsf{s} \\
= & \quad \{\ \tau_r,\ \tau_l \text{ interaction } \} \\
 & \mu \cdot \mathsf{B}\,(\mathsf{B}\,\mathsf{s} \cdot \tau_l \cdot \mathsf{s})\tau_l \cdot \mathsf{s} \\
= & \quad \{\ \mathsf{B} \text{ functor } \} \\
 & \mu \cdot \mathsf{B}\,\mathsf{B}\,\mathsf{s} \cdot \mathsf{B}\,\tau_l \cdot \mathsf{B}\,\mathsf{s} \cdot \tau_l \cdot \mathsf{s} \\
= & \quad \{\ \tau_r,\ \tau_l \text{ interaction } \} \\
 & \mu \cdot \mathsf{B}\,\mathsf{B}\,\mathsf{s} \cdot \mathsf{B}\,\tau_l \cdot \tau_r \\
= & \quad \{\ \mu \text{ natural } \} \\
 & \mathsf{B}\,\mathsf{s} \cdot \mu \cdot \mathsf{B}\,\tau_l \cdot \tau_r \\
= & \quad \{\ \delta_l \text{ definition } \} \\
 & \mathsf{B}\,\mathsf{s} \cdot \delta_l
\end{array}
$$

□

11. STRONG FUNCTORS. A functor T is strong if it possess a (tensorial) *strength*, *i.e.*, a natural transformation

$$\tau_r^{\mathsf{T}} : \mathsf{T} \times - \Longrightarrow \mathsf{T}(\mathsf{Id} \times -)$$

subject to certain conditions. Its effect is to *distribute* context "$-$" along functor T (see §14 for a set of examples). If types are modeled in such a setting, universal combinators (for example, cata and anamorphisms) will possess a somewhat more general shape, able to deal with the presence of extra parameters in the functions being defined. And this is possible even in case the underlying category is not cartesian closed (and therefore *currying* is not available).

There is, however, another reason supporting the strongness requirement, and probably a more fundamental one when building categorical models for, *e.g.*, functional programming. In fact, a basic requirement in such models is the need for arrows to *internalise*, for every functor capturing a type signature, its action on morphisms. Such a collection of arrows will, therefore, give semantics to a generalised *map* functional.

The task of mimicking the action of a functor T on morphisms within the (semantic) category, is accomplished by a natural transformation σ^{T}, called the *functorial strength*. It turns out that both notions of strength are the two sides of the same coin in the sense that, for the same functor, one uniquely determines the other.

The basic results on strength can be found on [Koc72]. A standard reference on strong data types is [CS92]. A subsequent paper [CS95] describes a term logic for programming with them, which is implemented in the experimental language CHARITY [CF92].

12. FUNCTORIAL STRENGTH. As mentioned above, a *functorial strength* σ^{T} is a natural transformation which internalises the action of a functor T on morphisms. Its component, at arbitrary objects X, Y, is a map

$$\sigma^{\mathsf{T}}_{X,Y} : \mathsf{C}[X,Y] \longrightarrow \mathsf{C}[\mathsf{T}\ X, \mathsf{T}\ Y]$$

satisfying

$$\begin{tikzcd}
\mathbf{1} \arrow[r, "\underline{\mathsf{id}_{\mathsf{T}\ X}}"] \arrow[d, "\underline{\mathsf{id}_X}"'] & \mathsf{C}[\mathsf{T}\ X, \mathsf{T}\ X] \\
\mathsf{C}[X,Y] \arrow[ur, "\sigma^{\mathsf{T}}_{X,X}"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{C}[Y,Z] \times \mathsf{C}[X,Y] \arrow[r, "\cdot"] \arrow[d, "\sigma^{\mathsf{T}}_{Y,Z} \times \sigma^{\mathsf{T}}_{X,Y}"'] & \mathsf{C}[X,Z] \arrow[d, "\sigma^{\mathsf{T}}_{X,Z}"] \\
\mathsf{C}[\mathsf{T}\ Y, \mathsf{T}\ Z] \times \mathsf{C}[\mathsf{T}\ X, \mathsf{T}\ Y] \arrow[r, "\cdot"] & \mathsf{C}[\mathsf{T}\ X, \mathsf{T}\ Z]
\end{tikzcd}$$

Of course, if C is Cartesian closed, homsets have internal representations as exponentials and, consequently, functorial strength assumes the more familiar form $\sigma^{\mathsf{T}}_{X,Y} : X^Y \longrightarrow \mathsf{T}\ X^{\mathsf{T}\ Y}$. The subject is far more elaborate than we can glimpse here. It arises in the context of *enriched category theory* [Kel82], a branch of research lead by the idea of *internalising*: 'meta' notions normally lying on top of a category, become (internally) represented by (families of) morphisms in the category itself. In a Cartesian closed category, a functor with functorial strength is said to be *enriched over the exponential*, which is thus another common designation for a strong functor. Notice that the axioms for functorial strength captured by the two diagrams above are no more than an internal representation of the properties of a functor: identities are preserved and the functor commutes with composition.

13. TENSORIAL STRENGTH. The notion of a tensorial strength provides an alternative way of establishing the strongness of a functor via a first-order formulation. Moreover, it explicitly offers a way of distributing 'context', and therefore, dealing with extra parameters in a computation. We review the concept below and discuss its formulation for the kind of functors used in the thesis.

Let C be a category with finite products and T an endofunctor in C. T is said to have a (*tensorial*) *strength* if there exists a natural transformation

$$\tau^{\mathsf{T}}_r : \mathsf{T} \times - \Longrightarrow \mathsf{T}(\mathsf{Id} \times -)$$

called a *right strength*, or simply a *strength*, such that, for all $I, X, Y \in \mathrm{obj}(\mathsf{C})$, equations

$$\mathsf{Tl} \cdot \tau_r \ = \mathsf{l} \tag{12}$$

$$\mathsf{Ta}^\circ \cdot \tau_r \ = \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{a}^\circ \tag{13}$$

hold. These equations express the *unit* and the *associativity* law for τ_r and can be represented as

$$\begin{tikzcd}
\mathsf{T}I \times \mathbf{1} \arrow[r, "\tau^{\mathsf{T}}_{r_{I,\mathbf{1}}}"] \arrow[dr, "\mathsf{l}_{\mathsf{T}I}"'] & \mathsf{T}(I \times \mathbf{1}) \arrow[d, "\mathsf{T}(\mathsf{l}_{\mathsf{T}I})"] \\
& \mathsf{T}I
\end{tikzcd}$$

$$\begin{tikzcd}
\mathsf{T}I \times (X \times Y) \arrow[rr, "\tau^{\mathsf{T}}_{r_{I,X \times Y}}"] \arrow[d, "\mathsf{a}^\circ"'] & & \mathsf{T}(I \times (X \times Y)) \arrow[d, "\mathsf{Ta}^\circ"] \\
(\mathsf{T}I \times X) \times Y \arrow[r, "\tau^{\mathsf{T}}_{r_{I,X}} \times \mathsf{id}_Y"] & \mathsf{T}(I \times X) \times Y \arrow[r, "\tau^{\mathsf{T}}_{r_{I \times X, Y}}"] & \mathsf{T}((I \times X) \times Y)
\end{tikzcd}$$

Dually, a natural transformation $\tau_l^{\mathsf{T}} : - \times \mathsf{T} \Longrightarrow \mathsf{T}(- \times \mathsf{Id})$ satisfying similar laws is called a *left strength*. Assuming T implicit, note that $\tau_l = \mathsf{Ts} \cdot \tau_r \cdot \mathsf{s}$ and $\tau_r = \mathsf{Ts} \cdot \tau_l \cdot \mathsf{s}$. The unit and associativity axioms also hold for the *left strength*, in the dual form:

$$\mathsf{Tr} \cdot \tau_l \;= \mathsf{r} \tag{14}$$

$$\mathsf{Ta} \cdot \tau_l \;= \tau_l \cdot (\mathsf{id} \times \tau_l) \cdot \mathsf{a} \tag{15}$$

Note that τ_l^{T} (or its 'right' version) and σ^{T} can be mutually defined (see, *e.g.*, [Kel82] for a proof) by

$$\tau_l^{\mathsf{T}} \;=\; \mathsf{ev} \cdot (\sigma^{\mathsf{T}} \cdot \mathsf{sp} \times \mathsf{id}) \tag{16}$$

$$\sigma^{\mathsf{T}} \;=\; \mathsf{id}^{\mathsf{T}\;\mathsf{ev} \cdot \tau_l^{\mathsf{T}}} \cdot \mathsf{sp} \tag{17}$$

where sp and ev are, respectively, the unit and counit of the product-exponential adjunction.

14. A Strength Catalogue. In any distributive category all *polynomial* functors have a tensorial strength. Actually, this result extends to all *regular* functors (in particular, the strength for type functors was established in [Spe93]). For the former, however, a pointfree definition, by induction on the structure of polynomial functors, is as follows:

$$\begin{array}{ll}
\tau_{r_{I,X}}^{\underline{K}} = \pi_1 & : K \times X \to K \\
\tau_{r_{I,X}}^{\mathsf{Id}} = \mathsf{id}_{I \times X} & : I \times X \to I \times X \\
\tau_{r_{I,X}}^{\mathsf{T}\mathsf{T}'} = \mathsf{T}\tau_{r_{I,X}}^{\mathsf{T}'} \cdot \tau_{r_{\mathsf{T}'I,X}}^{\mathsf{T}} & : \mathsf{T}\,\mathsf{T}'I \times X \to \mathsf{T}\,\mathsf{T}'(I \times X)
\end{array}$$

and

$$\begin{array}{l}
\tau_{r_{I,X}}^{\mathsf{T}\times\mathsf{T}'} = (\tau_{r_{I,X}}^{\mathsf{T}} \times \tau_{r_{I,X}}^{\mathsf{T}'}) \cdot \langle \pi_1 \times \mathsf{id}_X, \pi_2 \times \mathsf{id}_X \rangle \\
\qquad : (\mathsf{T}I \times \mathsf{T}'I) \times X \to \mathsf{T}(I \times X) \times \mathsf{T}'(I \times X) \\
\tau_{r_{I,X}}^{\mathsf{T}+\mathsf{T}'} = (\tau_{r_{I,X}}^{\mathsf{T}} + \tau_{r_{I,X}}^{\mathsf{T}'}) \cdot \mathsf{dl}_{\mathsf{T}I,\mathsf{T}'I,X} \\
\qquad : (\mathsf{T}I + \mathsf{T}'I) \times X \to \mathsf{T}(I \times X) + \mathsf{T}'(I \times X)
\end{array}$$

Strength for functors Id^A and $\mathcal{P}$ is also easily defined:

$$\mathsf{T}I^A \times X \xrightarrow{\mathsf{sp}} (\mathsf{T}I^A \times X \times A)^A \xrightarrow{\mathsf{xr}^A} (\mathsf{T}I^A \times A \times X)^A$$
$$\xrightarrow{(\mathsf{ev}\times\mathsf{id})^A} (\mathsf{T}I \times X)^A \xrightarrow{\tau_{r_{I,X}}^{\mathsf{T}}} \mathsf{T}(I \times X)^A$$

and

$$\mathcal{P}\mathsf{T}I \times X \xrightarrow{\tau_{r_{\mathsf{T}I,X}}^{\mathcal{P}}} \mathcal{P}(\mathsf{T}I \times X) \xrightarrow{\mathcal{P}\tau_{r_{I,X}}^{\mathsf{T}}} \mathcal{P}\mathsf{T}(I \times X)$$

where, $\tau_{r_{I,X}}^{\mathcal{P}} : \mathcal{P}I \times X \longrightarrow \mathcal{P}(I \times X)$ is defined in the obvious way:

$$\lambda \langle c, x \rangle \,.\; \{\langle i, x \rangle |\; i \in c\}$$

15. Strong Natural Transformation. A natural transformation $\phi : \mathsf{T} \Longrightarrow \mathsf{R}$ between strong functors is called *strong* iff it verifies

$$\phi \cdot \tau_r^{\mathsf{T}} \;= \tau_r^{\mathsf{R}} \cdot (\phi \times \mathsf{id}) \tag{18}$$

It is known that, regarded as natural transformations, the identity, projections, and universal arrows to (respectively, from) the final (respectively, initial) objects in a category are all strong. The same applies to products, splits and eithers of strong natural transformations, as well as to the pre- or post-composition of such a transformation with a strong functor.

16. Composing Monads. In what conditions can two monads be composed to yield a new monad? The answer to this question is not trivial because the simple composition of the corresponding functors does not lead, in general, to new monads. The problem is, of course, in the way μ is applied. This entails the need to assert the existence of some sort of distributive law to alter the order of functor application. A general result on this topic can be found in [BW85]. Basically it is proved that given two monads B_1 and B_2 a new monad based on functor $\mathsf{B}_1\ \mathsf{B}_2$ can be defined, provided there exists a natural transformation $\gamma : \mathsf{B}_2\ \mathsf{B}_1 \longrightarrow \mathsf{B}_1\ \mathsf{B}_2$ satisfying a number of conditions. If this is the case, unit and multiplication of the composed monad are given, respectively, by

$$\eta^{\mathsf{B}_1\ \mathsf{B}_2} = \eta^{\mathsf{B}_1} \cdot \eta^{\mathsf{B}_2} \tag{19}$$

$$\mu^{\mathsf{B}_1\ \mathsf{B}_2} = \mathsf{B}_1\ \mu^{\mathsf{B}_2} \cdot \mu^{\mathsf{B}_1} \cdot \mathsf{B}_1\ \gamma \tag{20}$$

Instead of detailing the general conditions on γ, we shall record here three particular cases which are relevant in the context of this thesis. It should also be mentioned the existence of several other results on monad composition, developed mainly in the context of functional programming (see, for example, [KW92]).

- The *exception* monad (and, of course, the *partiality* monad which is a specialisation of the latter) can be composed with any other monad B, originating $\mathsf{B}\ (\mathsf{Id} + E)$. The distributive law γ is, in this case, defined by

$$\gamma \;=\; \mathsf{B} + E \xrightarrow{[\mathsf{B}\ \iota_1, \eta^{\mathsf{B}} \cdot \iota_2]} \mathsf{B}\ (\mathsf{Id} + E)$$

- The same applies to a monad generated by a monoid M, if B strong. The resulting monad is then $\mathsf{B}\ (M \times \mathsf{Id})$, with

$$\gamma \;=\; M \times \mathsf{B}M \xrightarrow{\tau_l^{\mathsf{B}}} \mathsf{B}\ (M \times \mathsf{Id})$$

- Assuming again B strong, it can be composed with the exponential monad, originating B^S, with

$$\gamma \;=\; \mathsf{B}\ (\mathsf{Id}^S) \xrightarrow{\mathsf{sp}} (\mathsf{B}\ (\mathsf{Id}^S) \times S)^S \xrightarrow{(\tau_r^{\mathsf{B}})^S} (\mathsf{B}\ (\mathsf{Id}^S \times S))^S \xrightarrow{(\mathsf{B}\ \mathsf{ev})^S} \mathsf{B}^S$$

Finally, notice that, if B is a commutative monad, so are $\mathsf{B}\ (M \times \mathsf{Id})$, B^S and $\mathsf{B}\ (\mathsf{Id} + \mathbf{1})$. Of course the same is not true for the general exception monad which is itself non commutative.

17. Monad Morphisms. A morphism between two (strong) monads $\langle \mathsf{B}, \eta, \mu \rangle$ and $\langle \mathsf{B}', \eta', \mu' \rangle$, is a strong natural transformation $\psi : \mathsf{B} \Longrightarrow \mathsf{B}'$ such that

$$\psi \cdot \eta = \eta' \tag{21}$$

$$\psi \cdot \mu = \mu' \cdot \psi \cdot \mathsf{B}\ \psi \tag{22}$$

These laws, whose diagrams are shown below, are just the 'homomorphism' conditions to guarantee that the monad structure is preserved along the morphism. Notice, in the μ diagram, how a horizontal composition of ψ with itself is decomposed.

$$\begin{tikzcd}
\mathsf{Id}_{\mathsf{C}} \arrow[r, "\eta^{\mathsf{B}}"] \arrow[d, equal] & \mathsf{B} \arrow[d, "\psi"] \\
\mathsf{Id}_{\mathsf{C}} \arrow[r, "\eta^{\mathsf{B}'}"'] & \mathsf{B}'
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{B} \arrow[d, "\psi"'] & \mathsf{B}^2 \arrow[l, "\mu^{\mathsf{B}}"'] \arrow[r, "\mathsf{B}\psi"] \arrow[d, "\psi;\psi"] & \mathsf{B}\mathsf{B}' \arrow[dl, "\psi"] \\
\mathsf{B}' & \mathsf{B}'^2 \arrow[l, "\mu^{\mathsf{B}'}"] &
\end{tikzcd}$$

Being strong also implies that

$$\psi \cdot \tau_r \ = \tau_r^{\mathsf{B}'} \cdot (\psi \times \mathsf{id}) \tag{23}$$

An example of a monad morphism from the *sequence* monad to itself is the usual *list reverse* function. As a final remark, note that monad morphisms compose (*cf.*, vertical composition of natural transformations) and there exists, for each monad, a trivial identity morphism. Therefore, monads and their morphisms form a category. Moreover it is immediate to show that the *identity* monad Id is initial in such a category, with $?_{\mathsf{B}} : \mathsf{Id} \longrightarrow \mathsf{B}$ defined as η^{B}, for any other monad B. Similarly, the category has as final object the monad over constant functor $\underline{\mathbf{1}}$.

References

[BW85] M. Barr and C. Wells. *Toposes, Triples and Theories.* Springer-Verlag, 1985.

[CF92] R. Cockett and T. Fukushima. About Charity. Yellow Series Report No. 92/480/18, Dep. Computer Science, University of Calgary, June 1992.

[CS92] R. Cockett and D. Spencer. Strong categorical datatypes I. In R. A. G. Seely, editor, *Proceedings of Int. Summer Category Theory Meeting, Montréal, Québec, 23–30 June 1991*, pages 141–169. AMS, CMS Conf. Proceedings 13, 1992.

[CS95] R. Cockett and D. Spencer. Strong categorical datatypes II: A term logic for categorical programming. *Theor. Comp. Sci.*, 139:69–113, 1995.

[Kel82] G. M. Kelly. *Basic Concepts of Enriched Category Theory*, volume 64 of *London Mathematical Society Lecture Notes Series.* Cambridge University Press, 1982.

[Koc72] A. Kock. Strong functors and monoidal monads. *Archiv für Mathematik*, 23:113–120, 1972.

[KW92] D. King and P. Wadler. Combining monads. In *Proc. 5th Annual Glasgow Workshop on Functional Programming*, 1992.

[Mac71] S. Mac Lane. *Categories for the Working Mathematician.* Springer Verlag, 1971.

[Mog89] E Moggi. Computational lambda-calculus and monads. In *Proceedings of the Logic in Computer Science Conference*, 1989.

[Mog91] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[Spe93] D. L. Spencer. *Categorical Programming with Functorial Strength.* PhD thesis, The Oregon Graduate Institute of Science and Technology, Janeiro 1993.

[Wad92] P. Wadler. Comprehending monads. *Math. Struct. in Comp. Sci.*, 2:461–493, 1992. (Special issue of selected papers from 6'th Conference on Lisp and Functional Programming.).

[Wad95] P. Wadler. Monads for functional programming. In J. Jeuring and E. Meijer, editors, *Advanced Functional Programming.* Springer Lect. Notes Comp. Sci. (925), 1995.